FLEXI Project Documentation

Numerics Research Group
Institute for Aerodynamics and Gas Dynamics
University of Stuttgart, Germany

June 6, 2019
# Contents

1 Introduction 5  
   1.1 How to use the user guide 5

2 Getting started 7  
   2.1 Installation 7  
      2.1.1 Prerequisites 7  
      2.1.2 Obtaining the source 8  
      2.1.3 Compiling the code 9  
      2.1.4 Running the code 10  
   2.2 Basic Usage 10  
      HOPR 11  
      FLEXI 11  
      Parameter file 11  
      posti_visu tool 12  
      HDF5 12  
   2.3 Feature list 12

3 Code options 14  
   3.1 Compiler options 14  
   3.2 Parameter file options 17

4 Workflow 27  
   4.1 Mesh generation using HOPR 27  
   4.2 Solver settings 27  
   4.3 Initial and boundary conditions 28  
      4.3.1 Initial conditions 29  
      4.3.2 Boundary conditions 29  
   4.4 Material properties 30  
   4.5 Output time interval 31  
   4.6 Evaluation during runtime 31  
   4.7 Parallel execution 31  
      4.7.1 Domain decomposition 32  
      4.7.2 Choosing the number of cores 32  
   4.8 Test case environment 32  
   4.9 Post processing / posti_visu tool 33
5 Tools Overview

5.1 POSTI

- Visualization ........................................... 35
- Swap meshes ........................................... 36
- Record points ......................................... 36
- Time averaging ........................................ 37
- Test cases ............................................. 37

5.2 tools folder

- animate .................................................. 39
- convergence_test ...................................... 40
- userblock .............................................. 40
- sortfiles.sh ............................................ 41
- getload.py ............................................. 41
- testcases .............................................. 41

6 Tutorials

6.1 Freestream ............................................ 42

- Mesh Generation with HOPR .......................... 42
- Flow Simulation with FLEXI .......................... 43
- Numerical settings .................................... 44

6.2 Lid-driven Cavity ...................................... 45

- Flow description ...................................... 46
- Compiler options ...................................... 46
- Part I: Basic Tutorial - Flow at $Re = 100$ ...... 47
- Part II: Advanced Tutorial - Flow at $Re = 400$ .. 58

6.3 Flow around a NACA 0012 airfoil ................. 63

- Compiler options ...................................... 64
- Mesh Generation with HOPR .......................... 65
- Flow Simulation with FLEXI .......................... 65
- Numerical settings .................................... 66
- Boundary conditions .................................. 66
- Running the code ..................................... 68
- Evaluation of the lift and drag forces .......... 68
- Wall velocities ......................................... 69
- Visualization .......................................... 69
- Remove outflow reflections using the sponge zone 70
- Restarting the simulation ............................ 72
- Two-dimensional computation ....................... 73
- Using record points .................................. 75

6.4 Convergence Test ..................................... 78

- Manufactured Solution ............................... 79
- Inviscid Convergence Test ............................ 79
- Viscous Convergence Test ............................ 82
- Execution of the Convergence Test with non-conform meshes 84
1 Introduction

FLEXI is a high-order discontinuous Galerkin (DG) simulation code for the solution of the time-dependent compressible Navier-Stokes equations on unstructured hexahedral elements in three space dimensions. The code was specifically designed for very high order accurate simulations on massively parallel systems. It is licensed under GPLv3, written in Fortran and parallelized with MPI. Implemented features are

- Arbitrary order nodal polynomial tensor product basis using Gauss or Gauss Lobatto collocation points
- Various Riemann solvers for inter-element coupling
- Large eddy simulation capabilities through different de-aliasing strategies and subgrid scale models
- Split form DG formulation with various kinetic energy or entropy preserving flux formulations
- Matching high order curved mesh generation from external mesh formats (CGNS, GMSH) or simple analytic blocks via the open source preprocessor HOPR
- Nonconforming interfaces based on the mortar approach
- Non-reflecting boundary conditions and damping zones for direct aeroacoustic computations
- Automatic domain decomposition for parallel simulations based on a space filling curve
- High order explicit Runge-Kutta time integration
- I/O using the HDF5 library optimized for massively parallel jobs
- Shock capturing employing finite volume subcells

1.1 How to use the user guide

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 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 lists the most important compiler and runtime options.
- Chapter 4 is meant as a complete user guide with a detailed description how to use and apply the features of FLEXI from a user’s point of view. This includes setting up
1.1. *HOW TO USE THE USER GUIDE*  

Chapter 1. *Introduction*

solver settings, initial and boundary conditions, the mesh interface, parallel execution and the currently available post processing capabilities.

- Chapter 5 lists all tools contained in the **FLEXI** repository, including **POSTI** post-processing tools.
- Simulation tutorials are contained in Chapter 6.
- The unit test system used to test key routines with CTest is described in Chapter 7.
2 Getting started

2.1 Installation

2.1.1 Prerequisites

**FLEXI** has been tested for various Linux distributions. This includes Ubuntu 14.04 LTS, 16.04 LTS and 18.04 LTS, OpenSUSE 42.1 and CentOS 7. The suggested packages in this section can of course be replaced by self compiled versions.

The required packages for the Ubuntu Linux distributions are listed in table 2.1. Under Ubuntu, they can be obtained using the apt environment:

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

<table>
<thead>
<tr>
<th>Package</th>
<th>Ubuntu 14.04</th>
<th>Ubuntu 16.04</th>
<th>Ubuntu 18.04</th>
</tr>
</thead>
<tbody>
<tr>
<td>git</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>cmake</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>cmake-curses-gui</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>liblapack3</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>liblapack-dev</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>gfortran</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>g++</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>mpi-default-dev</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>zlib1g-dev</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>exuberant-ctags</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

The required packages for OpenSUSE and CentOS are listed in table 2.2. Under OpenSUSE, packages are installed by the following command.

```
sudo zypper install git
```

The ```PATH``` variable must be extended by the openmpi path

```
export PATH=$PATH:/usr/lib64/mpi/gcc/openmpi/bin
```
2.1. INSTALLATION

Chapter 2. Getting started

Under CentOS, packages are installed by the following command.

```sh
sudo yum install git
```

Additionally, the PATH variable must be extended by the openmpi path

```sh
export PATH=$PATH:/usr/lib64/openmpi/bin
```

<table>
<thead>
<tr>
<th>Package</th>
<th>OpenSUSE 42.1</th>
<th>CentOS 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>git</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>cmake</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>lapack-devel</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>openmpi</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>openmpi-devel</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>zlib-devel</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>gcc-fortran</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>gcc</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>gcc-c++</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ctags-etaags</td>
<td>-</td>
<td>o</td>
</tr>
</tbody>
</table>

Table 2.2: OpenSUSE/CentOS packages. x: required, o: optional, -: not available

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 FLEXI correctly. This is done using the command

```sh
ulimit -s unlimited
```

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

2.1.2 Obtaining the source

The FLEXI repository is available at GitHub. To obtain the most recent version you have two possibilities:

- Clone the FLEXI repository from Github

  ```sh
git clone https://github.com/flexi-framework/flexi.git
```

- Download FLEXI from Github:

  ```sh
  wget https://github.com/flexi-framework/flexi/archive/master.tar.gz
tar xzf master.tar.gz
  ```
2.1. INSTALLATION  

Chapter 2. Getting started

Note that cloning FLEXI from GitHub may not be possible on some machines, as e.g. the HLRS at the University of Stuttgart restricts internet access. Please refer to section 8.1 of this user guide.

2.1.3 Compiling the code

- Open a terminal
- Change into the FLEXI directory
- Create a new subdirectory and use CMake to configure and compile the code

```bash
mkdir build; cd build
cmake ../
make
```

The executables flexi and posti_visu are contained in your FLEXI directory in build/bin/.

Custom configuration of compiler options may be done using

```bash
ccmake ../
```

For a list of all compiler options see Section 3.1.

2.1.3.1 Directory paths

In the following, we write $FLEXIROOT as a substitute for the path to the FLEXI repository. Please replace $FLEXIROOT in all following commands with the path to your FLEXI repository or add an environment variable $FLEXIROOT.

Furthermore, the path to executables is omitted in the following, so for example, we write flexi instead of $FLEXIROOT/build/bin/flexi.

Here is some explanation for Linux beginners:

In order to execute a file, you have to enter the full path to it in the terminal. There are two different ways to enable typing flexi instead of the whole path (do not use both at the same time!)

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

```bash
alias flexi='FLEXIROOT/build/bin/flexi'
```

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

```bash
. ~/.bashrc
```

2. You can add the FLEXI binary directory to your $PATH environment variable by adding
2.2. BASIC USAGE

For a first minimal FLEXI run, do the following:

- Open a terminal
- Navigate to a directory, in this case temp
  ```bash
  cd temp
  ```
- Copy the cavity tutorial folder
  ```bash
  cp -r $FLEXIROOT/tutorials/cavity/Basic_Re100 .
  cd Basic_Re100
  ```
- Run flexi
  ```bash
  flexi parameter_flexi.ini
  ```
- Convert the output files to the vtu format
  ```bash
  posti_visu cavity_State_0000000.200000000.h5
  ```
- Visualize using e.g. ParaView.

2.2 Basic Usage

For a basic overview of the framework and the single components of the flow solver a flowchart is given in Figure 2.1.

![Flowchart: Basic modules and files used by FLEXI](image)
HOPR

A standalone high-order preprocessor HOPR has been developed to generate high-order meshes from input data from external linear mesh generators. Different file formats are supported. HOPR has been recently made open source under the GPLv3 license. It generates a FLEXI conform mesh format in HDF5 for efficient parallel initialization. For a complete overview of HOPR, see https://www.hopr-project.org.

HOPR can be compiled in the same way as FLEXI (see section 2.1.3). The basic command to run HOPR is

```
hopr parameter.ini
```

Note that the path to the HOPR executable is omitted in the command (see 2.1.3).

FLEXI

FLEXI, a high order DGSEM based CFD solver, is the core module in the tool chain. Generally FLEXI requires two main files as input, a mesh file in HDF5 format generated by HOPR and a parameter file where the main settings for the CFD simulation are set. The results files generated by FLEXI are also HDF5 files.

The basic command to run FLEXI is

```
mpirun -np [no. processors] flexi parameter.ini
```

Note: Adding `mpirun -np [no. processors]` before the FLEXI executable starts FLEXI in parallel with the specified number of threads. If it is omitted, FLEXI is run on one processor without MPI. This also applies to all other tools mentioned below.

Parameter file

The `parameter.ini` file contains the main settings for the CFD simulation. The parameter file defines e.g.

- CFL (Courant-Friedrichs-Lewy) number
- polynomial degree,
- simulation end time and dump/analyze intervals
- boundary conditions
- initial and boundary states

A complete list of all runtime options that can be set in the parameter file can be obtained with the command

```
flexi --help
```
2.3. FEATURE LIST

It is also supplied in Section 3.2. The `--help` option also works for most other FLEXI tools.

**posti_visu tool**

To visualize the results e.g. with ParaView, a converter tool is provided. The posti_visu tool takes the HDF5 files generated by FLEXI.

The basic command to run the posti_visu tool is

```
mpirun -np [no. processors] posti_visu parameter.ini [flexi_outputfile.h5]
```

In this case a parameter file is specified in which options like the type and amount of the visualization nodes and mesh options are defined - see Section 4.9 for all available options. You can also omit the parameter file argument:

```
mpirun -np [no. processors] posti_visu [flexi_outputfile.h5]
```

This runs posti_visu using only standard options, i.e.

- equidistant visualization nodes
- amount of visualization nodes equals number of collocation points per element
- allowing for curved meshes
- visualizing the conservative variables

**HDF5**

HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data. For further information and to download the software, visit the HDF5 website at https://www.hdfgroup.org.

2.3 Feature list

The currently implemented features of FLEXI include

- Equation systems:
  - compressible Euler equations
  - compressible Navier-Stokes equations
  - linear scalar advection and diffusion
- Space discretization: DGSEM method (David A. Kopriva and Gassner 2010; Hindenlang et al. 2012)
  - Legendre Gauss
  - Legendre Gauss Lobatto
2.3. **FEATURE LIST**

- **Time discretization**: explicit Runge-Kutta methods
  - standard RK methods
  - low storage RK methods (Carpenter and Kennedy 1994)
  - strong stability preserving RK methods (Niegemann, Diehl, and Busch 2012)
- **Two- or three-dimensional domains**
- **Riemann solvers**:
  - local Lax-Friedrichs
  - HLL
  - HLLC
  - Roe-Pike
- **Curved Meshes**
- **Nonconforming Meshes via mortar interfaces** (David A Kopriva, Woodruff, and Hussaini 2002)
- **Shock capturing**
  - Employing finite volume subcells
  - Several shock indicators available
- **Boundary conditions**
  - Various subsonic inflow and outflow conditions (Carlson 2011)
  - exact boundaries (Dirichlet)
  - periodic boundaries
  - slip wall (Euler wall)
  - non-slip walls (Navier-Stokes wall)
    * adiabatic
    * isothermal
- **Splitform discontinuous Galerkin schemes** (G. J. Gassner, Winters, and Kopriva 2016)
- **Dealiasing** (Gassner and Beck 2013)
  - filtering
  - overintegration
- **Lifting methods**
  - Bassi Rebay 1 (Bassi and Rebay 1997)
  - Bassi Rebay 2 (Bassi and Rebay 1997)
- **Sponge zone** (Flad et al. 2014)
- **Time averaging**
3 Code options

3.1 Compiler options

This section describes the main configuration options which can be set when building FLEXI using CMake. Some options are dependent on others being enabled (or disabled), so the available ones may change.

The first set of options describe general CMake behaviour:

- **CMAKE_BUILD_TYPE**: This statically specifies what build type (configuration) will be built in this build tree. Possible values are
  - **Release**
    “Normal” execution.
  - **Profile**
    Performance profiling using gprof.
  - **Debug**
    Debug compiler for detailed error messages during code development.

- **CMAKE_HOSTNAME**: This will display the host name of the machine you are compiling on.

- **CMAKE_INSTALL_PREFIX**: If “make install” is invoked or INSTALL is built, this directory is prepended onto all install directories. This variable defaults to /usr/local on UNIX.

For some external libraries and programs that FLEXI uses, the following options apply:

- **CTAGS_PATH**: This variable specifies the Ctags install directory, an optional program used to jump between tags in the source file.

- **FLEXI_BUILD_HDF5**: ON/OFF
3.1. COMPILER OPTIONS

This will be set to ON if no prebuilt HDF5 installation was found on your machine. In this case a HDF5 version will be built and used instead.

- **HDF5_DIR**:
  If you want to use a prebuilt HDF5 library that has been built using the CMake system, this directory should contain the CMake configuration file for HDF5 (optional).

The following options enable or disable specific features of FLEXI. If you want to use a certain feature, make sure to enable it during the build process!

- **FLEXI_2D**: ON/OFF
  If set to ON the code will run in two-dimensional mode. You have to provide a mesh that consists of only one layer of elements in the third dimension.

- **FLEXI_EQNSYSNAME**:
  This variable defines the equation system, which will be compiled and used for the simulation. Possible values are
  - Navierstokes
  - Linearscalaradvection

- **FLEXI_FV**: ON/OFF
  Set this to ON to enable the usage of the finite volume subcell shock capturing mechanism.

- **FLEXI_FV_RECONSTRUCTION**: ON/OFF
  Only available if FLEXI_FV is set to ON. Enables the reconstruction of interface values in the finite volume subcells. Needed for calculation of gradients and to use a second order finite volume scheme.

- **FLEXI_LIFTING**:
  Two different lifting methods for the parabolic part of the equation system are implemented. Possible values are
  - BR1: First method of Bassi and Rebay (Bassi and Rebay 1997)
  - BR2: Second method of Bassi and Rebay (Bassi et al. 1997)

- **FLEXI_MKL**: ON/OFF
  This flag defines, whether Intel’s MKL (Math Kernel Library) should be used. This is only meaningful when FLEXI is compiled with Intel compiler.

- **FLEXI_MPI**: ON/OFF
  This flag defines, whether FLEXI is compiled with MPI (necessary for parallel execution).

- **FLEXI_NODETYPE**:
**FLEXI** space discretization is based on a DG method. Here two different basis functions could be used, see (David A. Kopriva and Gassner 2010) for details.

- **GAUSS**
- **GAUSS-LOBATTO**

- **FLEXI_PAPI**: ON/OFF
  
  Enable to use the PAPI library to perform performance measurements (e.g. flop counts).

- **FLEXI_PARABOLIC**: ON/OFF
  
  This variable defines, whether the parabolic part of the chosen system should be included or not. Practically, this corresponds to setting the viscosity or diffusivity coefficient to zero, but is much more efficient, since the lifting routines are not necessary.

- **FLEXI_POLYNOMIAL_DEGREE**:

  Since **FLEXI** is a high order CFD solver based on polynomial basis function, the polynomial degree can already be chosen in the compile process. If the default value N is chosen, different polynomial degrees can be defined later in the parameter file.

  - N
  - #

- **FLEXI_SPLIT_DG**: ON/OFF

  Enable to use the split form of the discontinuous Galerkin operator. Allows to use kinetic energy or entropy stable flux functions. Only available for Gauss-Lobatto nodes.

- **FLEXI_TESTCASE**:

  Some specific (and often used) simulation setups are encapsulated in test cases. These include e.g. case-specific initialization, analyze routines, boundary conditions etc. The default test case does not include any additions.

  - default
  - taylorgreenvortex
  - phill
  - channel
  - riemann2d

  See section 4.8 for details.

- **FLEXI_VISCOSITY**:

  There are different modeling approaches for the viscosity in **FLEXI**. You can choose

  - constant
  - sutherland
  - powerlaw

The remaining part of the options deal with the post-processing framework **POSTI**.
3.2. PARAMETER FILE OPTIONS

- **FLEXI_BUILDPOSTI**: ON/OFF
  
  Enable to also build the post-processing tools next to the actual simulation software. When this general option is enabled, you will need to enable the specific options for the tools that should be built.

- **POSTI_BUILD_***: ON/OFF
  
  Each of the **POSTI** tools has its own build option. Enable to build this specific tool.

- **POSTI_USE_PARAVIEW***: ON/OFF
  
  Enable to build the ParaView plugin for visualization of **FLEXI** simulation data. The ParaView libraries etc. must be available on the system and the environment variable $ParaView_DIR set accordingly.

### 3.2 Parameter file options

A *parameter.ini* file is needed to control the code. An overview of all options in the parameter file can be generated by following command in the terminal:

```
flexi --help
```

Generally following types are used:

```
INTEGER = 1
REAL = 1.23456
LOGICAL = T ! True
LOGICAL = ! False
STRING = FLEXI
VECTOR = (/1.0,2.0,3.0/)
```

The concept of the parameter file is described as followed:

- each single line is saved and examined for specific variable names
- the examination is case-insensitive
- comments can be set with symbol "!" in front of the text

```
! commented text
```

- numbers can also be set by using "pi"

```
vector = (/1,2Pi,3Pi/)
```

- the order of defined variables is with one exception generally indifferent, but it is preferable to group similar variables
- the order becomes important only by modifying boundary conditions, if you want to modify a specific boundary by addressing its name, the related boundary type has to be defined
3.2. PARAMETER FILE OPTIONS

The following tables describe the main configuration options which can be used in the parameter file.

### MPI

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupSize</td>
<td>0</td>
<td>Define size of MPI subgroups, used to e.g. perform grouped IO, where group master collects and outputs data.</td>
</tr>
</tbody>
</table>

### IO_HDF5

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatheredWrite</td>
<td>F</td>
<td>Set true to activate gathered HDF5 IO for parallel computations. Only local group masters will write data after gathering from local slaves.</td>
</tr>
</tbody>
</table>

### Interpolation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
<td>Polynomial degree of computation to represent to solution</td>
</tr>
</tbody>
</table>

### Restart

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResetTime</td>
<td>F</td>
<td>Override solution time to t=0 on restart.</td>
</tr>
</tbody>
</table>

### Output

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVisu</td>
<td></td>
<td>Polynomial degree at which solution is sampled for visualization.</td>
</tr>
<tr>
<td>NOut</td>
<td>-1</td>
<td>Polynomial degree at which solution is written. -1: NOut=N, &gt;0: NOut</td>
</tr>
<tr>
<td>ProjectName</td>
<td></td>
<td>Name of the current simulation (mandatory).</td>
</tr>
</tbody>
</table>
### 3.2. PARAMETER FILE OPTIONS

#### Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
<td>F</td>
<td>Write log files containing debug output.</td>
</tr>
<tr>
<td>ErrorFiles</td>
<td>T</td>
<td>Write error files containing error output.</td>
</tr>
<tr>
<td>OutputFormat</td>
<td>None</td>
<td>File format for visualization: None, Tecplot, TecplotASCII, ParaView. Note: Tecplot output is currently unavailable due to licensing issues.</td>
</tr>
<tr>
<td>ASCIIOutputFormat</td>
<td>CSV</td>
<td>File format for ASCII files, e.g. body forces: CSV, Tecplot.</td>
</tr>
<tr>
<td>doPrintStatusLine</td>
<td>F</td>
<td>Print: percentage of time, ...</td>
</tr>
<tr>
<td>WriteStateFiles</td>
<td>T</td>
<td>Write HDF5 state files. Disable this only for debugging issues. NO SOLUTION WILL BE WRITTEN!</td>
</tr>
</tbody>
</table>

#### Mesh

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshFile</td>
<td></td>
<td>(relative) path to meshfile (mandatory).</td>
</tr>
<tr>
<td>useCurveds</td>
<td>T</td>
<td>Controls usage of high-order information in mesh. Turn off to discard high-order data and treat curved meshes as linear meshes.</td>
</tr>
<tr>
<td>interpolateFromTree</td>
<td>T</td>
<td>For non-conforming meshes, built by refinement from a tree structure, the metrics can be built from the tree geometry if it is contained in the mesh. Can improve free-stream preservation.</td>
</tr>
<tr>
<td>meshScale</td>
<td>1.0</td>
<td>Scale the mesh by this factor (shrink/enlarge).</td>
</tr>
<tr>
<td>meshdeform</td>
<td>F</td>
<td>Apply simple sine-shaped deformation on cartesian mesh (for testing).</td>
</tr>
<tr>
<td>crossProductMetrics</td>
<td>F</td>
<td>Compute mesh metrics using cross product form. Caution: in this case free-stream preservation is only guaranteed for N=3*NGeo.</td>
</tr>
<tr>
<td>debugmesh</td>
<td>0</td>
<td>Output file with visualization and debug information for the mesh. 0: no visualization, 3: Paraview binary</td>
</tr>
<tr>
<td>BoundaryName</td>
<td></td>
<td>Names of boundary conditions to be set (must be present in the mesh!). For each BoundaryName a BoundaryType needs to be specified.</td>
</tr>
<tr>
<td>BoundaryType</td>
<td></td>
<td>Type of boundary conditions to be set. Format: (BC_TYPE,BC_STATE)</td>
</tr>
</tbody>
</table>
### Mesh

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>writePartitionInfo</td>
<td>F</td>
<td>Write information about MPI partitions into a file.</td>
</tr>
</tbody>
</table>

### Equation of State

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseNonDimensionalEqn</td>
<td>F</td>
<td>Set true to compute $R$ and $\mu$ from bulk Mach Reynolds (nondimensional form)</td>
</tr>
<tr>
<td>BulkMach</td>
<td></td>
<td>Bulk Mach ($\text{UseNonDimensionalEqn}=T$)</td>
</tr>
<tr>
<td>BulkReynolds</td>
<td></td>
<td>Bulk Reynolds ($\text{UseNonDimensionalEqn}=T$)</td>
</tr>
<tr>
<td>kappa</td>
<td>1.4</td>
<td>Heat capacity ratio / isentropic exponent</td>
</tr>
<tr>
<td>$R$</td>
<td>287.058</td>
<td>Specific gas constant</td>
</tr>
<tr>
<td>$Pr$</td>
<td>0.72</td>
<td>Prandtl number</td>
</tr>
<tr>
<td>$\mu_0$</td>
<td>0.0</td>
<td>Dynamic Viscosity</td>
</tr>
<tr>
<td>$T_s$</td>
<td>110.4</td>
<td>Sutherland’s law for variable viscosity: $T_s$</td>
</tr>
<tr>
<td>$T_{ref}$</td>
<td>280.0</td>
<td>Sutherland’s law for variable viscosity: $T_{ref}$</td>
</tr>
<tr>
<td>ExpoSuth</td>
<td>1.5</td>
<td>Sutherland’s law for variable viscosity: Exponent</td>
</tr>
</tbody>
</table>

### Equation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IniRefState</td>
<td></td>
<td>Refstate required for initialization.</td>
</tr>
<tr>
<td>RefState</td>
<td></td>
<td>State(s) in primitive variables (density, velx, vely, velz, pressure).</td>
</tr>
<tr>
<td>BCStateFile</td>
<td></td>
<td>File containing the reference solution on the boundary to be used as BC.</td>
</tr>
</tbody>
</table>

### Riemann

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Riemann</td>
<td>RoeEntropyFix</td>
<td>Riemann solver to be used: LF, HLLC, Roe, RoeEntropyFix, HLL, HLLE, HLLEM</td>
</tr>
<tr>
<td>RiemannBC</td>
<td>Same</td>
<td>Riemann solver used for boundary conditions: Same, LF, Roe, RoeEntropyFix, HLL, HLLE, HLLE</td>
</tr>
</tbody>
</table>

### SplitDG

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
### 3.2. PARAMETER FILE OPTIONS

#### SplitDG

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SplitDG</td>
<td>PI</td>
<td>SplitDG formulation to be used: SD, MO, DU, KG, PI</td>
</tr>
</tbody>
</table>

#### Exactfunc

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IniExactFunc</td>
<td></td>
<td>Exact function to be used for computing initial solution.</td>
</tr>
<tr>
<td>AdvVel</td>
<td></td>
<td>Advection velocity ((v_1,v_2,v_3)) required for exactfunction CASE(2,21,4,8)</td>
</tr>
<tr>
<td>MachShock</td>
<td>1.5</td>
<td>Parameter required for CASE(10)</td>
</tr>
<tr>
<td>PreShockDens</td>
<td>1.0</td>
<td>Parameter required for CASE(10)</td>
</tr>
<tr>
<td>IniCenter</td>
<td></td>
<td>Shu Vortex CASE(7) ((x,y,z))</td>
</tr>
<tr>
<td>IniAxis</td>
<td></td>
<td>Shu Vortex CASE(7) ((x,y,z))</td>
</tr>
<tr>
<td>IniAmplitude</td>
<td>0.2</td>
<td>Shu Vortex CASE(7)</td>
</tr>
<tr>
<td>IniHalfwidth</td>
<td>0.2</td>
<td>Shu Vortex CASE(7)</td>
</tr>
<tr>
<td>delta99_in</td>
<td></td>
<td>Blasius boundary layer CASE(1338)</td>
</tr>
<tr>
<td>x_in</td>
<td></td>
<td>Blasius boundary layer CASE(1338)</td>
</tr>
</tbody>
</table>

#### Filter

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FilterType</td>
<td>None</td>
<td>Type of filter to be applied. None, CutOff, Modal, LAF</td>
</tr>
<tr>
<td>NFilter</td>
<td></td>
<td>Cut-off mode (FilterType=CutOff or LAF)</td>
</tr>
<tr>
<td>LAF_alpha</td>
<td></td>
<td>Relaxation factor for LAF, see Flad et al. JCP 2016</td>
</tr>
<tr>
<td>HestFilterParam</td>
<td></td>
<td>Parameters for Hesthaven filter (FilterType=Modal)</td>
</tr>
</tbody>
</table>

#### Overintegration

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OverintegrationType</td>
<td>none</td>
<td>Type of overintegration. None, CutOff, ConsCutOff</td>
</tr>
<tr>
<td>NUnder</td>
<td></td>
<td>Polynomial degree to which solution is filtered (OverintegrationType = 1 or 2</td>
</tr>
</tbody>
</table>
### 3.2. PARAMETER FILE OPTIONS

#### Chapter 3. Code options

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDG</td>
<td>DG</td>
<td></td>
<td>Specify type of indicator to be used: DG, FV, Persson, Ducros, halfhalf, checkerboard</td>
</tr>
<tr>
<td>IndVar</td>
<td>1</td>
<td></td>
<td>Specify variable upon which indicator is applied, for general indicators.</td>
</tr>
<tr>
<td>IndStartTime</td>
<td>0.0</td>
<td></td>
<td>Specify physical time when indicator evaluation starts. Before this time a high indicator value is returned from indicator calculation. (Idea: FV everywhere at begin of computation to smooth solution)</td>
</tr>
<tr>
<td>nModes</td>
<td>2</td>
<td></td>
<td>Number of highest modes to be checked for Persson modal indicator.</td>
</tr>
<tr>
<td>FVBoundaries</td>
<td>F</td>
<td></td>
<td>Use FV discretization in element that contains a side of a certain BC_TYPE</td>
</tr>
<tr>
<td>FVBoundaryType</td>
<td>BC_TYPE</td>
<td></td>
<td>Set it to BC_TYPE, setting 0 will apply FV to all BC Sides</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FV</th>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FV_IndUpperThreshold</td>
<td></td>
<td></td>
<td>Upper threshold: Element is switched from DG to FV if indicator rises above this value</td>
</tr>
<tr>
<td>FV_IndLowerThreshold</td>
<td></td>
<td></td>
<td>Lower threshold: Element is switched from FV to DG if indicator falls below this value</td>
</tr>
<tr>
<td>FV_toDG_indicator</td>
<td>F</td>
<td></td>
<td>Apply additional Persson indicator to check if DG solution after switch from FV to DG is valid.</td>
</tr>
<tr>
<td>FV_toDG_limit</td>
<td></td>
<td></td>
<td>Threshold for FV_toDG_indicator</td>
</tr>
<tr>
<td>FV_toDGinRK</td>
<td>F</td>
<td></td>
<td>Allow switching of FV elements to DG during Runge Kutta stages. This may violated the DG timestep restriction of the element.</td>
</tr>
<tr>
<td>FV_IniSupersample</td>
<td>T</td>
<td></td>
<td>Supersample initial solution inside each sub-cell and take mean value as average sub-cell value.</td>
</tr>
<tr>
<td>FV_IniSharp</td>
<td>F</td>
<td></td>
<td>Maintain a sharp interface in the initial solution in the FV region</td>
</tr>
<tr>
<td>FV_LimiterType</td>
<td>1</td>
<td></td>
<td>Type of slope limiter of second order reconstruction</td>
</tr>
<tr>
<td>swebyb</td>
<td></td>
<td></td>
<td>beta parameter for Sweby limiter</td>
</tr>
</tbody>
</table>
## 3.2. PARAMETER FILE OPTIONS

### Lifting

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doWeakLifting</td>
<td>F</td>
<td>Set true to perform lifting in weak form.</td>
</tr>
<tr>
<td>doConservativeLifting</td>
<td>F</td>
<td>Set true to compute the volume contribution to the gradients in conservative form, i.e. deriving the solution multiplied by the metric terms instead of deriving the solution and multiplying by the metrics. Only available for doWeakLifting = .FALSE.</td>
</tr>
</tbody>
</table>

### EddyViscParameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eddyViscType</td>
<td>none</td>
<td>(0) none: No eddy viscosity, (1) Smagorinsky CS EddyViscParameters constant</td>
</tr>
<tr>
<td>PrSGS</td>
<td>0.7</td>
<td>Turbulent Prandtl number</td>
</tr>
<tr>
<td>VanDriest</td>
<td>F</td>
<td>Van Driest damping, only for channel flow!</td>
</tr>
</tbody>
</table>

### Sponge

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpongeLayer</td>
<td>F</td>
<td>Turn on to use sponge regions for reducing reflections at boundaries.</td>
</tr>
<tr>
<td>damping</td>
<td>1.0</td>
<td>Damping factor of sponge. U_t = U_t - damping * (U - U_base) in fully damped regions.</td>
</tr>
<tr>
<td>SpongeShape</td>
<td></td>
<td>Set shape of sponge: (1) ramp : cartesian / vector-aligned, (2) cylindrical</td>
</tr>
<tr>
<td>SpongeDistance</td>
<td></td>
<td>Length of sponge ramp. The sponge will have maximum strength at the end of the ramp and after that point.</td>
</tr>
<tr>
<td>xStart</td>
<td></td>
<td>Coordinates of start position of sponge ramp (SpongeShape=ramp) or center (SpongeShape=cylindrical).</td>
</tr>
<tr>
<td>SpongeDir</td>
<td></td>
<td>Direction vector of the sponge ramp (SpongeShape=ramp)</td>
</tr>
<tr>
<td>SpongeRadius</td>
<td></td>
<td>Radius of the sponge zone (SpongeShape=cylindrical)</td>
</tr>
<tr>
<td>SpongeAxis</td>
<td></td>
<td>Axis vector of cylindrical sponge (SpongeShape=cylindrical)</td>
</tr>
<tr>
<td>SpongeViz</td>
<td>F</td>
<td>Turn on to write a visualization file of sponge region and strength.</td>
</tr>
</tbody>
</table>
3.2. PARAMETER FILE OPTIONS

**Sponge**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpongeBaseFlow</td>
<td>1</td>
<td>Type of baseflow to be used for sponge. (1) constant: fixed state, (2) exact function, (3) file: read baseflow file, (4) pruett: temporally varying, solution adaptive Pruett baseflow</td>
</tr>
<tr>
<td>SpongeRefState</td>
<td></td>
<td>Index of refstate in ini-file (SpongeBaseFlow=constant)</td>
</tr>
<tr>
<td>SpongeExactFunc</td>
<td></td>
<td>Index of exactfunction (SpongeBaseFlow=exactfunction)</td>
</tr>
<tr>
<td>SpongeBaseFlowFile</td>
<td></td>
<td>FLEXI solution (e.g. TimeAvg) file from which baseflow is read.</td>
</tr>
<tr>
<td>tempFilterWidth</td>
<td></td>
<td>Temporal filter width used to advance Pruett baseflow in time.</td>
</tr>
</tbody>
</table>

**TimeDisc**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeDiscMethod</td>
<td>CarpenterRK4-5</td>
<td>Specifies the type of time-discretization to be used, e.g. the name of a specific Runge-Kutta scheme. Possible values: * standardrk3-3 * carpenterrk4-5 * niegemannrk4-14 * toulorgerk4-8c * toulorgerk3-7c * toulorgerk4-8f * ketchesonrk4-20 * ketchesonrk4-18</td>
</tr>
<tr>
<td>TEnd</td>
<td></td>
<td>End time of the simulation (mandatory).</td>
</tr>
<tr>
<td>CFLScale</td>
<td></td>
<td>Scaling factor for the theoretical CFL number, typical range 0.1..1.0 (mandatory)</td>
</tr>
<tr>
<td>DFLScale</td>
<td></td>
<td>Scaling factor for the theoretical DFL number, typical range 0.1..1.0 (mandatory)</td>
</tr>
<tr>
<td>maxIter</td>
<td>-1</td>
<td>Stop simulation when specified number of timesteps has been performed.</td>
</tr>
<tr>
<td>NCalcTimeStepMax</td>
<td>1</td>
<td>Compute dt at least after every Nth timestep.</td>
</tr>
</tbody>
</table>

**Analyze**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalcErrorNorms</td>
<td>T</td>
<td>Set true to compute L2 and LInf error norms at analyze step.</td>
</tr>
<tr>
<td>AnalyzeToFile</td>
<td>F</td>
<td>Set true to output result of error norms to a file (CalcErrorNorms=T)</td>
</tr>
<tr>
<td>Analyze_dt</td>
<td>0.0</td>
<td>Specifies time intervall at which analysis routines are called.</td>
</tr>
</tbody>
</table>
3.2. PARAMETER FILE OPTIONS

### Analyze

- **nWriteData** 1
  - Intervall as multiple of Analyze_dt at which HDF5 files (e.g. State, TimeAvg, Fluc) are written.

- **NAnalyze**
  - Polynomial degree at which analysis is performed (e.g. for L2 errors). Default: 2*N.

- **AnalyzeExactFunc**
  - Define exact function used for analyze (e.g. for computing L2 errors). Default: Same as IniExactFunc.

- **AnalyzeRefState**
  - Define state used for analyze (e.g. for computing L2 errors). Default: Same as IniRefState.

- **doMeasureFlops** T
  - Set true to measure flop count, if compiled with PAPI.

### AnalyzeEquation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalcBodyForces</td>
<td>F</td>
<td>Set true to compute body forces at walls</td>
</tr>
<tr>
<td>CalcBulkState</td>
<td>F</td>
<td>Set true to compute the flows bulk quantities</td>
</tr>
<tr>
<td>CalcMeanFlux</td>
<td>F</td>
<td>Set true to compute mean flux through boundaries</td>
</tr>
<tr>
<td>CalcWallVelocity</td>
<td>F</td>
<td>Set true to compute velocities at wall boundaries</td>
</tr>
<tr>
<td>CalcTotalStates</td>
<td>F</td>
<td>Set true to compute total states (e.g. Tt,pt)</td>
</tr>
<tr>
<td>CalcTimeAverage</td>
<td>F</td>
<td>Set true to compute time averages</td>
</tr>
<tr>
<td>WriteBodyForces</td>
<td>T</td>
<td>Set true to write bodyforces to file</td>
</tr>
<tr>
<td>WriteBulkState</td>
<td>T</td>
<td>Set true to write bulk state to file</td>
</tr>
<tr>
<td>WriteMeanFlux</td>
<td>T</td>
<td>Set true to write mean flux to file</td>
</tr>
<tr>
<td>WriteWallVelocity</td>
<td>T</td>
<td>Set true to write wall velocities file</td>
</tr>
<tr>
<td>WriteTotalStates</td>
<td>T</td>
<td>Set true to write total states to file</td>
</tr>
<tr>
<td>VarNameAvg</td>
<td></td>
<td>Names of variables to be time-averaged</td>
</tr>
<tr>
<td>VarNameFluc</td>
<td></td>
<td>Names of variables for which Flucs (time-averaged square of the variable) should be computed. Required for computing actual fluctuations.</td>
</tr>
</tbody>
</table>

### RecordPoints

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP_inUse</td>
<td>F</td>
<td>Set true to compute solution history at points defined in recordpoints file.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RP_DefFile</td>
<td></td>
<td>File containing element-local parametric recordpoint coordinates and structure.</td>
</tr>
<tr>
<td>RP_MaxMemory</td>
<td>100</td>
<td>Maximum memory in MiB to be used for storing recordpoint state history. If memory is exceeded before regular IO level states are written to file.</td>
</tr>
<tr>
<td>RP_SamplingOffset</td>
<td>1</td>
<td>Multiple of timestep at which recordpoints are evaluated.</td>
</tr>
</tbody>
</table>
4 Workflow

In this chapter, the complete process of setting up a simulation in FLEXI is detailed.

4.1 Mesh generation using HOPR

FLEXI obtains its computational meshes solely from the high order preprocessor HOPR (available under GPLv3 at https://www.hopr-project.org) in HDF5 format. The design philosophy is that all tasks related to mesh organization, different input formats and the construction of high order geometrical mappings are separated from the parallel simulation code. These tasks are implemented most efficiently in a serial environment.

The employed mesh format is designed to make the parallel read-in process as simple and fast as possible. For details concerning the mesh format please refer to the HOPR HDF5 Curved Mesh Format documentation.

Using HOPR, simple, structured meshes can be directly created using an inbuilt mesh generator. More complex geometries can be treated by importing meshes generated by external mesh generators in CGNS or GMSH format. A number of strategies to create curved boundaries are also included in HOPR.

The test cases provided in Chapter 6 come with both a ready to use mesh file as well as a parameter file for HOPR, which can be used to generate or modify the meshes as needed.

Provided the mesh file has been set up, its location must be specified in the parameter file.

MeshFile = [path to mesh file.h5]

4.2 Solver settings

Before setting up a simulation, the code must be compiled with the desired parameters. The most important compiler options to be set are

- FLEXI_EQNSYSNAME, e.g. Navier-Stokes
- FLEXI_NODETYPE, the nodal collocation points used during the simulation. Available options are either GAUSS or GAUSS-LOBATTO.

All other options are set in the parameter file. The most important steps are
Set the polynomial degree $N$

Defines the polynomial degree of the solution. The order of convergence follows as $N + 1$. Each grid cell contains $(N + 1)^3$ collocation points to represent the solution.

Choose a de-aliasing approach.

For under-resolved Navier-Stokes simulations, e.g. in an LES setting, de-aliasing is important for numerical stability. Various choices are available and set using OverintegrationType.

- OverintegrationType=1
  
The first option is a filtering strategy. The complete operator is first evaluated at $N (U^N_t)$ and then filtered to a lower effective degree $NUnder (U^{NUnder}_t)$. To use this variant, specify $NUnder$ to a value smaller than $N$.

- OverintegrationType=2
  
  In this variant of the first option, the operator in reference space, e.g. $JU_t$, is first projected to the $NUnder$ node set before converting it to physical space $U^{NUnder}_t = JU^{NUnder}_t / J^{NUnder}$. This implementation enforces conservation. To use this variant, specify $NUnder$ to a value smaller than $N$.

An alternative approach to guaranteeing non-linear stability is to use split form DG methods. This needs to be specified during compile time using the FLEXI_SPLIT_DG option, and the specific split flux formulation must be set using the parameter SplitDG.

Choose a Riemann solver

The Riemann solver defines how inter-element coupling is accomplished. The available variants are listed in Section 3.2. Use the Riemann and the RiemannBC options to specify which Riemann solver is to be used at internal interfaces and at Dirichlet boundary conditions, respectively. The default Riemann solver is “Roe with entropy fix”.

Choose a time discretization method

The time discretization method is set using the option TimeDiscMethod. Various explicit Runge-Kutta variants are available and listed in Section 3.2. By default, the low-storage fourth order Runge-Kutta scheme by (Carpenter and Kennedy 1994) is employed.

4.3 Initial and boundary conditions

Initial and boundary conditions are controlled via the so-called RefState and ExactFunction constructs.

The RefState basically specifies a state vector in primitive form $(\rho, u, v, w, p)^T$. An arbitrary number of RefStates can be defined:
In this example, the first state would result in a parallel flow in $x$ direction at $Ma = 1$, the second state at $Ma = 0.3$.

### 4.3.1 Initial conditions

The code contains a number of pre-defined analytic solution fields (ExactFunctions), which are invoked by specifying their respective number. For instance the initialization of a simple constant freestream is achieved by setting

```plaintext
IniExactFunc=1
```

The associated state vector to be used is determined by

```plaintext
IniRefState=1
```

This implies that the first of the two available RefStates is used for initialization.

Note: currently, the ExactFunctions contained in the code are not documented yet. They can be looked up in the source file `src/equations/navierstokes/equation.f90`.

### 4.3.2 Boundary conditions

The names of the boundaries are contained in the mesh file and can be used in the **FLEXI** parameter file to override the boundary conditions already set in the parameter file, if necessary.

**FLEXI** lists the boundaries and their respective boundary conditions during initialization:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>State</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_periodicz-</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>BC_periodicy-</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>BC_periodicx+</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>BC_periodicy+</td>
<td>1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>BC_periodicx-</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BC_periodicz+</td>
<td>1</td>
<td>0</td>
<td>-3</td>
</tr>
</tbody>
</table>

Suppose, we wish to apply a Dirichlet boundary condition with RefState 2 at the two lateral boundaries. Therefore, we have to add the following lines to the parameter file

```plaintext
BoundaryName=BC_periodicy-  
BoundaryType=(/2,2/)  
BoundaryName=BC_periodicy+  
BoundaryType=(/2,2/)
```
Note that the first index within brackets specifies BC_TYPE, while the second one specifies BC_STATE, in this case the number of the RefState to be used. In general, BC_STATE identifies either a RefState, an ExactFunction or remains empty, dependent on the BC_TYPE. Currently implemented boundary types for Navier-Stokes are listed in table 4.1.

Table 4.1: Boundary conditions.

<table>
<thead>
<tr>
<th>Boundary Condition</th>
<th>BC_TYPE</th>
<th>BC_STATE</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic BC</td>
<td>1</td>
<td>-</td>
<td>Can only be defined in HOPR</td>
</tr>
<tr>
<td>Weak Dirichlet</td>
<td>2</td>
<td>RefState</td>
<td>Like 2, but using an external state set by BCStateFile</td>
</tr>
<tr>
<td>Weak Dirichlet</td>
<td>12</td>
<td>-</td>
<td>Like 2, but using an ExactFunction</td>
</tr>
<tr>
<td>Weak Dirichlet</td>
<td>22</td>
<td>ExactFunction</td>
<td>Like 2, but using an ExactFunction</td>
</tr>
<tr>
<td>Wall adiabatic</td>
<td>3</td>
<td>-</td>
<td>Isothermal wall, temperature is specified via $p$ and $\rho$ contained in the RefState</td>
</tr>
<tr>
<td>Wall isothermal</td>
<td>4</td>
<td>RefState</td>
<td>Slip, symmetry or Euler wall</td>
</tr>
<tr>
<td>Wall slip</td>
<td>9</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Outflow Mach no.</td>
<td>23</td>
<td>RefState</td>
<td></td>
</tr>
<tr>
<td>Outflow Pressure</td>
<td>24</td>
<td>RefState</td>
<td></td>
</tr>
<tr>
<td>Outflow Subsonic</td>
<td>25</td>
<td>RefState</td>
<td></td>
</tr>
<tr>
<td>Inflow Total pressure / Temp.</td>
<td>27</td>
<td>RefState</td>
<td>Special Refstate: total quantities $(T_i, \alpha, \beta, 0, p_t)$</td>
</tr>
</tbody>
</table>

4.4 Material properties

At present, the only available equation of state in the Navier-Stokes solver of FLEXI is the ideal gas. The gas constant, adiabatic exponent, Prandtl number and viscosity are specified in the parameter file using $R$, $\kappa$, $Pr$ and $\mu_0$.

1 see (Carlson 2011) for details on the listed inflow/outflow boundary conditions.
4.5 Output time interval

Set the end time of the simulation using $\text{TEnd}$ and the interval in which the solution is dumped to the hard drive with $\text{Analyze\_dt}$.

Note that evaluation of body forces and other runtime analysis routines are also invoked once in every analyze interval determined by $\text{Analyze\_dt}$. Set e.g. $\text{nWriteData}=10$ to a value greater one to restrict the solution output to every 10th $\text{Analyze\_dt}$.

Restart the simulation

The simulation may be restarted from an existing state file

```
flexi parameter.ini [restart_file.h5]
```

Note: when restarting from an earlier time (or zero), all later state files possibly contained in your directory are deleted!

4.6 Evaluation during runtime

At every $\text{Analyze\_dt}$, the following evaluations are possible:

- $\text{CalcErrorNorms}=T$: Calculate the $L_2$ and $L_{\infty}$ error norms based on the specified $\text{ExactFunc}$ as reference. This evaluation is used for e.g. convergence tests.
- $\text{CalcBodyForces}=T$: Calculate the pressure and viscous forces acting on every wall boundary condition (BC\_TYPE=3,4 or 9) separately. The forces are written to .dat files.
- $\text{CalcBulkState}=T$: Calculate the bulk quantities (e.g. bulk velocity in channel flow).
- $\text{CalcWallVelocity}=T$: Due to the discontinuous solution space and the weakly enforced boundaries, the no-slip condition is not exactly fulfilled. The deviation depends mainly on the resolution in the near-wall region. Thus, this evaluation can be used as a resolution measure at the wall.

4.7 Parallel execution

The simulation code is specifically designed for (massively) parallel execution using the MPI library. For parallel runs, the code must be compiled with $\text{FLEXI\_MPI=ON}$.

Parallel execution is then controlled using mpirun

```
mpirun -np [no. processors] flexi parameter.ini
```
4.7.1 Domain decomposition

The grid elements are organized along a space-filling curved, which gives a unique one-dimensional element list. In a parallel run, the mesh is simply divided into parts along the space-filling curve. Thus, domain decomposition is done fully automatic and is not limited by e.g. an integer factor between the number of cores and elements. The only limitation is that the number of cores may not exceed the number of elements.

4.7.2 Choosing the number of cores

Parallel performance heavily depends on the number of processing cores. The performance index is defined as

\[ PID = \frac{\text{WallTime}}{n\text{Cores} \cdot n\text{DOF} \cdot n\text{TimeSteps}} \]

and measures the CPU time per degree of freedom and time step. During runtime, the average \( PID \) is displayed in the output

![Calculation time per stage/DOF: 5.59330E-07 sec]

When compared to the single performance, it can be used as a parallel efficiency measure. The \( PID \) is mainly dependent on the load per core

\[ \text{Load} = \frac{n\text{DOF}}{n\text{Cores}} \]

and the polynomial degree \( N \). Load values for optimal performance lie in the range \( \text{Load} = 2000 \text{ -- } 5000 \). A detailed parallel performance analysis at the example of a Cray XC-40 system is given in [Atak et al. 2016].

4.8 Test case environment

The test case environment can be used as to add test case-specific code for e.g. custom source terms or diagnostics to be invoked during runtime.

The compiler option \texttt{FLEXI\_TESTCASE} sets the current test case. The test cases are contained in the \texttt{src/testcase/} folder.

Standardized interfaces are defined for initialization, source terms and analysis routines

- \texttt{InitTestcase}
  
  Read in testcase related parameters from the \texttt{parameter.ini}, initialize the corresponding data structures.
4.9 POST PROCESSING / POSTI_VISU TOOL

- **FinalizeTestcase**
  Deallocate test case specific data structures.

- **ExactFuncTestcase**
  Define test case specific analytic expressions for initial or boundary conditions.

- **CalcForcing**
  Impose test case specific source terms, e.g. the pressure gradient in test case channel.

- **AnalyzeTestCase**
  Perform test case specific diagnostics.

Currently supplied test cases are

- **default**
- **channel**: turbulent channel flow with steady pressure gradient source term
- **phill**: periodic hill flow with controlled pressure gradient source term
- **riemann2d**: a two dimensional Riemann problem
- **taylorgreenvortex**: automatic diagnostics for the Taylor-Green vortex flow

Note that the test case environment is currently only applicable to the *Navier-Stokes* equation system.

## 4.9 Post processing / posti_visu tool

*FLEXI* comes with a posti_visu tool for visualization with Paraview. The posti_visu tool takes the HDF5 files generated by FLEXI and converts them to vtu-files readable by Paraview.

```
posti_visu [posti-prm-file [flexi-prm-file]] statefile [statefiles]
```

The posti_visu tool runs in parallel with activated *FLEXI_MPI* flag

```
```

Multiple HDF5 files can be passed to the posti_visu tool at once. The runtime parameters to be set in parameter_postiVisu.ini are

<table>
<thead>
<tr>
<th>NodeTypeVisu</th>
<th>VISU</th>
<th>Node type of the visualization basis: VISU,GAUSS,GAUSS-LOBATTO,CHEBYSHEV-GAUSS-LOBATTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVisu</td>
<td></td>
<td>Polynomial degree at which solution is sampled for visualization.</td>
</tr>
</tbody>
</table>

Table 4.2: Runtime parameters for the posti_visu tool.
4.9. POST PROCESSING / POSTI_VISU TOOL

VarName: Names of variables, which should be visualized.

Some of these options are duplicates from options for FLEXI. You can use the same parameter file for both executables.

The default value for NodeTypeVisu uses equidistant nodes which include the boundary points of elements. The default value for NVisu is to use the same number of sampling points as collocation points on each element. For high quality visualization, it is usually adviseable to choose NVisu higher than the polynomial degree of the computation in order to keep interpolation errors small. The parameter VarName specifies the variables being visualized. It can be used multiple times for each variable. Visualization of derived quantities such as Velocities and Pressure is possible. If no VarName is specified, the five conservative variables are visualized.

The following lines can be used as an example for the parameter_postiVisu.ini file.

```
NVisu = 10
varName = MomentumX
varName = VelocityX
varName = Density
varName = Pressure
varName = Temperature
```
5 Tools Overview

This section gives an overview over the tools contained in the FLEXI repository. It also provides references to the tutorials where their usage is explained. There are two different kinds of tools:

- POSTI-tools can be compiled together with FLEXI given the according cmake options.
- In the tools folder, a collection of mostly shell or python scripts can be found. They are mostly used to manage FLEXI runs and FLEXI output files.

5.1 POSTI

POSTI tools are mostly documented in section 4.9 and in the tutorials (chapter 6). Here, an overview is given together with references to the respective tutorials. A list and description of input parameters can be shown with the command

\[ \text{[posti_toolname]} \ --\text{help} \]

for all POSTI tools using a parameter file.

5.1.1 Visualization

<table>
<thead>
<tr>
<th>posti_visu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
</tr>
<tr>
<td>Basic usage</td>
</tr>
<tr>
<td>Further info / usage example</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Paraview plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
</tr>
<tr>
<td>Basic usage</td>
</tr>
</tbody>
</table>
5.1. POSTI Chapter 5. Tools Overview

**Paraview plugin**

Further info / usage example  No tutorials so far

### 5.1.2 Swap meshes

**posti_swapmesh**

Brief description  Interpolates state file data from one mesh to another. Uses high-order interpolation and a Newton coordinate search algorithm. Meshes do not have to be conforming. A reference state can be given for areas in the target mesh not covered by the source mesh.

Basic usage  

Further info / usage example  No tutorials so far

### 5.1.3 Record points

**posti_preparerecordpoints**

Brief description  Enables FLEXI to record values at a set of physical points over time with a higher temporal sampling rate than the state file output interval. The record point coordinates and the FLEXI mesh are defined in the parameter file. Creates an additional .h5 file, whose path is passed to FLEXI as a parameter.

Basic usage  

Further info / usage example  6.3.13

**posti_visualizerecordpoints**

Brief description  Performs post-processing of the *_RP_* files written by FLEXI: merges several time steps and writes output such as value over time or spectra.

Basic usage  

Further info / usage example  6.3.13
### 5.1. POSTI Chapter 5. Tools Overview

#### posti_evaluaterecordpoints

<table>
<thead>
<tr>
<th>Brief description</th>
<th>Evaluate the values at recordpoints a posteriori from existing statefiles. Can be used if the recordpoints have not been set during the simulation, but will only give coarse temporal resolution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic usage</td>
<td><code>posti_evaluaterecordpoints [parameter.ini] [statefile.h5]</code></td>
</tr>
<tr>
<td>Further info / usage example</td>
<td>No tutorials so far</td>
</tr>
</tbody>
</table>

#### 5.1.4 Time averaging

#### posti_mergetimeaverages

<table>
<thead>
<tr>
<th>Brief description</th>
<th>Averages several FLEXI State or TimeAverage files. If TimeAverage files are the input, each files is weighted with its time averaging period. State files are all weighted equally. All HDF5 data sets are averaged. No parameter file is needed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic usage</td>
<td><code>posti_mergetimeaverages [statefile1.h5 statefile2.h5 ...]</code></td>
</tr>
<tr>
<td>Further info / usage example</td>
<td>No tutorials so far</td>
</tr>
</tbody>
</table>

#### posti_calcfluctuations

<table>
<thead>
<tr>
<th>Brief description</th>
<th>Calculates Fluctuations from Mean and MeanSquare as given in the (merged) TimeAverage files. Fluctuations are written into an additional data set in the same HDF5 file. All applicable fluctuations are calculated. No parameter file is needed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic usage</td>
<td><code>posti_calcfluctuations [statefile1.h5 statefile2.h5 ...]</code></td>
</tr>
<tr>
<td>Further info / usage example</td>
<td>No tutorials so far</td>
</tr>
</tbody>
</table>

#### 5.1.5 Test cases
### posti_channel_fft

<table>
<thead>
<tr>
<th>Brief description</th>
<th>Calculates the mean velocity and Reynolds stress profiles of the turbulent channel flow test case by averaging both in the direction parallel to the wall and by averaging the upper and lower half of the channel. Furthermore, kinetic energy spectra dependent on the distance to the wall are computed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic usage</td>
<td><code>posti_channel_fft [parameter_channelfft.ini] [statefile1.h5 statefile2.h5 ...]</code></td>
</tr>
<tr>
<td>Further info / usage example</td>
<td>6.9</td>
</tr>
</tbody>
</table>
5.2 tools folder

The scripts provided in the tools folder are generally not part of the tutorials. They are briefly described below. The path to the python files (of the form $FLEXIROOT/tools/SUBDIR/) is omitted in the following.

For most python tools, possible arguments and syntax can be shown with the -h argument:

```python
python2 [toolname.py] -h
```

5.2.1 animate

The python script animate_paraview.py creates movies from a series of state files using PvBatch, a GUI-less interface to ParaView. You need ParaView installed on your system (details can be found in the ParaView section) and the directory containing the PvBatch executable needs to be a part of your $PATH. Before running this script, you have to visualize your FLEXI state file with ParaView and save the current view via Save State..., e.g. under the name pvstate.pvsm. You also need the MEncoder tool installed. The basic command to run the script is

```python
python2 animate_paraview.py -l [pvstate.pvsm] -r [path_to_posti_paraview_plugin] [statefile1.h5 statefile2.h5 ...]
```

Apart from the movie file, the script also outputs a .png-file for each HDF5 file given as input. In order to visualize a set of .vtu-files, e.g. from posti_visu output, omit the -r argument and pass .vtu-files instead of .h5-files. Further options can be shown with the -h argument.

There are further tools for image handling in this folder:

The tool concatenatepics.py stitches several pairs of images (e.g. creates a time series of stitched images from two time series of images). A possible command could look like this (Further options can be shown with the -h argument):

```python
python concatenatepics.py -d e -p left*.png -a right*.png
```

The tool crop.py crops several images to the same size. Simply pass all images as arguments:

```python
python crop.py [image*.png]
```

The script pics2movie.py creates a movie from several images using the mencoder tool (which is also done as part of the animate_paraview.py script. Basic usage is again

```python
python pics2movie.py [image*.png]
```

and further options can again be shown with the -h argument.
5.2.2 convergence_test

The python scripts convergence and convergence_grid provide automated convergence tests for p- and h-convergence, respectively. The basic command is

```
python2 convergence flexi [parameter.ini]
```

where convergence can be replaced by convergence_grid for h-convergence. Further options can again be shown with the -h option.

Note that for h-convergence, the mesh names are hard-coded to the form CART_HEX_PERIODIC_MORTAR_XXX_2D_mesh.h5, where XXX is the number of elements in each direction, and MORTAR and 2D are optional. The polynomial degree in the parameter file is always overwritten by that passed to the script as an optional argument, with a default value of 3, if no such argument is passed.

5.2.3 userblock

The userblock contains complete information about a FLEXI run (git branch of the repository, differences to that branch, cmake configuration and parameter file) and is prepended to every .h5 state file. The parameter file is prepended in ASCII format, the rest is binary and is generated automatically during the build process with the generateuserblock.sh script. It can be extracted and printed using the extract_userblock.py script. Its basic usage is

```
python2 extract_userblock.py -XXX [statefile.h5]
```

where -XXX can be replaced by

- -s to show all available parts of the userblock (such as CMAKE or GIT BRANCH)
- -a to print the complete userblock
- -p [part] to print one of the parts listed with the -s command.

The second python tool in this folder is rebuild.py. It extracts the userblock from a state file and builds a FLEXI repository and binary identical to the one that the state file was created with. In order to do so, it clones a FLEXI git repository, checks out the given branch, applies the stored changes to the git HEAD and builds FLEXI with the stored cmake options. If run with the parameter file given in the INIFILE part of the userblock, this binary should reproduce the same results/behaviour (possible remaining sources of different output are for example differences in restart files, compilers, linked libraries or machines). The basic usage is

```
python2 rebuild.py [dir] [statefile.h5]
```

where dir is an empty directory that the repository is cloned into and where the flexi executable is built. statefile.h5 is the state file whose userblock is used to rebuild the flexi executable. Help can be shown via -h for both userblock scripts.
5.2.4 sortfiles.sh

The sortfiles.sh script sorts all .h5-files in subfolders State, BaseFlow, TimeAvg and RP, while keeping the last time instance at the upper level. It also copies Log.*.sdb, .log and .out files into a logs subdirectory. The project name is hard-coded in the script and has to be adapted there, the directory that is to be sorted is passed as an argument.

5.2.5 getload.py

This script is specific to runs on HPC systems. It calculates a suitable number of nodes and cores to achieve

- a number of degrees of freedom per core which is close to a target
- an average number of elements per core which is just below a close integer, such that parallel efficiency is not impaired by a few cores with higher load that the others have to wait for.

No arguments are passed to this script, all input values are hard-coded and have to be adjusted in the script.

5.2.6 testcases

The python script plotChannelFFT.py creates plots of the mean velocity and the Reynolds stress profiles as well as the turbulent energy spectra based on the posti_channel_fft HDF5 output files. Basic usage is:

```
python plotChannelFFT.py -p projectname -t time
```

Further options can be shown with the -h argument.
6 Tutorials

This chapter will give a detailed overview of flow simulations with FLEXI. It assumes that you are familiar with how to set the compiler options and how to compile the code. The path to all executables is omitted. It is assumed that you either added aliases for flexi, hopr and all posti tools, or that you added the binary directories to your $PATH variable as described in 2.1.3.

Each tutorial is equipped with .ini files parameter_hopr.ini, parameter_flexi.ini, parameter_postiVisu.ini as well as mesh file *_mesh.h5 in HDF5 format (created with HOPR).

| parameter_hopr.ini | parameter_flexi.ini | parameter_postiVisu.ini | mesh.h5 |

We suggest to copy each folder to a new directory, where you can run and modify the “INI-files”.

6.1 Freestream

The setup considers a freestream scenario with constant pressure \( p = 101325.0 \) Pa, density \( \rho = 1.225 \) kg/m\(^3\) and velocity vector \( \mathbf{U} = (1, 1, 1)^T \) m/s.

Copy the freestream tutorial folder

```bash
cp -r $FLEXI_TUTORIALS/freestream ./
cd freestream
```

6.1.1 Mesh Generation with HOPR

The mesh files used by FLEXI are created by supplying an input file parameter_hopr.ini with the appropriate information.

```bash
hopr parameter_hopr.ini
```

This creates the mesh file cartbox_mesh.h5 in HDF5 format. Alternatively, if you do not want to run hopr, you can also use the provided mesh.
6.1.2 Flow Simulation with FLEXI

The simulation setup is defined in parameter_flexi.ini. The initial condition is selected via the variable vector $\text{RefState}=(1.225,1.,1.,1.,101325.)$ which represents the solution vector $(\rho,u,v,w,p)^T$. FLEXI allows for multiple $\text{RefState}$ vectors and numerates them respectively for them to be used by different functions. In this example a single $\text{RefState}$ is supplied and therefore is given the number 1.

IniRefState = 1: the initial condition uses $\text{RefState} 1$ for the initial flow field solution.

IniExactFunc = 1: the used exact function routine uses $\text{RefState} 1$, e.g., for the calculation of the $L_2$ error norm.

Constant flow properties like the gas constant are given in table 6.1 and define the gas behavior in combination with the ideal gas law $p = \rho RT$.

<table>
<thead>
<tr>
<th>Property</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic viscosity $\mu$</td>
<td>mu0</td>
<td>0.000018547</td>
</tr>
<tr>
<td>ideal gas constant $R$</td>
<td>R</td>
<td>276</td>
</tr>
<tr>
<td>isentropic coefficient $\kappa$</td>
<td>kappa</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Table 6.1: Numerical settings
6.1.3 Numerical settings

The DG solution on the mesh is represented by piecewise polynomials and the polynomial degree in this tutorial is chosen as $N = 3$.

The default settings for these properties are displayed in table 6.2.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>polynomial degree</td>
<td>3</td>
</tr>
<tr>
<td>MeshFile</td>
<td>cartbox_mesh.h5</td>
<td></td>
</tr>
<tr>
<td>tend</td>
<td></td>
<td>1e-6</td>
</tr>
<tr>
<td>Analyze_dt</td>
<td></td>
<td>1e-6</td>
</tr>
<tr>
<td>nWriteData</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>CFLscale</td>
<td></td>
<td>0.99</td>
</tr>
<tr>
<td>DFLscale</td>
<td></td>
<td>0.4</td>
</tr>
</tbody>
</table>

The command

```
flexi parameter_flexi.ini > std.out
```

runs the code and dumps all output into the file `std.out`. If the run has completed successfully, which should take only a brief moment, the contents of the working folder should look like in figure 6.1

![Figure 6.1: The folder contents after a successful run](image)

Two additional files have been created, which are named `Projectname_State_Timestamp.h5`. They contain the solution vector of the conserved variables at each interpolation node at the given time, which corresponds to multiplies of `Analyze_dt`. If these files are not present, something went wrong during the execution of FLEXI. In that case, check the `std.out` file for an error message.

After a successful completion, the last lines in this files should look like in figure 6.2

![Figure 6.2](image)

To visualize the solution, the `State`-files must be converted into a format suitable for ParaView. Issue the command
6.2 Lid-driven Cavity

This tutorial describes how to set up and run the first non-trivial flow problem. The lid-driven cavity flow is a standard test case for numerical schemes, and a number of results have been published in literature, see e.g. (Ghia 1982), (Gao 2016). This tutorial assumes that you have completed the previous tutorial, know how to edit files and postprocess the solution with your favorite visualization tool, e.g. ParaView. Also, the later parts of the tutorial assume that you have access to a computer with an MPI-based parallelization with at least 4 computing cores - otherwise, it will just take a lot longer :).

The tutorial is split into two parts: The basic part will teach you about setting up the code and running the simulations. The advanced part will build on this and give you a glimpse on how to make modification to code to accommodate more complex simulation and add features you might need. If you are just interested in running the code as is, you may skip the advanced part, or just complete parts of it.
6.2.1 Flow description

The flow under consideration is essentially incompressible and two-dimensional, but we will use the three-dimensional code for the compressible Navier-Stokes equations to solve it here. This is not the most efficient way to compute this flow, but it works well to show you how to set up and run a simulation in this tutorial. The computation will be conducted in a three-dimensional, square domain with periodic boundary conditions in the “third” direction. The wall of the cavity are modelled as isothermal walls, and a fixed flow is prescribed at the upper boundary, i.e. the lid of the domain. For the Reynolds numbers investigated here, this generates a steady, vortical flow field in the cavity.

The following picture (figure 6.3) shows the resulting velocity field and streamlines for $Re = 400$.

![Figure 6.3: Contours of velocity magnitude for the $Re = 400$ lid-driven cavity case.](image)

6.2.2 Compiler options

Make sure that FLEXI is compiled with the cmake options listed in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAKE_BUILD_TYPE</td>
<td>Release</td>
<td></td>
</tr>
<tr>
<td>FLEXI_EQYNSYSNAME</td>
<td>navierstokes</td>
<td></td>
</tr>
<tr>
<td>FLEXI_PARABOLIC</td>
<td>ON</td>
<td></td>
</tr>
</tbody>
</table>
The standard settings are sufficient for this example. To check whether they are set, change
to your build folder and open the cmake GUI

```
ccmake [flexi root directory]
```

If necessary, set the above options and then compile the code by issuing

```
make
```

### 6.2.3 Part I: Basic Tutorial - Flow at $Re = 100$

Copy the cavity tutorial folder

```
cp -r $FLEXI_TUTORIALS/cavity .
```

Step into the Basic_Re100 subfolder. The contents of the folder should look like in picture
(figure 6.4). In case you do not want to generate the mesh files yourselves, the meshes have
already been provided.

Figure 6.4: Files for the basic lid-driven cavity tutorial.

#### 6.2.3.1 Mesh Generation with HOPR

The domain of interest consists of a square 2D geometry. Although the flow field is two-
dimensional, we will create a three-dimensional domain here and apply periodic boundary
conditions in the $z$-direction. Also, we will only use one element in that direction to save
computational costs.

Like before, the mesh files used by FLEXI are created by supplying an input file `parameter_hopr.ini`
with the appropriate information to HOPR.

```
hopr parameter_hopr.ini
```

A full tutorial on how to run HOPR is available at the project site HOPR-project, so we will
just review the basics here. In `parameter_hopr.ini`, the following lines create a cubical domain
6.2. LID-DRIVEN CAVITY

from \([0,0,0] \rightarrow [1,1,1]\), discretized by \(4 \times 4\) elements in the \(x-y\) plane and 1 element in the \(z\)-direction.

<table>
<thead>
<tr>
<th>Corner</th>
<th>! Corner node positions: (/ x_1,y_1,z_1, x_2,y_2,z_2, ... , x_8, y_8,z_8/)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4×4×1</td>
<td>! number of elements in each direction</td>
</tr>
</tbody>
</table>

The next line maps the 6 faces of the cube to the boundary conditions following the CGNS notation, i.e. it indicates which of the boundary conditions defined below belong to which face.

<table>
<thead>
<tr>
<th>BCIndex</th>
<th>! Indices of Boundary Conditions for six Boundary Faces (z-,y-,x+,y+,x-,z+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,3,6,4,5,2</td>
<td>! BC Index 1</td>
</tr>
<tr>
<td>1,0,0,1</td>
<td>! (/ Type , curveIndex , State , alpha /)</td>
</tr>
<tr>
<td>1,0,0,1</td>
<td>! (/ Type , curveIndex , State , alpha /)</td>
</tr>
<tr>
<td>1,0,0,1</td>
<td>! (/ Type , curveIndex , State , alpha /)</td>
</tr>
<tr>
<td>1,0,0,1</td>
<td>! (/ Type , curveIndex , State , alpha /)</td>
</tr>
<tr>
<td>1,0,0,1</td>
<td>! (/ Type , curveIndex , State , alpha /)</td>
</tr>
</tbody>
</table>

For example, the first boundary condition defined will be applied to the \(z-\) face of the cube, while the third one will be applied to the \(y-\) face and so on. The order \((z-,y-,x+,y+,x-,z+)\) is given by the CGNS standard and thus fixed, so if you keep this line as it is, the boundary conditions below can be defined in the (less confusing) order of \((z-,z+,y-,y+,x-,x+)\).

Figure: BC Indices for the cube, front and back view.

The boundary conditions are now assigned to faces in the following lines:

| BoundaryName=BC_zminus | ! BC index 1 |
| BoundaryType=/1,0,0,1/ | ! (/ Type , curveIndex , State , alpha /) |
| BoundaryName=BC_zplus  | ! BC index 2  |
6.2. LID-DRIVEN CAVITY

<table>
<thead>
<tr>
<th>BoundaryType=(/1,0,0,-1/)</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td>vv=(/0.,0.,1./)</td>
<td>!</td>
</tr>
<tr>
<td>BoundaryName=BC_wall_lower</td>
<td>! BC index 3</td>
</tr>
<tr>
<td>BoundaryType=(/4,0,1,0/)</td>
<td>!</td>
</tr>
<tr>
<td>BoundaryName=BC_free</td>
<td>! BC index 4</td>
</tr>
<tr>
<td>BoundaryType=(/2,0,0,0/)</td>
<td>!</td>
</tr>
<tr>
<td>BoundaryName=BC_wall_left</td>
<td>! BC index 5</td>
</tr>
<tr>
<td>BoundaryType=(/4,0,1,0/)</td>
<td>!</td>
</tr>
<tr>
<td>BoundaryName=BC_wall_right</td>
<td>! BC index 6</td>
</tr>
</tbody>
</table>

The identifier **BoundaryName** can be chosen freely, it is good practice to start it with **BC**_. The identifier **BoundaryType** specifies what kind of boundary is to be applied to the face. For the \(z\)−oriented faces, periodic boundary conditions are chosen, which are denoted by type 1, the first index in the array. The second index indicates if the face is **curved** (not for this test case), and the third indicates which **reference state** will be used at the boundary. This will be discussed in more detail later, it is important to note that the state can be set here when generating the grid, but can also be overwritten in *parameter_flexi.ini* later. The fourth entry is used to match periodic boundary condition sides, i.e. these need to form a pair of the form \([a,−a]\), where \(a\) is a unique integer for each periodic pair and the sign specifies the orientation. In this example, the \(z−\) and \(z+\) sides are connected and the associated connection vector \(vv\) is in the \(z−\)direction. In summary, to specify a periodic side pair, the first index in **BoundaryType** must be 1, the fourth index must be a matching pair of integers and the connection vector must be specified.

In the \(y−\)direction, we choose an isothermal wall boundary condition at the lower side (type 4) and a Dirichlet / fixed state at the top of the cavity (type 2). Again, the corresponding states can be set here, but overwritten later.

In the \(x−\)direction, the left and right walls are again chosen to be isothermal wall boundary conditions.
6.2. LID-DRIVEN CAVITY

Chapter 6. Tutorials

Figure: BC Types for the lid driven cavity, front and back view.

For this basic tutorial, the simple meshes shown in picture (figure 6.5) will be used.

Figure 6.5: Meshes for the basic lid-driven cavity tutorial.

6.2.3.2 Preparing the Flow Simulation with FLEXI

The simulation setup is defined in `parameter_flexi.ini`. To get help on any of the parameters listed therein, you can run FLEXI from the command line by typing

```
flexi --help
```

or

```
flexi --help SECTION
```

or
6.2. LID-DRIVEN CAVITY

The first command lists all parameters grouped thematically, however, this is not mandatory. The second command lists all parameters of a certain section and the last command only gives help about a specific option. All lines starting with a “!” are comments.

6.2.3.2.1 Output parameters

The parameter `ProjectName` is used to name all the files generated by the simulation, e.g. the state at time 0.3 (the solution vector of conserved variables at each node) is written to the file `Tutorial_Cavity_Re100_State_0000000.300000000.h5` and so on. The parameter `outputFormat` is set to 0 here, indicating that the output is off. Turning it on will result in an on-the-fly visualization output of the solution states, but this is not recommended as good practice - it will slow down the code considerably, in particular in parallel mode. State-files in the HDF5-format are written always independent of this parameter. The recommended way is to use the `posti_visu` tool after the simulation to generate Paraview-readable files. Note that currently, the only output format available is `vtk` (by setting `outputFormat = 3`), since Tecplot output is not available due to GPL licensing issues.

6.2.3.2.2 Interpolation / Discretization parameters

The parameter `N` sets the degree of the solution polynomial, e.g. in this example, the solution is approximated by a polynomial of degree 3 in each spatial direction. This results in \((N + 1)^3\) degrees of freedom for each (3D) element. In general, \(N\) can be chosen to be any integer greater or equal to 1, however, the discretization and the timestep calculation has not extensively been tested beyond \(N \approx 23\). Usually, for a good compromise of performance and accuracy is found for \(N \in [3,...,9]\).

`NAnalyze` determines the polynomial degree for the analysis routines, e.g. the accuracy of the calculation of error norms or testcase specific integrals during the computation. A good rule of thumb is to set `NAnalyze = 2 \times N`. 
6.2.3.2.3 Mesh parameters

The parameter `MeshFile` contains the name of the HOPR mesh file in HDF5 format (and/or the full path to it). `UseCurveds` indicates whether the mesh is considered to be curved, i.e. if high-order mesh information should be used. Setting this to `F` can be used to discard high order information in the mesh file and treat it as a linear mesh. For the current tutorial, the meshes are linear by design.

The boundary conditions are set via the `BoundaryName` identifier, which specifies a name that must be present in the mesh file (see section on mesh generation above). Each line containing the boundary name must be followed by a line containing the `BoundaryType`. A list of types available for the Navier-Stokes equations can be found in table 4.1. For types that require additional information (like Dirichlet boundaries), the second index in `BoundaryType` refers to the `RefState` (short for reference state) which is used to determine the unknowns / quantities from the outside for this boundary condition. For example, for a Dirichlet boundary (Type 2), the full reference state is set at the boundary, so the lines

```plaintext
BoundaryName=BC_free
BoundaryType=(/2,1/)
```

indicate that the first reference state vector listed below is set at this boundary (the lid part of the cavity). Note that the reference vectors are always in primitive variables, i.e. \((\rho, u, v, w, p)^T\) unless specified otherwise. Also, for the boundaries `BC_wall_left` and `BC_wall_right`, the same reference state is chosen. These boundaries are isothermal walls, so a wall temperature needs to be specified - this is computed from the primitive variables in the associated reference state.

Note that the lines for the lower wall boundary are commented out. As has been mentioned above, the boundary conditions can be set in HOPR directly, and then be overwritten here. This is just an example to show how both approaches work. Later, when running FLEXI with these settings, it is good practice to inspect the boundary condition information as understood by FLEXI. In this case, the output of FLEXI to the console will look like this:
6.2. LID-DRIVEN CAVITY

| BoundaryName     | BC_wall_left | *CUSTOM |
| BoundaryType     | (/ 4, 1 /)   | *CUSTOM |
| BoundaryName     | BC_wall_right| *CUSTOM |
| BoundaryType     | (/ 4, 1 /)   | *CUSTOM |
| BoundaryName     | BC_free      | *CUSTOM |
| BoundaryType     | (/ 2, 1 /)   | *CUSTOM |

Boundary in HDF file found | BC_free
was | 2 0
is set to | 2 1
Boundary in HDF file found | BC_wall_left
was | 4 1
is set to | 4 1
Boundary in HDF file found | BC_wall_right
was | 4 1
is set to | 4 1

**BOUNDARY CONDITIONS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>State</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_zminus</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BC_zplus</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>BC_wall_lower</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BC_free</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BC_wall_left</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BC_wall_right</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**6.2.3.2.4 Equation parameters**

In this section of the parameter file, the settings associated with the equation to be solved are set, including initial conditions and reference data for the boundary conditions. The parameter *IniExactFunc* specifies which solution or function should be used to fill the initial solution vector, i.e. it specifies what the starting flow field looks like. Setting this to 1 selects a uniform initial state in the whole domain. Note that this solution is also used to compute the error norms and can also be time-dependent. The state itself is defined by the parameter *IniRefstate*, in this case, the second one is used.

```
! EQUATION
 IniExactFunc = 1
 IniRefState  = 2
 RefState     = (/1.0,1.0,0.0,0.0,71.4285714286/)
 RefState     = (/1.0,0.0,0.0,0.0,71.4285714286/)
 mu0          = 0.01
 R            = 1
 Pr           = 0.72
 kappa        = 1.4
```

R2019-06-06 53
As described above, the RefStates are given in primitive variables. The second one describes a fluid at rest and is used to initialize a resting fluid in the cavity. The first state is used to determine the driving flow at the top of the cavity (and to compute the wall temperatures for the boundaries). Constant flow properties like the gas constant are given in table 6.4 and define the gas behavior in combination with the ideal gas law \( p = \rho RT \). Note that the code itself does not distinguish between dimensional and non-dimensional quantities and it is the user’s responsibility to set all data consistently. For anything other than an ideal gas with constant viscosity and heat conductivity, physically meaningful quantities should be set.

<table>
<thead>
<tr>
<th>Property</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic viscosity ( \mu )</td>
<td>mu0</td>
<td>0.1</td>
</tr>
<tr>
<td>ideal gas constant ( R )</td>
<td>R</td>
<td>1</td>
</tr>
<tr>
<td>isentropic coefficient ( \kappa )</td>
<td>kappa</td>
<td>1.4</td>
</tr>
<tr>
<td>Prandtl number ( Pr )</td>
<td>Pr</td>
<td>0.72</td>
</tr>
</tbody>
</table>

From these settings, the Mach- and Reynolds number can be computed as follows, taking into account a reference cavity length of 1 and the magnitude of the driving velocity:

\[
Mach = \frac{u}{c} = 1.0/ \sqrt{\frac{\kappa}{\rho}} = 1.0/10.0 = 0.1
\] (6.1)
\[
Re = \frac{uL\rho}{\mu_0} = \frac{1.0}{0.01} = 100
\] (6.2)

Since we are comparing against an incompressible reference solution, setting the Mach number to 0.1 is a good compromise between accuracy and efficiency.

6.2.3.3 Temporal discretization parameters

![TIMEDISC]

The parameter tend determines the end time of the solution, Analyze_dt the interval at which the analysis routines (like error computation, checking of wall velocities etc. see below) are called. The multiplier nWriteData determines the interval at which the solution state files
are written to the file system, e.g. in this case \( n\text{WriteData} \times \text{Analyze\_dt} = 0.1 \) is the interval for writing to disc. The CFL and DFL numbers determine the explicit time step restriction for the advective and viscous parts. Note that these values should always be chosen to be \(< 1\), but since the determination of the timestep includes some heuristics, both values should be chosen conservatively.

### 6.2.3.4 Analysis parameters

```
! ============================================================== !
! ANALYZE                                                        !
! ============================================================== !
CalcErrorNorms= T ! Calculate error norms
CalcBodyForces= T ! Calculate body forces
CalcWallVelocity= T ! Calculate wall velocities
CalcMeanFlux= T ! Calculate mean flux through boundaries
```

These switches trigger the output of analysis files.

Now that all the necessary preparations have been made, the simulation can be started.

### 6.2.3.5 Running the Simulation and Results

The command

```
flexi parameter_flexi.ini > std.out
```

runs the code and dumps all output into the file `std.out`. If you wish to run the code in parallel using MPI, the standard command is

```
mpirun -np XX flexi parameter_flexi.ini > std.out
```

where \( XX \) is an integer denoting the number of processes to be used in parallel. Note that \textsc{FLEXI} uses an element-based parallelization strategy, so the minimum load per process/core is one grid element, i.e. do not use more cores than cells in the grid!

If the simulation runs without error, the screen output should look similar to figure 6.6, showing information on the timestep and producing the analysis results selected via the parameter file. Once the run has completed, your working folder will contain a number of State files in HDF5 format and * .dat ASCII files for the analysis data, which can be opened in Tecplot, a spreadsheet tool etc.

Since we start the simulation from a fluid at rest, it will take some iterations / timesteps to achieve a steady state solution. One way to check if the solution has converged to a steady state is to check some characteristic quantities, e.g. the velocities at the walls, body forces etc. In figure 6.7, the temporal evolution of the velocities at the lower wall are plotted over time for 4 different simulations, and for all cases, at \( t_{\text{end}} = 5 \), a sufficiently stationary
solution has been achieved. Note that since the boundary conditions are applied weakly in a DG setting, a velocity slip at walls can occur, with its magnitude depending on the local wall resolution.

Figure 6.6: Sample screen output of the cavity simulation.

![Sample screen output](image)

Figure 6.7: Time evolution of wall velocity at lower wall for the $Re = 100$ lid-driven cavity case.

![Time evolution of wall velocity](image)

For comparison with published data, 4 simulations where run on the provided meshes. Details are listed in table 6.5.
Table 6.5: $Re = 100$ lid driven cavity: Simulations

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Mesh</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cavity_2x2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>cavity_4x4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>cavity_8x8</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>cavity_2x2</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure 6.8: Contours of velocity magnitude for the $Re = 100$ lid-driven cavity case.
A contour plot of the velocity magnitude at the end time is given in Figure 6.8. To generate this plot, convert the State files to a Paraview format by invoking

```
posti_visu parameter_postiVisu.ini parameter_flexi.ini your_Statefiles.h5
```

For a more quantitative comparison with published data, you can generate a plot of the $u-$velocity on the centerline ($x = 0$) of the cavity. Figure 6.9 shows the results for the 4 simulations run here, along with published data available in (Ghia 1982), (Gao 2016).

![Figure 6.9: Comparison of centerline velocities for the $Re = 100$ lid-driven cavity case with published results.](image)

For simulation 1, the agreement with literature results is fair. This is due to the coarse resolution with $2 \times (3 + 1) = 8$ degrees of freedom per x- and y-direction. Doubling the grid elements results in an improved match, doubling it again (simulation 3), the agreement with the published data is excellent. It should be noted that the same accuracy can be achieved by increasing $N$ and keeping the coarse grid. Simulation 3 and 4 have nearly identical results, although the number of degrees of freedom differs by a factor of 2. This is a feature of high order schemes for smooth problems.

### 6.2.4 Part II: Advanced Tutorial - Flow at $Re = 400$

In this part of the tutorial, we will expand upon what has been learned in the basic section above. The general setup of the computation remains the same, but the Reynolds number is increased, which will be accounted for by a new mesh and a higher resolution. Also, we will give a first glimpse at how to make your own changes to the code and show how to add a new function to be used as a custom initial or boundary condition. It is recommend that you...
have completed the basic tutorial, have access to at least 4 cores for the computation (or just have the patience to wait longer) and are familiar with **FORTRAN** syntax.

If not done so already, copy the `cavity` tutorial folder

```
cp -r $FLEXI_TUTORIALS/cavity .
```

and step into the `Advanced_Re400` subfolder.

### 6.2.4.1 Mesh Generation with HOPR

To account for the increased Reynolds number, the number of elements in the $x - y$-plane is increased to $12 \times 12$. Also, a stretching in the $y$–direction is introduced, as can be seen in picture 6.10. In `parameter_hopr.ini`, the following line facilitates an exponential stretching:

```
factor = (/1.0, -1.2, 1./) ! element stretching  
a constant growth factor (+/- changes direction)
```

You can generate your own mesh or re-use the provided one, labeled `cavity12x12_stretch_mesh.h5`.

![Figure 6.10: Stretched mesh for the $Re = 400$ lid-driven cavity case.](image)

### 6.2.4.2 Adding a custom initial or boundary function

Before running the simulation, we will include a custom function to be used as boundary condition for the top boundary, i.e. the velocity driving the flow in the cavity. In this, we follow the suggestions from (Gao 2016), where the $u(x)$ velocity at the lid is given as
6.2. LID-DRIVEN CAVITY

Chapter 6. Tutorials

\[
\begin{aligned}
    u(x) &= \begin{cases} 
        c_1 x^4 + c_2 x^3 + c_3 x^2 + c_4 x, & \text{if } 0 \leq x < 0.2 \\
        d_1 x^4 + d_2 x^3 + d_3 x^2 + d_4 x + d_5, & \text{if } 0.8 < x \leq 1.0 \\
        1, & \text{otherwise}
    \end{cases} 
\end{aligned}
\] (6.3)

with

\[
\begin{aligned}
    [c_1, c_2, c_3, c_4] &= 1000 \times [4.9333, -1.4267, 0.1297, -0.0033] \\
    [d_1, d_2, d_3, d_4, d_5] &= 10000 \times [0.4933, -1.8307, 2.5450, 1 - 0.5709, 0.3633]
\end{aligned}
\] (6.4)

This assumes the top boundary to be from 0...1, as is the case in our domain. To add this new function to FLEXI, locate the file "$FLEXI/src/equations/navierstokes/idealgas/exactfunc.f90" and open it in the text editor of your choice. Locate the \textit{SUBROUTINE ExactFunc}, which provides functions to the boundary, initial condition and analysis routines of the code.

The header of the routine you are looking for is shown in figure 6.11.

![Figure 6.11: Header for the subroutine ExactFunc](image)

To add equation 6.3 to the code, add a new \textit{CASE} to the routine, in which you define the state vector for the primitive variables, and then convert them to conservative ones (see e.g. case 8 for how this is done). You might also need to introduce some new local variables for this routine. To check if your changes are syntactically correct, compile the code with your changes by invoking \textit{make} from your build directory. If the compilation process was not successful, check the output on the screen that will give you some hints on what might be wrong.
6.2. LID-DRIVEN CAVITY

To benefit from this tutorial, it is recommend that you do try to complete this programming task. For reference, figure 6.12 shows a sample piece of code with a correct implementation as CASE 9. From now on, we will refer to CASE 9 as the case number in question, yours might be different of course.

```
CASE(9) !lid driven cavity flow from Gao, Hesthaven, Warburton
!"Absorbing layers for weakly compressible flows", to appear, JSC, 2016
!* Special "regularized" driven cavity BC to prevent singularities at corners
!* top BC assumed to be in x-direction from 0..1
Prim = RefStatePrim(IniRefState, :)
DO i=1,5
  x_p(i)=x(1)**(i-1)
END DO
vec_c(1:5)=1000*(/0.6,-0.0033,0.1297,-1.4267,4.9333/)
vec_d(1:5)=10000*(/0.3633,-1.5709,2.5458,-1.8307,0.4933/)

IF (x(1).LT.0.2) THEN
  prim(2)=DdT_PRODUCT(vec_c,x_p)
ELSEIF (x(1).LE.0.8) THEN
  prim(2)=1.0
ELSE
  prim(2)=DdT_PRODUCT(vec_d,x_p)
ENDIF
CALL PrmToCons(prim,Resu)
```

Figure 6.12: Sample code for the custom equation from (Gao 2016).

Note that in the sample code, we do not specify the full primitive state vector within the routine, but re-use the IniRefState and just overwrite the u-velocity. This is a matter of choice, but it allows to set the Mach number by setting the IniRefState accordingly.

6.2.4.3 Preparing the Flow Simulation with FLEXI

To setup the simulation, you can either modify the parameter_flexi.ini files from the basic tutorial or use the provided ones. We will conduct 2 simulations: one with the constant driving flow boundary conditions as before, and one with the new custom equation specified as equation CASE 9. Two parameter files for these cases are provided, the one with the custom boundary condition named parameter_flexi_custombc.ini. For both cases, the changes in the parameterfile are:

```
MeshFile = cavity12x12_stretch_mesh.h5
...
mu0 = 0.0025
...
tend = 100.0
```

The mesh file needs to be adjusted as well as the viscosity to set the Reynolds number. Also, the end time is set to 100 to account for the longer accommodation period the simulation will need to achieve a steady state. Setting it to 100 is very conservative, as we will see later, feel free to lower it as you see fit.
For the custom boundary condition case, the parameter file needs to be adjusted to address the new boundary settings. According to table 4.1, Dirichlet boundary conditions with a specified reference equation (instead of a reference state) are of type 22. The second index in the entry refers thus to the equation (CASE 9) we have programmed above.

\begin{verbatim}
   BoundaryName=BC_free
   BoundaryType=(/22,9/)
\end{verbatim}

Now we are ready to run both simulations and compare the results.

### 6.2.4.4 Running the Simulation and Results

The command

\begin{verbatim}
   flexi parameter_flexi.ini > std.out
\end{verbatim}

runs the code and dumps all output into the file std.out. If you wish to run the code in parallel using MPI, the standard command is

\begin{verbatim}
   mpirun -np XX flexi parameter_flexi.ini > std.out
\end{verbatim}

![Figure 6.13: Evolution of wall velocities at the lower wall for Re = 400 lid driven cavity simulations.](image)

Change the parameter file to `parameter_flexi_custombc.ini` for the second run. Note that you can adjust the end time if you wish. From figure 6.13, which shows the evolution of the mean velocity at the lower wall, the flow reaches steady state after about $t = 35$. The following figures show the flow field and comparison of the centerline velocities with published results.
6.3 Flow around a NACA 0012 airfoil

In this tutorial, the simulation around a NACA 0012 airfoil at $Re = 5000$ and $Ma = 0.4$ is considered. First, we explain how to set the main flow parameters. We then describe the
evaluation of lift and drag and visualization of the flow field. Finally, we show how to use the sponge zone to remove artificial reflections from the outflow boundary, so that a clean acoustic field is retained.

Figure 6.15: Contours of velocity magnitude for the flow around a NACA 0012 airfoil.

Copy the naca0012 tutorial folder to your desired working directory.

```
cp -r $FLEXI_TUTORIALS/ naca0012 .
```

### 6.3.1 Compiler options

Make sure that FLEXI is compiled with the cmake options listed in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAKE_BUILD_TYPE</td>
<td>Release</td>
<td></td>
</tr>
<tr>
<td>FLEXI_EQYNSYSNAME</td>
<td>navierstokes</td>
<td></td>
</tr>
<tr>
<td>FLEXI_PARABOLIC</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>FLEXI_MPI</td>
<td>ON</td>
<td>optional</td>
</tr>
</tbody>
</table>

The standard settings are sufficient for this example. To check whether they are set, change to your build folder and open the cmake GUI

```
cmake [flexi root directory]
```
If necessary, set the above options and then compile the code by issuing

\texttt{make}

### 6.3.2 Mesh Generation with HOPR

The mesh file used by FLEXI is created from the external linear mesh NACA0012_652.cgns and a 3rd order boundary description NACA0012_652_splitNg2.cgns using HOPR

```
hopr parameter_hopr.ini
```

This creates the mesh file NACA0012_652_Ng2_mesh.h5 in HDF5 format. If HOPR is not available, the mesh file is supplied in this tutorial.

### 6.3.3 Flow Simulation with FLEXI

The simulation setup is defined in parameter\_flexi.ini. The initial condition is selected via the variable vector RefState=(1, 0.990268069, 0.139173101, 0, 4.4642857/) which represents the vector of primitive solution variables \((\rho, u, v, w, p)\). The chosen velocity vector \((u, v)^T\) yields an angle of attack of \(\alpha = 8^\circ\) and a velocity magnitude of 1. The first supplied RefState is given the number 1, the second is given the number 2 and so on.

IniRefState = 1: the initial condition uses RefState 1 for the initial flow field solution.

IniExactFunc = 1: exact function routine for initialization, case 1 initializes a freestream state based on IniRefState.

Material properties are given in table 6.7. Based on the ideal gas law, we get

\[
Ma = 1/\sqrt{\kappa p/\rho} = 0.4
\]

Note that in this non-dimensional setup the mesh is scaled such that the chord length is unity, i.e. \(C = 1\). Then to arrive at \(Re = \rho u C/\mu = 5000\), the viscosity is set to

\[
\mu = \rho u C/Re = 1/Re = 0.0002
\]

<table>
<thead>
<tr>
<th>Property</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic viscosity (\mu)</td>
<td>mu0</td>
<td>0.0002</td>
</tr>
<tr>
<td>ideal gas constant (R)</td>
<td>R</td>
<td>4.4642857</td>
</tr>
<tr>
<td>Prandtl number (Pr)</td>
<td>Pr</td>
<td>0.72</td>
</tr>
<tr>
<td>isentropic coefficient (\kappa)</td>
<td>kappa</td>
<td>1.4</td>
</tr>
</tbody>
</table>
6.3.4 Numerical settings

The DG solution on the mesh is represented by piecewise polynomials and the polynomial degree in this tutorial is chosen as $N = 3$.

The main code settings are displayed in table 6.8.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Polynomial degree</td>
<td>3</td>
</tr>
<tr>
<td>MeshFile</td>
<td>Mesh file to be used</td>
<td>NACA0012_652_Ng2_mesh.h5</td>
</tr>
<tr>
<td>tend</td>
<td>end time of the simulation</td>
<td>10</td>
</tr>
<tr>
<td>Analyze_dt</td>
<td>time interval for analysis</td>
<td>0.01</td>
</tr>
<tr>
<td>nWriteData</td>
<td>dump solution every n’th Analyze_dt</td>
<td>10</td>
</tr>
<tr>
<td>CFLscale</td>
<td></td>
<td>0.9</td>
</tr>
<tr>
<td>DFLscale</td>
<td></td>
<td>0.9</td>
</tr>
</tbody>
</table>

6.3.5 Boundary conditions

The boundary conditions were already set in the mesh file by hopr. Thus, the simulation runs without specifying the boundary conditions in the FLEXI parameter file. The freestream boundaries of the mesh are Dirichlet boundaries using the same state as the initialization, e.g. IniRefState by default.

The boundary types and states used by the simulations can always be checked in the initial console output of FLEXI. In order to run a quick test run, first set

```
maxiter=1
```

in your parameter file to enforce the simulation to stop after one time step. Then run the code

```
flexi parameter_flexi.ini
```

The code will finish after one time step. The initialization output contains the boundary conditions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>State</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_wall</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BC_inflow</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BC_outflow</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BC_zminus</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BC_zplus</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>
Type 2 identifies the weak Dirichlet boundary condition, type 3 the adiabatic wall. The boundary conditions in $z$ direction are not relevant for this 2D example and are realized as periodic boundaries.

Suppose we want to change the wall boundary condition from adiabatic to isothermal.

- This is done by first defining a second RefState. Add to the parameter file the second of the following lines

```
RefState=(/1.,0.990268069,0.139173101,0.,4.4642857/)
RefState=(/1.,0.990268069,0.139173101,0.,4.9107143/)
```

Note that in the second RefState the pressure was increased by 10%. From the ideal gas law, we then know that the temperature also is raised by 10%, since the density has been kept constant. Also note that the isothermal wall ignores the velocity components but only uses the thermodynamic quantities.

- Second, overwrite the initial wall boundary condition

```
BoundaryName=BC_wall
BoundaryType=(/4,2/)
```

Here, 4 implies the use of the isothermal boundary condition and 2 indicates the RefState to be used for the wall. Note that the temperature is set indirectly by the RefState given in terms of $\rho$ and $p$.

To check the newly set boundary condition, run the code once more for one iteration. The output should now look like this

```
Boundary in HDF file found | BC_wall
was | 3 0
is set to | 4 2
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>State</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_wall</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>BC_inflow</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BC_outflow</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BC_zminus</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BC_zplus</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

Now the wall boundary condition is isothermal using the temperature defined by the second RefState listed in your parameter file. A complete overview over the boundary conditions and how to use them is given in Section 4.3.2.

Before starting your simulation, remember to disable the `maxiter` line again by setting it to -1.
6.3.6 Running the code

We proceed by running the code in parallel. For example using 4 processors, use the following command

```
mpirun -np 4 flexi parameter_flexi.ini
```

On a 2012 laptop with core i5 processor, this simulation takes about 40 minutes.

6.3.7 Evaluation of the lift and drag forces

The forces acting on the airfoil are one of the main desired output quantities from the simulation. They are calculated on the fly during runtime. The associated flags in the parameter file are

```
CalcBodyForces=T
WriteBodyForces=T
```

The first line activates the calculation of the forces at each `Analyze_dt`, the second line enforces output of the forces to a `.dat` file.

The body forces are a good measure for convergence. In the context of time-dependent flows this determines whether the solution has reached a quasi steady state. The figure 6.16 shows the \(x\) and \(y\) components of the force acting on the airfoil until `TEnd=10`.

![Resulting forces on the airfoil up to \(t = 10\).](image)

The lift and drag coefficients can be easily calculated by rotating these forces from the computational reference frame to the one of the freestream.
From the forces, it is clear that the steady state has not yet been reached and the simulation must be run further. Before we proceed with the simulation, we will nonetheless examine the preliminary results to check the quality of the simulation.

### 6.3.8 Wall velocities

Due to the weak coupling between the grid cells and to boundaries, boundary conditions are enforced weakly, e.g. by applying a specific flux. This adds largely to the stability of the scheme. However, as a result the no-slip condition at the wall is not exactly fulfilled by the numerical solution. Rather, it is approximated as far as the resolution allows.

Evaluation of the velocity vector near the wall helps quantifying this error, which can be seen as a quality measure for the near wall resolution. In the parameter file, the computation and output to a .dat file of the average and extreme values of the wall velocity at every \text{Analyze\_dt} are activated by setting

\text{CalcWallVelocity}=\text{T}
\text{WriteWallVelocity}=\text{T}

During the computation, we get output like the following:

<table>
<thead>
<tr>
<th>Wall Velocities (mean/min/max) :</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_wall 2.737875251E-02 3.639096145E-04 6.228999317E-01</td>
</tr>
</tbody>
</table>

In our case, the wall velocity is on average at about 3\% of the freestream velocity, reaching a peak of 60\%. This peak typically occurs at the quasi-singularity at the trailing edge. To decrease this deviation from the theoretical no-slip condition, either the wall-normal mesh size must be decreased or the polynomial degree increased.

It is important to note that both of the above measures will, besides increasing the number of degrees of freedom, *decrease the time step*, which directly affects the computational time. Thus, it is important to achieve an acceptable trade-off between the acceptable error and the computational time.

In this tutorial, the observed slip velocity is deemed uncritical and we proceed with the same resolution.

### 6.3.9 Visualization

To visualize the solution, the \textit{State}-files must be converted into a format suitable for \textit{Paraview}. Issue the command

\texttt{mpirun -np 4 \textit{posti\_visu parameter\_postiVisu.ini parameter\_flexi\_navierstokes.ini NACA0012\_Re5000\_AoA8\_State\_0000000.0*}}
to generate the corresponding \textit{vtu}-files using supersampling with \((NVisu+1)^3\) points in each cell. The \textit{vtu} file can then be loaded into \textbf{Paraview}. Figure 6.17 shows a visualization of the density at \(t = 10\). The levels of \(0.99 < \rho < 1.01\) are chosen to reveal the acoustic radiation from the airfoil.

![Image](image.png)

\textbf{Figure 6.17:} Contours of density with \(0.99 < \rho < 1.01\) at \(t = 10\) for the flow around a NACA 0012 airfoil.

The large scale vortex shedding of the wake due to the high angle of attack is clearly visible. Acoustic radiation from the airfoil can also be observed.

Now, a problem becomes apparent: the vortex street propagating towards the outflow boundary results in a second, artificial acoustic source at the outflow boundary. This is one of the fundamental problems in direct aeroacoustic computations. Before we proceed with the simulation, we will now make use of the sponge zone functionality of \textbf{FLEXI} to remove this artificial source.

\textbf{6.3.10 Remove outflow reflections using the sponge zone}

The sponge zone introduces a dissipative source term to the discrete operator, which is only active in a user-specified region, typically upstream of the outflow boundary. We use the sponge zone to dampen the vortices convected downstream before they hit the outflow boundary. See (Flad et al. 2014) for the background of our sponge zone implementation.

In order to activate the sponge zone, set

```
SpongeLayer=T
```

in the parameter file.
Most of the other parameters are already set for the current simulation:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpongeShape</td>
<td>Set shape of sponge: 1: cartesian</td>
<td>1</td>
</tr>
<tr>
<td>damping</td>
<td>Damping factor of sponge</td>
<td>1.0</td>
</tr>
<tr>
<td>xStart</td>
<td>Coordinates of start position of sponge ramp (for SpongeShape=1)</td>
<td>(2.0,0,0)</td>
</tr>
<tr>
<td>SpongeDistance</td>
<td>Set shape of sponge: 1: Cartesian sponge region.</td>
<td>3.0</td>
</tr>
<tr>
<td>SpongeDir</td>
<td>Direction vector of the sponge ramp</td>
<td>(/1,0,0/)</td>
</tr>
<tr>
<td>(for SpongeShape=1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpongeBaseFlow</td>
<td>Type of baseflow to be used for sponge</td>
<td>4</td>
</tr>
<tr>
<td>tempFilterWidth</td>
<td>Temporal filter width used to advance</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>Pruett baseflow in time.</td>
<td></td>
</tr>
<tr>
<td>SpongeViz</td>
<td>Write a visualization file of the</td>
<td>T</td>
</tr>
</tbody>
</table>

The source term is of the form

\[ \tilde{U}_t = U_t - d\sigma(\vec{x}) (U - U_B) \]  \hspace{1cm} (6.5)

First, damping determines the strength of the source term, i.e. \( d \) in the above equation. It is dependent on the mean convection velocity, the desired amount of amplitude reduction and the thickness of the sponge zone. Typically, some trial and error is necessary to obtain an appropriate value. In non-dimensional calculations, i.e. velocity and length scale are of \( O(1) \), \( d = 0.1\ldots2 \).

Ramping of the source term from 0 is necessary to avoid reflections at the sponge interface. If such reflections occur, it is necessary to choose a thicker sponge ramp so that the source term is ramped up less steeply. We choose a parallel ramp by setting SpongeShape=1. The ramp’s starting position, thickness and direction are set by xStart, SpongeDistance and SpongeDir, respectively. These parameters govern the shape function \( \sigma(\vec{x}) \) which smoothly ramps the source term from 0 to 1.

It is important to emphasize that the regions where the source term is active should not be interpreted physically. They should be considered a part of the boundary condition. Likewise, they must be placed sufficiently far downstream of the airfoil such that they do not influence the near field solution. With the chosen settings, the sponge zone starts one chord behind the airfoil and is ramped up to 1 at the outflow boundary, located 4 chords behind the airfoil.

In order to visualize the ramping function \( d\sigma(\vec{x}) \), set SpongeViz=T. For that purpose, change the project name to e.g. “test” and set maxiter=1 to immediately stop the simulation after...
one iteration. This will output a .vtu file ready for visualization. After that, don’t forget to change back the project name and reset maxiter=-1.

Now, we have to choose the desired baseflow ($U_B$). For the current configuration, the moving average ($\text{SpongeBaseFlow}=4$) is appropriate. It produces a mean field slowly progressing in time, which adapts to the potential flow around the airfoil. The parameter tempFilterWidth determines the effective time window for the moving average. It should be chosen somewhat larger than the largest time scales to be damped. Its value of 2.0 here is chosen based on the frequency of the oscillations in the body forces seen in Figure 6.16.

The moving average base flow needs an initial value. One option is to provide an initial flow field from a file using the $\text{SpongeBaseflowFile}$ parameter. If this parameter is not set, the code will initialize the base flow with the same values as the solution itself. Thus, in the case of a fresh computation, the base flow will be initialized with the $\text{IniExactFunc}$ used to initialize the solution. In the case of a restart, like in the present tutorial, the base flow will be initialized with the state file provided for the restart of the simulation.

Note that using the moving average baseflow, the code will dump additional *.baseflow*.h5 files to the hard drive, which are necessary to restart the simulation with the corresponding base flow from a given time. If these files are available and exhibit the same project name as the current simulation, the code automatically detects the right base flow file to initialize the base flow at the restart time. However, if in that situation $\text{SpongeBaseflowFile}$ is still provided in the parameter file, the moving average will be restarted using the flow field provided in the parameter file, which may not be desirable.

### 6.3.11 Restarting the simulation

If the simulation run is interrupted or if you decide to run the simulation further than $\text{TEnd}=10$, FLEXI can easily be restarted. In the current setting, the code dumps the solution in 0.1 time units intervals. Since we want to progress our simulation further, set $\text{TEnd}=25$ in the parameter file. Since we have now turned on the sponge zone, it is also advisable to modify the project name, i.e.

```
ProjectName=NACA0012_Re5000_AoA8_SP
```

If the state files are in the current folder, restart the simulation as follows

```
mpirun -np 4 flexi parameter_flexi.ini
NACA0012_Re5000_AoA8_State_000010.000000000.h5
```

Note that it is also possible to change the polynomial degree of the simulation during restart. This can be used to generate a rough initial solution at low $N$ and then switch to a high quality approximation at higher $N$ later. The code will automatically project the solution onto the new polynomial basis at startup. Restart is however only possible using the same mesh file.
Once the simulation is done, visualize the results again following the instructions in Section 6.3.9.

The following two figures 6.18 and 6.19 show the resulting instantaneous density contours with (Figure 6.18, $t = 25$) and without sponge zone (Figure 6.19, $t = 20$). Clearly, the source at the outflow is damped to uncritical levels. Fine-tuning of the sponge parameters may improve the results.

![Image](image.png)

Figure 6.18: Contours of density with $0.99 < \rho < 1.01$ at $t = 25$ for the flow around a NACA 0012 airfoil with moving average sponge zone.

Another look at the body forces indicates that a quasi-steady state has been reached.

### 6.3.12 Two-dimensional computation

The laminar flow around this airfoil is in nature two-dimensional. Nonetheless, we ran this simulation using a three-dimensional code by imposing periodic boundary conditions and only one mesh element in the spanwise direction. This means we

- Compute one unnecessary variable (momentum in spanwise direction),
- Compute three-dimensional fluxes for all variables,
- Compute one unnecessary gradient,
- Use several degrees of freedom in the spanwise direction due to the high order ansatz in each element.

To circumvent this, FLEXI provides the option to perform truly two-dimensional calculations. To enable this, we need to set the flag FLEXI\_2D to ON during configuration. FLEXI will then perform two-dimensional computations, but you will need to provide a mesh that consists of only one element in the third dimension. Only then will you be able to start the
Figure 6.19: Contours of density with $0.99 < \rho < 1.01$ at $t = 20$ for the flow around a NACA 0012 airfoil without sponge zone.

Figure 6.20: Resulting forces on the airfoil up to $t = 25$. 
computation. Since this is already the case for the mesh used in this tutorial, we can start a
two-dimensional computation right away after we compile our code with FLEXI_2D set to
on. So switch to your build directory, set the CMAKE option FLEXI_2D to on, configure
and compile your code once more. Then switch back to the tutorial folder.

You can now immediately run the code again using the same parameter file, but your
new executable. All the options are the same for two-dimensional and three-dimensional
computations. For compatibility reasons, you will have to specify vectorial parameters using
three dimensions (e.g. the momentum in the RefState), but the third dimension will be
ignored. Pay attention to how much faster your code will run when using the two-dimensional
version.

By default the State-files will be written like three-dimensional data by extruding the solution
in the third dimension so the files will be compatible with all already existing post-processing
tools. You can set the option Output2D to TRUE if you want to write two-dimensional files,
thus reducing the size of the output considerably. Also keep in mind that no matter if two- or
three-dimensional, the number of dimensions of arrays will always be the same (one dimension
simply being of size 1 for two-dimensional computations) and also the number of variables
will be the same. This means we will still have a momentum in the third dimension, which is
simply set to zero (no calculations will be done for this momentum).

6.3.13 Using record points

Let’s assume we are interested in the temporal evolution of the flow variables in the vicinity
of the profile’s upper and lower surfaces. Using the State-files, we will always have the whole
flow field. If we want to have a very fine temporal resolution or our field is very large, the
cost of writing and storing these files can get prohibitive. As a remedy, FLEXI provides an
option to sample specific points in the flow field at a very fine temporal resolution (down to
a single time step) and only output those specific points. We call these points record points.

If record points should be used, three distinctive steps have to be taken:

- Create the record points themselves
- Run a simulation with active record points
- Visualize the results

The first and third step are performed with pre- and postprocessing tools respectively, while a
simulation using record points simply requires to set the specific options in FLEXI.

6.3.13.1 Prepare the record points

The coordinates of the record points are defined using a POSTI tool called preparerecord-
points, which is build when the respective CMake option is set. The tool takes a single
parameter file as an input, so it can be run like
6.3. FLOW AROUND A NACA 0012 AIRFOIL

A sample parameter file is included in the NACA0012 tutorial folder. You need to specify the mesh that should be used to create the record points and a project name. The options *NSuper* and *maxTolerance* control the algorithm that tries to find the record points in the mesh - a good default for *NSuper* is to set it to at least twice of the polynomial degree of your mesh. The main part of the parameter file is the actual definition of the record points. They are created in so called sets, and each set is part of a named group. In the example, we have two sets - one for the suction side and one for the pressure side of the airfoil - each assigned to a separate group. There are several different types of sets available, which can be used to create single points, lines or planes of record points. For the NACA example, we are using the *boundary layer plane* type of set. This special type of plane is defined by a spline which is projected onto the nearest boundary. Along this projected line, a user-specified number of points is distributed in an equidistant way. The plane is then created by following the normal vector of the boundary at each of the points for a specified distance. Along the normal direction, the record points can be stretched to cluster them near a wall.

The definition of one of the groups looks like this:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupName</td>
<td>suctionSide</td>
</tr>
<tr>
<td>BLPlane_GroupID</td>
<td>1</td>
</tr>
<tr>
<td>BLPlane_nRP</td>
<td>(/20,30/)</td>
</tr>
<tr>
<td>BLPlane_nCP</td>
<td>2</td>
</tr>
<tr>
<td>BLPlane_CP</td>
<td>(/0.9,0.014,0.5/)</td>
</tr>
<tr>
<td>BLPlane_height</td>
<td>0.05</td>
</tr>
<tr>
<td>BLPlane_CP</td>
<td>(/0.999,0.001,0.5/)</td>
</tr>
<tr>
<td>BLPlane_height</td>
<td>0.05</td>
</tr>
<tr>
<td>BLPlane_fac</td>
<td>1.04</td>
</tr>
</tbody>
</table>

The first parameter creates a group which is used for identification purposes - each set needs to be assigned to a group. Then, the new set is defined by setting the respective *GroupId* option, in this case *BLPlane_GroupID*. The ID is set to 1 to assign it to the first group. Following this general definitions, the actual parameters of the set follow. For the boundary layer plane, we first define the number of the points used in wall-tangential and wall-normal direction using the *BLPlane_nRP* option. The next six options define the spline that is projected onto the boundary. First we set the number of control points in the boundary, then define the coordinates of each of the points and the height in the wall-normal direction at this point. The last option defines the stretching that is used in the wall-normal direction.

To find out how the other types of sets are defined, have a look at the descriptions of their parameters.

If you now run the preparerecordpoints tool using the syntax given above, the algorithm will first calculate the physical coordinates of the recordpoints following your definitions. Then, it will attempt to identify the element in the mesh that contains each record point and the coordinates in the reference element therein. This is done by inverting the mapping from
reference to physical space using Newton’s method. This information is needed to later
interpolate the polynomial solution to exactly the positions of the record points. The results
are then written to a file called ‘ProjectName_RPSet.h5’. If you set the doVisuRP option to
true, you will also get a visualization of the record points that can be viewed in ParaView.

### 6.3.13.2 Using record points

To actually use the recordpoints during a simulation, you simply need to set a few options in
the FLEXI parameter file. The import ones are the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP_inUse</td>
<td>T</td>
</tr>
<tr>
<td>RP_DefFile</td>
<td>NACA0012_RPSet.h5</td>
</tr>
<tr>
<td>RP_SamplingOffset</td>
<td>1</td>
</tr>
</tbody>
</table>

With the first option, the general usage of the record points system can be turned on or off.
The RP_DefFile option must then be set to the name of the file that has been created using
the preparerecordpoints tool. You can specify that the solution should be sampled at every
RP_SamplingOffset timestep.

If you now run the simulation, besides the State-files additional RP-files will be created,
containing the value of the conservative variables at each record point at every sampling time
step.

### 6.3.13.3 Visualizing the solution at the record points

The visualizerecordpoints tool is used to post-process the recordpoints data. It can be used to
merge several of the RPSet.h5 files together and create a longer timeseries. Besides simply
visualizing the timeseries, there are a multitude of post-processing options available. You can
use the time series to determine the mean and fluctuating part of the solution or perform
spectral analysis using FFTs. For boundary layer planes, different types of turbulent quantities
like the skin friction can be directly computed.

As an example, we want to calculate the mean flow $\bar{U}$ and the temporal fluctuations $U'$
at our two record points planes. Again, a sample parameter file for the tool is given in the
tutorial folder. The options set are

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectName</td>
<td>NACA0012</td>
</tr>
<tr>
<td>GroupName</td>
<td>suctionSide</td>
</tr>
<tr>
<td>GroupName</td>
<td>pressureSide</td>
</tr>
<tr>
<td>RP_DefFile</td>
<td>NACA0012_RPSet.h5</td>
</tr>
<tr>
<td>OutputTimeAverage</td>
<td>T</td>
</tr>
<tr>
<td>doFluctuations</td>
<td>T</td>
</tr>
</tbody>
</table>

where besides setting a name for the project and giving the path to the file that contains
the record point definition, we define the name of the groups we want to visualize. They
must be the ones we defined when using the preparerecordpoints tool. Regarding actual
evaluation, we want to calculate the temporal average and the fluctuations, simply calculated
as \( U' = U - \bar{U} \). We did not specify any variables that we want to visualize, in that case
all conservative variables that are stored in the \( RPSet.h5 \) files will be used. If you want
to visualize a specific or a derived variable (e.g. the pressure), it can simply be set in the
parameter file like this:

\[
\text{VarName} = \text{Pressure}
\]

You can run the tool using

\[
\text{posti_visualizerecordpoints parameter_visualizeRecordpoints.ini NACA0012_Re5000_AoA8_RP_*}
\]

This will take all the time samples recorded during our simulation as input. For each of the
planes, a separate .vts file containing the temporal average will be written. The fluctuations
are time-resolved data, and due to limitations in the VTK file format every time step has to be
written to a single file. To avoid creating a lot of files in the working directory, those will be
stored in a subfolder called \( \text{timeseries} \). In the working directory, .pvd files for the fluctuations
are created. These files can be opened with ParaView and will contain the complete time
series with the correct time value.

### 6.4 Convergence Test

In this tutorial the order of convergence for \texttt{FLEXI} is computed. The procedure is fully
scripted, such that in the end a number of runs have been performed on a variation of grids
or a variation of polynomial degrees and the order of convergence is computed automatically.
A plot of the corresponding L2 error norms is produced and copied into the directory where
the convergence test is executed from. Note, the script is written in \texttt{Python 2.7}.

The convergence test is separated into two parts, first an inviscid test where the order of
convergence for the advective part, i.e. the Euler equation without viscous fluxes, is computed.
Then a viscous convergence test calculates the order of convergence with consideration of
the viscosity.

In this tutorial, the following topics are dealt with:

- Calling \texttt{FLEXI} using a Python script
- Compute the order of convergence
- Switch from a Navier-Stokes simulation to an Euler simulation
- Treatment of source terms in \texttt{FLEXI}
- Execution of \texttt{FLEXI} with Mortar meshes
6.4.1 Manufactured Solution

The basic principle of computing the order of convergence of FLEXI is to apply a benchmark test where an exact analytical solution is known. We apply here the method of manufactured solutions. A detailed description is found in Roache (Roache 2001). In general, a smooth function is proposed as solution to the equation system. Since the proposed function is generally not a solution to the equation system, a source term is calculated to force the corresponding solution. Therefore, the function is derived analytically and inserted into the equation system, cf. for the continuity equation

\[ \rho_t + (\rho u)_x = Q(x,t) \quad \text{with} \quad \rho = A + \sin(B(x,t)) \quad \text{and} \quad u = \text{const.} \quad (6.6) \]

The source term has to be added accordingly within the time integration loop of the flow solver. In FLEXI, sine waves in the density are advected on a constant velocity field. For the different equation systems, i.e. Euler or Navier-Stokes, different source terms have to be considered.

6.4.2 Inviscid Convergence Test

Copy the convtest tutorial folder

```bash
cp -r $FLEXI_TUTORIALS / convtest .
cd convtest
```

6.4.2.1 Compiler Options

Since this test case is valid for the Euler equations only, the code has to be compiled without parabolic fluxes. Hence, make sure to compile FLEXI with the cmake options in the following table

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAKE_BUILD_TYPE</td>
<td>Release</td>
<td></td>
</tr>
<tr>
<td>FLEXI_EQYNYSNAME</td>
<td>navierstokes</td>
<td></td>
</tr>
<tr>
<td>FLEXI_PARABOLIC</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>FLEXI_MPI</td>
<td>ON</td>
<td>optional</td>
</tr>
</tbody>
</table>

Table 6.10: Cmake options for the convergence test simulation.
6.4.2.2 Mesh Generation with HOPR

The mesh files for this tutorial can be found in the tutorial directory. Parameter files are provided to produce the meshes using HOPR. With `parameter_hopr.ini` conform meshes are created and the file `parameter_mortar_hopr.ini` generates non-conform meshes.

6.4.2.3 Manufactured Solution For The Inviscid Case

The manufactured solution is

\[ \rho = A \ast (1 + B \ast \sin(\Omega \ast |x - vt|)) \quad \text{with} \quad A, B, \Omega, v = \text{const.} \]  

(6.7)

The advantage is that for the Euler equation, the resulting source term is zero, \( Q(x, t) \equiv 0 \).

As mesh, a Cartesian box is used with periodic boundaries. The mesh and the corresponding solution are shown in following figure:

![Figure: Mesh and flow field solution of the density. View in x-y-plane.](image)

To investigate the order of convergence of a given polynomial degree \( N \), the mesh resolution has to increase. We provide meshes with 1, 2, 4 and 8 elements in each spatial direction. They are provided in the tutorial directory with an according parameter file for the preprocessing tool HOPR.

6.4.2.4 Flow Simulation with FLEXI

The inviscid convergence test is run from the parameter file `parameter_convtest_flexi.ini`. Essentially, any valid parameter file can be used, as a manufactured solution is simulated.
This allows to test the various methods and features of the code and investigate their order of convergence. However, for this tutorial we restrict the parameter file to a very simple baseline test case. The following entries have to be made:

**IniExactFunc = 2** : The manufactured solution is applied by means of the exact function. This function is used to initialize FLEXI and can in general be used for Dirichlet boundary conditions. Here, we apply only periodic boundary conditions.

**N_Analyze = 10** : The number of interpolation nodes for the analyze routines, needed for calculating the error norms. We suggest at least $N_{\text{Analyze}} = 2N$.

**AdvVel = (/0.3,0.,0./)** : The constant velocity vector $\vec{v}$

**CalcErrorNorms = T** : Flag to turn on the calculation of $L_2$ and $L_{\infty}$ error norms

**ProjectName = ConvTest** : Project name used to name the output files of FLEXI and the convergence test scripts.

### 6.4.2.5 Numerical settings

The default settings for the time integration are displayed in table 6.11.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tend</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>Analyze_dt</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>nWriteData</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>CFLscale</td>
<td></td>
<td>0.9</td>
</tr>
<tr>
<td>DFLscale</td>
<td></td>
<td>0.9</td>
</tr>
</tbody>
</table>

The remaining numerical settings necessary, e.g. the polynomial degree and the mesh filename are set via the script file. The script can be found in the directory:

```
$FLEXIROOT/tools/convergence_test
```

Two scripts are provided, the file `convergence_grid` calculates the order of grid convergence for a given polynomial degree $N$ with increasing mesh resolution. The file `convergence` calculates spectral convergence on a given mesh with increasing polynomial degree. Both scripts are written in Python 2.7. In the first case of grid convergence, the polynomial degree and the set of meshes can be adjusted. Here we choose a polynomial degree of 3, i.e. the theoretical order of convergence is $N + 1 = 4$. The spectral convergence is calculated for polynomials of degree $N \in [1, 10]$ on a mesh with 4 elements in each spatial direction.

The command
6.4. CONVERGENCE TEST

 runs the code. The standard output of **FLEXI** is written into the logfile `ConvTest.log`. An ASCII file `ConvTest_convfile_grid.csv` is written that includes all L2 and L_inf error norms for the state vector \( U \) for all meshes and the corresponding orders of convergence. Furthermore, a PDF file `ConvTest_convtest_grid.pdf` is generated that plots the L2 error of the momentum in \( x \)-direction against the number of elements of the meshes. Another curve represents the theoretical order of convergence for the chosen polynomial degree.

Spectral convergence can be investigated using the command

```bash
$FLEXIROOT/tools/convergence_test/convergence $FLEXIDIR/bin/flexi parameter_convtest_flexi.ini --gnuplot
```

Corresponding files are produced, where `_grid` is replaced by `_N`. Figure 6.21 shows the result for grid convergence.

![ConvTest X-Momentum](image)

Figure 6.21: Plot of grid convergence

Figure 6.22 shows the result for spectral convergence.

6.4.3 Viscous Convergence Test

The second convergence test includes the viscous terms. Make sure, **FLEXI** is compiled with the setting listed in the table 6.12 to include parabolic fluxes.
6.4. CONVERGENCE TEST

Table 6.12: Cmake options for the convergence test simulation.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAKE_BUILD_TYPE</td>
<td>Release</td>
<td></td>
</tr>
<tr>
<td>FLEXI_EQYNSYSNAME</td>
<td>navierstokes</td>
<td></td>
</tr>
<tr>
<td>FLEXI_PARABOLIC</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>FLEXI_MPI</td>
<td>ON</td>
<td>optional</td>
</tr>
</tbody>
</table>

For this case, another manufactured solution is chosen

\[ \rho = 2 + A \sin(\Omega |x| - v\pi t) \quad \text{with} \quad A, \Omega, v = \text{const.} \quad (6.8) \]

The same function is applied to the momentum in all spatial directions. The mass specific total energy in this case is \( \rho e = \rho p \). This manufactured solution has a non-zero source term. In FLEXI, this source term is added in the routine CalcSource in the file

\$FLEXIROOT/src/equations/navierstokes/idealgas/exactfunc.f90

Note that this manufacture can also be solved without considering the viscous terms. In this case the source term does not vanish.

The parameter file for this case is parameter_convtestvisc_flexi.ini. Some modifications have to be made, compared to the inviscid convergence test. The correct exact function for this test case is IniExactFunc = 4. Note, that a viscosity has to be prescribed. Our default value for this case is \( \mu_0 = 0.03 \).
6.5. TAYLOR GREEN VORTEX

Execution of the convergence tests is analogously to the inviscid case, e.g.

```
$FLEXIROOT/tools/convergence_test/convergence_grid $FLEXIDIR/bin/flexi parameter_convtestvisc_flexi.ini --gnuplot
```

6.4.4 Execution of the Convergence Test with non-conform meshes

FLEXI is capable of using non-conform meshes with mortar interfaces. To illustrate the execution of FLEXI with these meshes, four mortar meshes are provided for the convergence test. Their file names include the term MORTAR. To use the convergence test script, just open the script file at $FLEXIROOT/tools/convergence_test/convergence_grid and exchange the file names of the meshes. The script can then be run similar as described above.

6.4.5 Execution of Convergence Test Outside of Tutorials

The convergence test scripts are provided in the directory

```
$FLEXIROOT/tools/convergence_test
```

including the Python script to execute FLEXI

```
$FLEXIRoot/tools/convergence_test/execute_flexi.py
```

6.5 Taylor Green Vortex

This tutorial describes how to set up and run the basic test case for turbulent flows, the Taylor-Green-Vortex (TGV) - see e.g. (Gassner and Beck 2013). We will learn how to avoid catastrophic failure of the code due to non-linear instabilities. This is done by using polynomial de-aliasing. In a second step we add the sub grid scale model of Smagorinsky. The tutorial assumes that you are familiar with the general FLEXI and HOPR work flow (please finish the previous tutorials first if this sounds strange to you).

6.5.1 Flow description

The initial condition to the (TGV) is a sinus distribution in the u and v velocity components. This leads to rapid production of turbulent structures, after a short initial laminar phase. While the test case is incompressible in principle, we solve it here in a compressible setting. The chosen Mach number with respect to the highest velocity in the field is 0.1. The Reynolds
number of the flow is defined as $1/\nu$. The domain is set up as a triple periodic box with edge length $2\pi$.

### 6.5.2 Compiler options

Make sure that **FLEXI** is compiled with the cmake options listed in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAKE_BUILD_TYPE</td>
<td>Release</td>
<td></td>
</tr>
<tr>
<td>FLEXI_EQYNNSYSNAME</td>
<td>navierstokes</td>
<td></td>
</tr>
<tr>
<td>FLEXI_PARABOLIC</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>FLEXI_MPI</td>
<td>ON</td>
<td>optional</td>
</tr>
<tr>
<td>FLEXI_EDDYVISCOSITY</td>
<td>ON</td>
<td>optional</td>
</tr>
<tr>
<td>FLEXI_TESTCASE</td>
<td>taylorgreenvortex</td>
<td>optional</td>
</tr>
</tbody>
</table>

For others you may keep the default values. Compile the code.

### 6.5.2.1 Mesh Generation with HOPR

We use a mesh with 4 cells per direction for the tutorial. In case you want to generate other meshes the parameter file for **HOPR** is included in the tutorial directory (parameter_hopr.ini), the default mesh is included. Using 4 cells with a polynomial degree of $N = 7$, means we use the typical large eddy setup of 32 DOF per direction.
6.5.3 Tutorial - Flow at $Re = 1600$

Copy the tgv tutorial folder to your working directory

```
cp -r $FLEXI_TUTORIALS/taylorgreenvortex .
```

Step into the folder. In case you do not want to generate the mesh files yourselves, the meshes have already been provided.

6.5.3.1 Preparing the Flow Simulation with FLEXI

The simulation setup is defined in `parameter_flexi.ini`. To get help on any of the parameters listed therein, you can run FLEXI from the command line by typing

```
flexi --help
```

The parameters in the file are grouped thematically, however, this is not mandatory. All lines starting with a “!” are comments.

6.5.3.1.1 Output

The test case has its own analyze output (PROJECTNAME_TGVAnalysis.csv), that we will use. We don’t look at flow visualization in this tutorial. Besides other interesting quantities, the file contains the incompressible dissipation rate. This is the resolved dissipation of the gradient field, computed as the integral over the domain of the strain rate tensor norm $S_{ij}S_{ij}$, times viscosity times 2. It is stored in the second column of the file. We will use this quantity in the tutorial to verify your results.

6.5.3.1.2 Interpolation / Discretization parameters

```
! ===================================================================
! INTERPOLATION
! ===================================================================

N = 7
```

The parameter $N$ sets the degree of the solution polynomial, e.g. in this example, the solution is approximated by a polynomial of degree 7 in each spatial direction. This results in $(N + 1)^3$ degrees of freedom for each (3D) element. In general, $N$ can be chosen to be any integer greater or equal to 1, however, the discretization and the timestep calculation has not extensively been tested beyond $N \approx 23$. Usually, a good compromise of performance and accuracy is found for $N \in [3, \ldots, 9]$.

To apply polynomial de-aliasing there are the following options:
6.5. TAYLOR GREEN VORTEX

FLEXI has three ways of doing polynomial de-aliasing. Mode 0: don’t do it. Mode 1: a filter is applied to the time-update \((J \ast U_t)\). The filter is formulated as a Galerkin projection of degree \(N\) to \(N_{\text{Under}}\), the effective resolution is thus \(N_{\text{Under}}\). Mode 2: in principle identical to Mode 1, but takes into account non-linear metric terms. For the linear mesh of this tutorial the result is identical, while Mode 2 is slightly more computational expensive, so we omit it.

For FLEXI we can run under-resolved computations without sub grid scale model. The only artificial dissipation is then provided by the Riemann solver used for the inter-cell fluxes. You can change the Riemann solver to see the effect with the following parameters:

To add Smagorinsky’s model set the following parameter to 1, here CS is the Smagorinsky constant usually chosen around 0.1 for isotropic turbulence (such as TGV).

6.5.3.2 Running the Simulation and Results

The command

```
flexi parameter_flexi.ini > std.out
```

runs the code and dumps all output into the file \(\text{std.out}\). If you wish to run the code in parallel using MPI, the standard command is
mpirun -np XX flexi parameter_flexi.ini > std.out

where $XX$ is an integer denoting the number of processes to be used in parallel. Note that FLEXI uses an element-based parallelization strategy, so the minimum load per process/core is one grid element, i.e. do not use more cores than cells in the grid! FLEXI writes a TGV analyze file (PROJECTNAME_TGVAnalysis.csv). You can use your favorite plotting program to visualize the ASCII data. Using gnuplot you can create plots with the following syntax:

```bash
gnuplot
set key autotitle columnhead
set datafile separator "",""
plot "PROJECTNAME_TGVAnalysis.csv" u 1:2 w l
```

Where you replace PROJECTNAME with the projectname you defined in the parameter_flexi.ini file.

### 6.5.3.3 Part I: crashing simulation

First we run FLEXI without overintegration/de-aliasing. We will find that the code crashes, once scale production becomes relevant. You can compare your result to the plot in the tutorial folder.

![Crashing simulation](image)

### 6.5.3.4 Part II: Overintegration

We now use overintegration by changing the respective settings in the parameter_flexi.ini file as described above. For Overintegration==1 set $N = 11$ and $NUnder = 7$. You can compare
your result to the plot below.

6.5.3.5 Part III: Explicit LES model

To see the effect of adding explicit eddy viscosity we activate the LES model (Smagorinsky) as described above. To obtain the reference result of the following plot set $CS = 0.1$. Don’t forget to switch overintegration of again and set polynomial degree to $N = 7$. Feel free to play around with the constant, have fun!
6.6 SOD Shock tube

The Sod shock tube example (Sod 1978) is one of the most basic test cases to investigate the shock capturing capabilities of a CFD code. A initial discontinuity is located in the middle of the one dimensional domain \([0, 1]\). The left and right states are given by \(\rho = 1, v = 0, p = 1\) and \(\rho = 0.125, v = 0, p = 0.1\). These states are already set as RefState in the the `parameter_flexi.ini` file.

Copy the sod tutorial folder

```
cp -r $FLEXI_TUTORIALS/sod .
```

6.6.1 Mesh Generation with HOPR

The mesh files used by FLEXI are created by supplying an input file `parameter_hopr.ini` with the appropriate information.

```
hopr parameter_hopr.ini
```

This creates the mesh file `SOD_100_mesh.h5` in HDF5 format.

6.6.2 Flow Simulation with FLEXI

This example requires the Finite Volume shock capturing and the Euler equations. Therefore set the following options in cmake:

```
FLEXI_FV ON
FLEXI_PARABOLIC OFF
```

and recompile the FLEXI code. The simulation setup is defined in `parameter_flexi.ini` and includes options for the Finite Volume shock capturing.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndicatorType</td>
<td>Persson</td>
<td>first conservative (density) used for indicator evaluation</td>
</tr>
<tr>
<td>IndVar</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IndStartTime</td>
<td>0.001</td>
<td>until this time FV is used in the whole domain</td>
</tr>
<tr>
<td>FV_LimiterType</td>
<td>MinMod</td>
<td></td>
</tr>
<tr>
<td>FV_IndUpperThreshold</td>
<td>-3.</td>
<td>upper threshold (if IndValue above this value, switch to FV)</td>
</tr>
<tr>
<td>FV_IndLowerThreshold</td>
<td>-4.</td>
<td>lower threshold (if IndValue below this value, switch to DG)</td>
</tr>
</tbody>
</table>
Explanation of the Finite Volume specific options:

- **IndicatorType**: specifies the indicator function that is used to detect DG elements that contain a discontinuity. The Persson indicator (Persson and Peraire 2006) is an element local indicator, which compares the different modes of the DG polynomial. If the amount of solution in the highest mode compared to the amount in the lower modes is high the DG polynomial may oscillate. All indicator functions return a high value for trouble elements and a low value for smooth elements.

- **IndVar**: Variable that is used to evaluate the indicator function. Here the density is used. In general the pressure \((6)\) is a good choice.

- **IndStartTime**: Until this time the actual indicator function is overwritten by a very high value to force the use of FV elements in the hole domain. This is necessary if initial discontinuities (like in this example) are placed perfectly at the element boundaries. In this case an element local indicator (like Persson) can not detect the discontinuity.

- **FV_LimiterType**: Limiter of the second order reconstruction.

- **FV_IndUpperThreshold** and **FV_IndLowerThreshold**: These two threshold values are used to decide in which elements the DG method and where the FV sub-cell scheme should be used. In general a single threshold could be sufficient. If the indicator value raises above this threshold the element used the FV scheme, and if the value is below the DG scheme is used. Practice shows that this could lead to an ongoing switching between the two schemes. Therefore a DG element is switched only to FV if the indicator value gets greater than the upper threshold, but it does not switch back to DG immediately if the value falls below this threshold. This only happens if the indicator value falls below the lower threshold.

The command

```
flexi parameter_flexi.ini
```

runs the code and generates 5 state files `sod_State_TIMESTAMP.h5` for \(t = 0.0, 0.05, 0.10, 0.15, 0.20\). To visualize the solution, the State-files must be converted into a format suitable for ParaView. Execute the command

```
posti_visu parameter_postiVisu.ini parameter_flexi.ini sod_State_0000000.*
```

to generate the corresponding vtu- and vtm-files, which can then be loaded into ParaView. There are two types of vtu-files, which contain either the DG or the FV part of the solution. The vtm-files combine the DG and FV vtu-file of every timestamp. Load the vtm-files into ParaView.

Since this example is a 1D test case use the Plot Over Line-Filter in ParaView and select the X Axis as line. The result should look like in figure 6.23.
6.7 Double Mach Reflection

The Double Mach Reflection is a classical test case to investigate the abilities of a numerical scheme to represent shock and contact discontinuities. It was invented by Woodward and Colella (Woodward and Colella 1984). A Mach 10 oblique shock wave hits a reflecting wall and the initial conditions are given by the Rankine-Hugoniot conditions

\[
(\rho, v_1, v_2, p) = \begin{cases} 
(8.0, 8.25 \cdot \cos(30^\circ), -8.25 \cdot \sin(30^\circ), 116.5) & x < x_0 + \sqrt{\frac{1}{3} y} \\
(1.4, 0.0, 0.0, 1.0) & x \geq x_0 + \sqrt{\frac{1}{3} y}, 
\end{cases}
\]

where \( x_0 = \frac{1}{16} \) is the start of the wall and the computational domain is \( \Omega = [0, 4] \times [0, 1] \), which is discretized by an equidistant cartesian mesh.

Copy the dmr tutorial folder

```
cp -r $FLEXI_TUTORIALS/dmr .
```

6.7.1 Mesh Generation with HOPR

The mesh files used by FLEXI are created by supplying an input file `parameter_hopr.ini` with the appropriate information.

```
hopr parameter_hopr.ini
```

Figure 6.23: Solution of the Sod shock tube example at \( t = 0.2 \).
This creates the mesh file \textit{DMR\_mesh.h5} in HDF5 format.

### 6.7.2 Flow Simulation with FLEXI

This example requires the Finite Volume shock capturing. Therefore turn the option FLEXI\_FV in the cmake configuration on. Additionally you should switch FLEXI\_PARABOLIC off and recompile the FLEXI code. The simulation setup is defined in \textit{parameter\_flexi.ini} and includes options for the Finite Volume shock capturing.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndicatorType</td>
<td>Jameson</td>
<td>Pressure is used for indicator evaluation</td>
</tr>
<tr>
<td>IndVar</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>FV_LimiterType</td>
<td>MinMod</td>
<td></td>
</tr>
<tr>
<td>FV_IndUpperThreshold</td>
<td>0.010</td>
<td>upper threshold (if IndValue above this value, switch to FV)</td>
</tr>
<tr>
<td>FV_IndLowerThreshold</td>
<td>0.005</td>
<td>lower threshold (if IndValue below this value, switch to DG)</td>
</tr>
<tr>
<td>FV_toDG_indicator</td>
<td>T</td>
<td>enable an additional Persson indicator to check switch from FV to DG</td>
</tr>
<tr>
<td>FV_toDG_limit</td>
<td>-5.5</td>
<td>threshold for additional Persson indicator</td>
</tr>
<tr>
<td>FV_IniSupersample</td>
<td>T</td>
<td>supersample initial solution for every FV sub-cell, since DG polynomial oscillates for initial solution</td>
</tr>
</tbody>
</table>

Explanation of the Finite Volume specific options (read also the explanations for the Sod shock tube in chapter 6.6.2):

- \textbf{IndicatorType}: The Jameson indicator is used to detect troubled cells, that should use the FV operator. This indicator is an adaption of the switching function of the Jameson-Schmidt-Turkel scheme (Jameson, Schmidt, and Turkel 1981) to Finite Volume sub-cells. In contrast to the Persson indicator this indicator is not element local and therefore more robust for travelling discontinuities.

- \textbf{FV\_toDG\_indicator}: if set to T (true) an additional Persson indicator for the switch from FV to DG is used. A FV element designated for the switch to DG is converted to DG and the Persson indicator is evaluated for this DG polynomial to test if the polynomial is oscillating. Only if this test is passed the element becomes a DG element. Otherwise it remains a FV element. This helps especially for all indicator functions acting on the FV sub-cells. Here the Jameson indicator only acts on the direct adjacent DOFs of a specific DOF, which does not allow to capture all high frequencies of a polynomial.

- \textbf{FV\_toDG\_limit}: Threshold for the additional Persson indicator. An element can only switch back to DG from FV if the indicator value on the DG representation is below this threshold.
• **FV IniSupersample**: If this option is F (false) the solution is initialized as DG polynomials for all elements. The indicator function is evaluated to find all troubled elements, which are than converted to FV elements. This causes major trouble if discontinuities lay inside DG elements which leads to heavy oscillating polynomials. Converting these oscillations to FV may produce invalid solutions (i.e. negative density). Switching this option on (T) enables a super sampling of the initial solution for every FV sub-cell, which removes the mentioned problems with oscillating polynomials. The mean value of every FV sub-cell is computed by evaluating the initial solution in \((N + 1)^3\) equidistant points inside the sub-cell and then taking the arithmetic mean value.

The command

```
flexi parameter_flexi.ini
```

runs the code and generates 11 state files `dmr_State_TIMESTAMP.h5` for \(t = 0.0, 0.02, \ldots, 0.20\). To visualize the solution, the State-files must be converted into a format suitable for ParaView. Execute the command

```
posti_visu parameter_postiVisu.ini parameter_flexi.ini dmr_State_0000000.0*
```

to generate the corresponding `vtu`- and `vtm`-files, which can then be loaded into ParaView. There are two types of `vtu`-files, which contain either the DG or the FV part of the solution. The `vtm`-files combine the DG and FV `vtu`-file of every timestamp. Load the `vtm`-files into ParaView.

The result at \(t = 0.2\) should look like in figure 6.24.

![Figure 6.24: Distribution of DG and FV elements (top) and density (bottom) of Double Mach Reflection at \(t = 0.2\).](image-url)
6.8 Linear Scalar Advection-Diffusion Equation

Besides the Navier-Stokes equations, FLEXI provides another equation system, the three-dimensional linear scalar advection-diffusion (LinAdvDiff for short) equation:

$$\frac{\partial \Phi}{\partial t} + \nabla \cdot (u \Phi) = d \nabla^2 \Phi,$$

where a scalar solution $\Phi$ is advected with the constant (three-dimensional) velocity $u$ and is subjected to diffusion with a constant scalar diffusion coefficient $d$.

The LinAdvDiff equation is useful to test and develop features in a simple and computationally cheap environment as well as to investigate basic properties of the DG operator, as we will do in this tutorial.

6.8.1 Theoretical Background

The dispersion and dissipation properties of the DGSEM operator can be analysed (Gassner and Kopriva 2011) by looking at the evolution of solutions to the one-dimensional linear scalar advection equation (without physical dissipation, $d = 0$) when a wave with a specific wavenumber is initialised. You should have a look in the paper if you are interested in the details of the analysis, here only a short summary is presented.

Under the mentioned simplifications, the equation to consider simply reads

$$\frac{\partial \Phi}{\partial t} + u \frac{\partial \Phi}{\partial x} = 0$$

and we consider a wave-like analytical solution on an infinite domain

$$\Phi(x, t) = e^{i(kx - \omega t)}$$

with the constant scalar transport velocity $u$, the angular frequency $\omega = ku$ and the wavenumber $k$.

We now assume to have a uniform mesh with a mesh size of $\Delta x$ and seek numerical solutions of the form

$$\Phi^l = \hat{\Phi} e^{i(kl\Delta x - \omega t)}$$

where $\Phi^l$ is a vector containing the degrees of freedom in cell $l$ and $\hat{\Phi}$ is a complex amplitude vector, both of size $N + 1$. If we look at the matrix notation of the semi-discrete system (meaning only spatial discretization) in a single element, we can formulate an algebraic eigenvalue problem.
6.8. LINEAR SCALAR ADVECTION-DIFFUSION EQUATION

\[ \Delta \hat{\Phi} = \Omega \hat{\Phi}, \]

with \( \Omega = \frac{\omega \Delta x}{a} \). The matrix \( \Delta \) contains the spatial discretization and depends on the non-dimensional wavenumber \( K = k \Delta x \).

By looking at the solutions of this eigenvalue problem, one can obtain relations for the dissipation and dispersion behavior inherent of the (spatial) numerical scheme for different wavenumbers. In figure 6.25 we plot these relationships depending on the polynomial degree \( N \) using DGSEM with Gauss nodes for the so called physical mode. This mode is associated with the eigenvalue that follows the exact dispersion relation for the largest range of wavenumbers and also has the biggest influence on the overall numerical solution, at least for rather well-resolved waves.

![Dispersion and dissipation relationship for N = 1 – 10 over the modified wavenumber K*](image)

Figure 6.25: Dispersion and dissipation relationship for \( N = 1 – 10 \) over the modified wavenumber \( K^* \). For the dispersion, the dashed line gives the exact relation.

The quantities are normalized by the number of grid points, \( K^* = \frac{K}{N+1} \) and \( \Omega^* = \frac{\Omega}{N+1} \) to give a fair comparison between different polynomial degrees. This means that if \( K^* = \pi \), we have two points per wavelength which is the theoretical minimum to resolve a wave given by the Nyquist-theorem while a normalized wavenumber of 0 indicates a constant solution.

As we can see, the dissipation properties of higher-order approximations are significantly improved compared to low-order schemes. They are able to preserve waves with higher frequencies without dissipating them. We also observe a sharp increase of the dissipation error in the high modes associated with higher-order approximations, which is one of the reasons the timestep is scaled down for larger \( N \).

We will now try to show these properties by conducting some numerical experiments using the linear scalar advection equation.
6.8.2 Compiling FLEXI with LinAdvDiff

If you want to use the LinAdvDiff equations, you need to specify the equation system during the configuration. We are going to create a second build-folder to keep any existing FLEXI binaries that might have been compiled using the Navier-Stokes equation system.

```bash
cd $FLEXIROOT
mkdir buildLinAdv && cd buildLinAdv
cmake -DFLEXI_EQNSYSNAME = linearscalaradvection ../
make
```

Since we don’t want to have diffusion in this tutorial we can either turn off the parabolic terms in general during configuration by using the following cmake command instead of the previous one:

```bash
cmake -DFLEXI_EQNSYSNAME = linearscalaradvection -DFLEXI_PARABOLIC = OFF ../
```

or simply specify a diffusion coefficient of 0 in the parameter file.

Of course you can also use ccmake ../ and set the parameters in the interface. The ones necessary are listed in the table below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLEXI_EQNSYSNAME</td>
<td>linearscalaradvection</td>
<td></td>
</tr>
<tr>
<td>FLEXI_PARABOLIC</td>
<td>OFF</td>
<td>optional</td>
</tr>
</tbody>
</table>

Note: The path to the flexi binaries is now different than before. We therefore type it out in all commands in this tutorial. You can, of course, set a different alias for the path to this executable in your ~/.bashrc.

6.8.3 Setup

We want to run numerical experiments and solve the one-dimensional linear scalar advection equation. As our initialization condition we choose a single wave with a specific angular frequency and observe the behaviour of this wave depending on the normalised non-dimensional wavenumber $K^*$ and the polynomial degree $N$. Since the analysis in the previous section has been done on an infinite domain, we are going to make our computational domain large enough to neglect influences from the boundaries. An alternative would be to use periodic boundary conditions, but this would limit our choice of wavelengths since they have to fit in the domain and could also negatively influence the stability of the time discretization. When we conduct the experiments, we are going to focus our attention to a single element in the middle of the domain and observe the evolution of the wave in there.
6.8. LINEAR SCALAR ADVECTION-DIFFUSION EQUATION

6.8.3.1 Mesh

For this tutorial we are going to create a (quasi-)one-dimensional equidistant grid. We choose $\Delta x = 2$ which is equal to the reference element. We are going to make the computational domain rather long, so the boundary conditions will have no influence on our solution. Recall that for LinAdv information propagates with velocity $u$, so if our boundaries are a distance of $L$ away from the cell we are looking at, we can compute solutions up to $t = uL$ without having any influence of the boundary conditions.

In our case we are creating a cartesian grid with $x \in [-61, 61]$ and 61 cells to reach our desired $\Delta x$. We will later focus on the single element in $x \in [-1, 1]$. To achieve a (quasi-)one-dimensional simulation, we choose periodic boundary conditions in $y$– and $z$–direction. The boundary conditions in the $x$–direction will not matter, so we simply use Dirichlet-type BCs (BC type 2) with the analytical wave function as the boundary state. We specify this by setting the BC state to zero, which means that the initialization function will be used instead of a seperate function.

We provide a parameter file for HOPR to generate this mesh. It is located in the subdirectory mesh within the directory that belongs to this tutorial. Either run hopr with this parameter file

```
cd $FLEXIROOT/tutorials/linadv/mesh
hopr parameter_hopr.ini
```

or use the mesh file CART_1D_mesh.h5 which is provided in this directory.

6.8.3.2 FLEXI parameters

A parameter_flexi.ini is provided in the directory $FLEXIROOT/tutorials/linadv/. We are going to discuss the parameters that are specific to the LinAdvDiff equation system and the ones needed for this tutorial. In the section EQUATION in the parameter file you will find the options that directly influence the behaviour of the equation:

```
! ============================================================ !
! EQUATION !
! ============================================================ !
AdvVel = (/1.,0.,0./)
DiffC = 0.
```

The parameter AdvVel sets the advection velocity in all three space directions. Since we are simulating a one-dimensional case, we only set the first entry to something other than zero. Here we choose one, although the special initialization that we will be using will set the advection velocity to this value no matter what we put in here. Since we don’t want any physical diffusion, we set the diffusion coefficient DiffC to zero. If you did compile without the FLEXI_PARABOLIC option set to ON, you don’t need to set this.
We also need to set the initial condition and we choose a special one here to match the initial conditions of the analysis we want to compare against. Recall that the exact solution was

\[ \Phi(x, t) = e^{i(kx - \omega t)}, \]

where the amplitude of the wave is given by the real part of this complex expression. This real part is then given by

\[ A(x, t) = \cos(kx - \omega t). \]

This function is implemented as ExactFunc 6 in the LinAdvDiff equation system, so we are specifying this in the parameter file:

```
IniExactFunc = 6
```

What is left is to specify the angular frequency \( \omega \), since the wavenumber is given by \( k = \frac{\omega}{u} \).

The angular frequency can be set during runtime using the parameter

```
OmegaRef = 2.
```

We will adopt this parameter when we run our simulations.

To reduce the possibility that the error introduced by the time discretization will alter our results (we are focused here on the behaviour of the spatial discretization), we set our CFL number to a rather small value:

```
CFLscale = 0.1
```

Since this is a really small and fast computation, we are also going to use visualization routines during runtime, so we don’t need to convert the state files using the posti_visu tool. To do this, we set the following parameters in our parameter file:

```
outputFormat = 3
...
NVisu = 30
```

which will enable the .vtu output. Whenever the analyze routines are called, there will also be a call to the visualization routines. Since we did not set a Analyze_dt in the parameter file, the analyze routines will only be called at the beginning and the end of the simulation. The parameter NVisu sets the polynomial degree of the output basis.

**Remark:** There is also a parameter that allows us to only call the visualize routines every \( n \)-th call of the analyze functions. This parameter is called nWriteData. So if you set

```
nWriteData = 10
```

in the parameter file, the visualization file will only be written every 10th analyze routine call. The default for this value is simply 1.
6.8.4 Simulation and results

We are going to start with the simulation of a well-resolved wave and a moderate polynomial degree of $N = 4$ which is already set in the parameter file. For a modified wavenumber of $K^* = \frac{1}{4}\pi$, the dissipation and dispersion relations tell us that we can expect very little errors. To get the desired modified wavenumber, we need to calculate the corresponding angular frequency. Using the relations given above, the angular frequency can be computed as

$$\omega = \frac{K^*(N + 1)}{\Delta x} u.$$

This means we need to set the following frequency in the parameter file:

| OmegaRef | 1.96349540849 |

All other options are already set correctly. The simulation will be performed up to a time of $t = 5$. We can now run the simulation using

```
cd $FLEXIROOT/tutorials/linadv
$FLEXIROOT/buildLinAdv/bin/flexi parameter_flexi.ini
```

Even on a single processor this calculation should only take a few seconds.

Now we can have a look at the result by opening the file `LinAdvCosineWave_Solution_0000005_000000000.vtu` with ParaView. We are only interested in the central element in $x \in [-1, 1]$, so we can clip everything else. Also since we are performing a one-dimensional simulation, we can extract the solution along the $x$-axis and create a simple line-plot to look at the results. To see how dispersion and dissipation introduced by the numerics influence our results, we will also plot the analytical solution to the wave transportation problem, given by the equation above. The result can be seen in figure 6.26. As was expected, only very little deviation can be found from the analytical solution, since we are considering a very well resolved wave with a low value for $K^*$.

Of course it is much more interesting to look at the behaviour of waves that are not well resolved and how the polynomial degree influences the behaviour of these waves. To do this, we are going to perform four simulations with polynomial degrees ranging from 2 to 11 and set the normalised wavenumber to $K^* = 1.6$. Table 6.17 will give you an overview of the polynomial degrees and the angular frequencies needed to achieve a constant $K^*$.

<table>
<thead>
<tr>
<th>N</th>
<th>OmegaRef</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.4</td>
</tr>
<tr>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>6</td>
<td>5.6</td>
</tr>
<tr>
<td>11</td>
<td>9.6</td>
</tr>
</tbody>
</table>

Table 6.17: Angular frequencies needed to get $K^* = 1.6$. 

R2019-06-06 100
Figure 6.26: Result for $N = 4$ and a well resolved wave with $K^* = \frac{1}{4} \pi$ at $t = 5$. Comparison against exact wave transport.

Now run these four simulations by adjusting the values for $N$ and $\omega_{\text{Ref}}$ in the parameter file before each run. Remember to give each of them a unique ProjectName or else the results will be overwritten each time. Also make sure that $N_{\text{Visu}}$ is set to at least three times the value of $N$ to get a meaningful visualization.

We now compare the results as they can be found in the next figure. Again we plot the solution at $t = 5$ for the considered polynomial degrees against the analytical solution. Since we compare using a constant normalized wavenumber, the actual frequency of the wave is increased with the polynomial degree. If we take a look at the result for $N = 2$ (which should not be considered high-order), we observe both a significant amount of dissipation (a drop in amplitude by about 85%) as well as severe dispersion, e.g. a change in angular frequency and phase angle.
For a polynomial degree of $N = 4$ (at the lower end of what can be considered high-order), we already observe some improvements. The amplitude only drops by around 65%, although the phase shift is still clearly visible. This trend continues with higher polynomial degrees. For the highest value of $N$ tested here, we observe only small changes in phase angle or amplitude.

These results show how higher-order schemes are able to capture waves with fewer points per wavelength than the lower-order approximations. This is one of the central aspects why we use such schemes.

Figure: Result for a under-resolved wave with $K^* = 1.6$ at $t = 5$ for $N = 2$ (top left), $N = 4$ (top right), $N = 6$ (bottom left) and $N = 11$ (bottom right).
6.9 Plane Turbulent Channel Flow

This tutorial describes how to set up and run the Plane-Turbulent-Channel-Flow test case. We will learn how to use the split form DG method to guarantee non-linear stability of the turbulent channel flow. In a second step, we add the sub grid scale model of Smagorinsky combined with Van Driest type damping to run stable wall-bounded turbulent flows with explicit small scale dissipation. The tutorial assumes that you are familiar with the general FLEXI and HOPR work flow (please finish the previous tutorials first if this sounds strange to you).

6.9.1 Flow description

The flow is calculated in a plane channel with half-height $\delta = 1$, streamwise ($x$ coordinate) length $2\pi$ and span ($z$ coordinate) width $\pi$ with periodic boundaries in the $x$- and $z$-directions as well as no-slip walls at the top and the bottom of the domain. As initial conditions an analytical mean turbulent velocity profile is used superimposed with sinus perturbations in the $u$, $v$ and $w$ velocity components and a constant density of $\rho = 1$. The superimposed perturbations lead to rapid production of turbulent flow structures. Since the wall friction would slow down the flow over time, a constant pressure source term imposing a pressure gradient $\frac{dp}{dx} = -1$ is added as a volume source. While the test case is incompressible in principle, we solve it here in a compressible setting. The chosen Mach number with respect to the bulk velocity in the field is $Ma = 0.1$ according to the Moser channel test case. In this setting, the wall friction velocity $\tau$ will always be equal to 1. We can then define a Reynolds number based on the channel half-height and the wall friction velocity as $Re_\tau = 1/\nu$.

6.9.2 Compiler options

Make sure that FLEXI is compiled with the CMake options listed in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAKE_BUILD_TYPE</td>
<td>Release</td>
<td></td>
</tr>
<tr>
<td>FLEXI_EQYNYSYSNAME</td>
<td>navierstokes</td>
<td></td>
</tr>
<tr>
<td>FLEXI_PARABOLIC</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>FLEXI_MPI</td>
<td>ON</td>
<td>optional</td>
</tr>
<tr>
<td>FLEXI_EDDYVISCOSITY</td>
<td>ON</td>
<td>optional</td>
</tr>
<tr>
<td>FLEXI_NODETYPE</td>
<td>GAUSS-LOBATTO</td>
<td></td>
</tr>
<tr>
<td>FLEXI_SPLIT_DG</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>FLEXI_TESTCASE</td>
<td>channel</td>
<td></td>
</tr>
<tr>
<td>FLEXI_BUILDPOSTI</td>
<td>ON</td>
<td></td>
</tr>
</tbody>
</table>

R2019-06-06 103
6.9. PLANE TURBULENT CHANNEL FLOW

### POSTI BUILD CHANNEL FFT ON

For all other CMake options you may keep the default values. Compile the code.

#### 6.9.2.1 Mesh Generation with HOPR

We use a cartesian mesh with 4 cells per direction for the tutorial. The mesh is stretched in the wall-normal direction to accommodate for the straining of the vortexes close to the wall. In case you want to generate other meshes the parameter file for HOPR is included in the tutorial directory (parameter_hopr.ini), the default mesh is included. Using 4 cells with a polynomial degree of $N = 5$, means we use a large eddy simulation setup of 24 DOFs per direction.

#### 6.9.3 Tutorial - Flow at $Re_\tau = 180$

Copy the `plane_turbulent_channel_flow` tutorial folder to your working directory.

```bash
cp -r $FLEXI_TUTORIALS /plane_turbulent_channel_flow .
```

Step into the folder. In case you do not want to generate the mesh files yourself, a default mesh has already been provided.

#### 6.9.3.1 Preparing the Flow Simulation with FLEXI

The simulation setup is already defined in `parameter_flexi.ini`.

##### 6.9.3.1.1 Output

In this tutorial we don't look at the flow visualization of the instantaneous state files. Here, we will rather post process consecutive, instantaneous state files with the `posti_channel_fft` tool. As an output, we receive mean velocity and Reynolds stress profiles as well as turbulent energy spectra at different locations normal to the channel wall.

##### 6.9.3.1.2 Interpolation / Discretization parameters

In this tutorial we use the split form DG method to guarantee non-linear stability of the turbulent channel flow simulation. As already specified in the CMake options table 6.18, the `FLEXI_SPLIT_DG` option has to be switched ON in combination with the `FLEXI_NODETYPE GAUSS-LOBATTO`. FLEXI has several split flux formulations implemented. Therefore, a specific
split flux formulation has to be set in the `parameter_flexi.ini` file. In this tutorial the pre-defined split flux formulation by Pirozzoli is used, which results in a kinetic energy preserving DG scheme.

```
! ============================================================== !
! SplitDG                                                       
! ============================================================== !
SplitDG = PI ! SplitDG formulation to be used: SD, MO, 
             DU, KG, PI
```

To switch on Smagorinsky’s model set the `eddyViscType` to 1 in the `parameter_flexi.ini` file. In addition, the following parameters have to be set. CS is the Smagorinsky constant usually chosen around 0.11 for wall bounded turbulent flows and the turbulent Prandtl number is commonly set to 0.6. To ensure the correct behaviour of the eddy viscosity provided by Smagorinsky’s model when approaching a wall, Van Driest type damping has to be switched on.

```
! ============================================================== !
! LES MODEL                                                    
! ============================================================== !
eddyViscType = 0 ! Choose LES model, 1:Smagorinsky              
VanDriest = T ! Van Driest damping for LES viscosity (   
              channel flow only)                           
CS = 0.11 ! Smagorinsky constant                             
PrSGS = 0.6 ! turbulent Prandtl number                       
```

### 6.9.3.2 Running the Simulation and Results

Now run the simulation, either using

```
flexi parameter_flexi.ini
```

or

```
mpirun -np XX flexi parameter_flexi.ini
```

when you want to use more than one processor. Once the simulation finished state files can be post processed by the `posti_channel_fft` tool which was build by the `POSTI_BUILD_CHANNEL_FFT` CMake option. To run the postprocessing, the standard command is

```
posti_channel_fft parameter_channel_fft.ini [State1 State2 ...]
```

where the `parameter_channel_fft.ini` file is given in the tutorial folder and the amount of statefiles is specified by the user. In this tutorial we use all state files with a timestamp between $t = 10.0$ and $t = 15.0$. As an output you receive three different files. One containing the mean velocity profiles as well as the Reynolds stress profiles and the other two files contain turbulent
energy spectra. To visualize those files you can run the python script plotChannelFFT.py in the tools/testcases folder with the following command in your simulation directory

```
python $FLEXIROOT/tools/testcases/plotChannelFFT.py -p $PROJECTNAME -t $POSTITIME
```

where $PROJECTNAME specifies the project name specified in the parameter_flexi.ini file and $POSTITIME the timestamp of your output files from the posti_channel_fft tool.

### 6.9.3.3 Part I: SplitDG iLES

First, we run FLEXI without Smagorinsky’s model which we call an implicit LES (iLES), as no explicit sub-grid scale dissipation model is added. The resulting mean velocity and Reynolds stress profiles as well as turbulent energy spectra close to the centre of the channel are given in Figure 6.27.

![Figure 6.27: Mean velocity and Reynolds stress profiles (left) as well as turbulent energy spectra close to the centre of the channel (right) of an implicit LES at $Re_{\tau} = 180$.](image)

### 6.9.3.4 Part II: SplitDG with explicit LES model

In a second step, we run FLEXI with Smagorinsky’s model and Van Driest damping which needs to be switched on in the parameter file as described above. The resulting mean velocity and Reynolds stress profiles as well as turbulent energy spectra close to the centre of the channel are given in Figure 6.28. In comparison to the previous simulation you might recognize the effect of the explicit damping on the Reynolds stress profile $\overline{u'v'}$ close to the maximum, most. To further study the influence of Smagorinsky’s model play around with the spatial resolution both in terms of grid resolution as well as the polynomial degree N. You can also increase the Reynolds number to $Re_{\tau} = 395$ or $Re_{\tau} = 590$ and compare the results to DNS results from Moser et al. (Moser, Kim, and Mansour 1999).
6.10 Flow around a cylinder

In this tutorial, the simulation around a two-dimensional circular cylinder at $Re_D = 200$ and $Ma = 0.2$ is considered. The goal of this tutorial is to introduce the usage of a new functionality of the `posti_visualizerecordpoints` tool and the new tool `posti_dmd`.

6.10.1 Compiler options

Make sure that FLEXI is compiled with the CMake options listed in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAKE_BUILD_TYPE</td>
<td>Release</td>
<td></td>
</tr>
<tr>
<td>FLEXI_2D</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>FLEXI_EQYNSYSNAME</td>
<td>navierstokes</td>
<td></td>
</tr>
<tr>
<td>FLEXI_PARABOLIC</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>FLEXI_MPI</td>
<td>ON</td>
<td>optional</td>
</tr>
<tr>
<td>POSTI_BUILD_VISUALIZERECORDPOINTS</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>POSTI_DMD</td>
<td>ON</td>
<td></td>
</tr>
</tbody>
</table>

To check whether they are set, change to your build folder and open the CMake GUI

```
ccmake [flexi root directory]
```

If necessary, set the above options and then compile the code by issuing
6.10.2 Mesh Generation with HOPR

The mesh file used by FLEXI is created by HOPR.

```
./hopr parameter_hopr.ini
```

This creates the mesh file Cylinder_Re200_mesh.h5 in HDF5 format. If HOPR is not available, the mesh file is supplied in this tutorial.

6.10.3 Flow Simulation with FLEXI

The simulation setup is defined in parameter_flexi.ini. The initial condition is selected via the variable vector \( \text{RefState}=\langle 1., 1.0, 0., 0., 17.857 \rangle \) which represents the vector of primitive solution variables \((\rho, u, v, w, p)^T\).

Material properties are given in table 6.20. Based on the ideal gas law, we get

\[
Ma = \frac{1}{\sqrt{\kappa p/\rho}} = 0.2
\]

Note that in this non-dimensional setup the mesh is scaled such that the reference length is unity, i.e. \( D = 1 \). Then to arrive at \( Re = \rho u D/\mu = 200 \), the viscosity is set to

\[
\mu = \rho u D/Re = 1/Re = 0.005
\]

<table>
<thead>
<tr>
<th>Property</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic viscosity ( \mu )</td>
<td>mu0</td>
<td>0.005</td>
</tr>
<tr>
<td>ideal gas constant ( R )</td>
<td>R</td>
<td>17.857</td>
</tr>
<tr>
<td>Prandtl number</td>
<td>Pr</td>
<td>0.72</td>
</tr>
<tr>
<td>isentropic coefficient ( \kappa )</td>
<td>kappa</td>
<td>1.4</td>
</tr>
</tbody>
</table>

6.10.4 Numerical settings

The DG solution on the mesh is represented by piecewise polynomials and the polynomial degree in this tutorial is chosen as \( N = 4 \).

The main code settings are shown in table 6.21.
6.10. FLOW AROUND A CYLINDER

### Table 6.21: Numerical settings

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Polynomial degree</td>
<td>4</td>
</tr>
<tr>
<td>MeshFile</td>
<td>Mesh file to be used</td>
<td>Cylinder_Re200_mesh.h5</td>
</tr>
<tr>
<td>tend</td>
<td>end time of the simulation</td>
<td>300</td>
</tr>
<tr>
<td>Analyze_dt</td>
<td>time interval for analysis</td>
<td>0.01</td>
</tr>
<tr>
<td>nWriteData</td>
<td>dump solution every n’th Analyze_dt</td>
<td>500</td>
</tr>
<tr>
<td>CFLscale</td>
<td></td>
<td>0.9</td>
</tr>
<tr>
<td>DFLscale</td>
<td></td>
<td>0.9</td>
</tr>
</tbody>
</table>

#### 6.10.5 Boundary conditions

The boundary conditions were already set in the mesh file by HOPR. Thus, the simulation runs without specifying the boundary conditions in the FLEXI parameter file. The freestream boundaries of the mesh are Dirichlet boundaries using the same state as the initialization, the wall is modeled as an adiabatic wall. The boundary conditions in $z$ direction are not relevant for this 2D example, but would be realized as periodic boundaries for a 3D simulation. All boundary conditions used are listed below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>State</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_cylinder</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BC_farfield</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 6.10.6 Running the code

We proceed by running the code in parallel. For example, using 4 processors, use the following command

```
mpirun -np 4 flexi parameter_flexi.ini
```

The simulation runs for 300 convective time units to achieve periodic vortex shedding, thus the simulation can take up to one to two hours.

#### 6.10.7 Evaluation of Strouhal number

The Strouhal number (which is a non-dimensional frequency, $Sr = \frac{f \cdot D}{U}$, describing the oscillatory motion of the flow) is estimated using the forces acting on the cylinder induced by the vortex shedding. The forces are calculated on the fly during runtime. The associated flags in the parameter file are

```
CalcBodyForces=T
```

R2019-06-06
6.10. FLOW AROUND A CYLINDER

The first line activates the calculation of the forces at each \texttt{Analyze\_dt}, the second line enforces output of the forces to a file. In figure 6.29 the force in y-direction is plotted. By measuring the time from peak to peak over several periods the Strouhal number can be estimated to 0.1959 which is close to the expected value from literature.

![Figure 6.29: Resulting forces on the airfoil up to $t = 10$.](image)

6.10.8 Evaluation of the separation angle

The mean separation angle is evaluated using the \texttt{record points}-tool as introduced in 6.3.13. The simulation setup already contains the \texttt{record points} set and the record points are written during the simulation. The \texttt{record points} set contains a plane within the boundary layer of the upper cylinder side. This time we want to use the \texttt{Plane\_doBLProps} functionality within the \texttt{posti\_visualizerecordpoints} tool. With this tool we want to analyze the boundary layer properties such as the wall friction to estimate the separation point. The parameter needed are already set in the \texttt{parameter\_visualizeRecordpoints.ini} file.

You can run the tool using

\begin{verbatim}
posti_visualizerecordpoints parameter_visualizeRecordpoints.ini Cylinder_Re200_RP_ *
\end{verbatim}

After executing the tool, you will get a file named \texttt{Cylinder\_RP\_BLProps\_upperSide\_BLPla000001.vts} which can be visualized with ParaView. The Data we want to visualize is one dimensional, so you won’t be able to see the data in the render view. To visualize it you need the apply the plot over time filter. ParaView should automatically apply the correct range to plot on. By
plotting tau_w over the circumference you can estimate the separation angle to 113 deg (the intersection with the y = 0 line).

### 6.10.9 Dynamic Mode Decomposition

In this part of the tutorial we want to introduce the posti tool posti_dmd. The dynamic mode decomposition is an algorithm divide a temporal series into a set of modes which are associated with a frequency and grow/decay rate. With this tool we are also capable to determine the Strouhal frequency. The dynamic mode decomposition (DMD) is implemented according to Schmid et al. (Schmid, Meyer, and Pust 2009).

To use this tool, we need a higher temporal resolution of the written state files. Thus, we change the time tend to 310 and nWriteData to 1. We restart the simulation from the latest state file:

```
flexi parameter_flexi.ini Cylinder_Re200_State_0000300.000000000.h5
```

To execute the DMD on the density run the following command:

```
posti_dmd parameter_dmd.ini Cylinder_Re200_State_00003*
```

Depending on the available memory you might have to decrease the number of input state files. After execution you will see two additional files Cylinder_Re200_DMD_0000300.000000000.h5 and Cylinder_Re200_DMD_Spec_0000300.000000000.dat. The first file contains the field representation of the different modes and the second file contains the ritz spectrum of the modes.

To visualize the field run the following command:

```
posti_visu parameter_postivisu.ini Cylinder_Re200_DMD_0000300.000000000.h5
```

The new file Cylinder_Re200_Solution_0000300.000000000.vtu now contains five modes to visualize. Figure 6.30 shows the steady, the global, the first and the second harmonic mode. The global mode is the mode of the considered Strouhal number.

With the python script plot_RitzSpectrum.py the Ritz spectrum of the DMD can be plotted. The script is placed in the tools folder of FLEXI. To plot the spectrum execute:

```python
python plot_RitzSpectrum.py -d Cylinder_Re200_DMD_Spec_0000300.000000000.dat
```

The result is a Ritz spectrum as shown in fig 6.31. On the x-axis the frequency of the modes and on the y-axis the growth/decay factor is plotted, whereas modes with ω_r < 0 are damped. The modes placed directly on the x-axis are the already discussed modes, from left to right the global, the first, the second harmonic mode and so on. The color and size of the plotted
6.10. FLOW AROUND A CYLINDER

Figure 6.30: DMD modes of the density field. Top left steady mode, top right global mode, bottom left first harmonic, bottom right second harmonic.

modes represent the Euclidian norm of the mode which can be interpreted as an energy norm of the mode.

Figure 6.31: Ritz spectrum.
7 Unit tests

Unit tests are used to test individual key units of the source code. Currently these key routines include:

- Calculation of node positions and integration weights.
- Calculation of Vandermonde matrices.
- Calculation of derivative matrices.
- Algorithms to interpolate data from one set of nodes to another.
- Algorithm to prolong volume data to sides.
- Algorithm to perform a surface integral.
- Functionality of Read-In tools.

7.1 Integration of unit test with CTest

These unit tests are integrated into the FLEXI build process using the CTest tool. Usually CTest will be run every time you build FLEXI and give you an overview on the exit status of each test that looks something like this:

```
Run unit tests
Test project /home/FLEXI/build
  Start 1: NodesAndWeights
      1/6 Test #1: NodesAndWeights .............. Passed 0.01 sec
  Start 2: Vandermonde
      2/6 Test #2: Vandermonde .................. Passed 0.01 sec
  Start 3: DerivativeMatrix
      3/6 Test #3: DerivativeMatrix .............. Passed 0.00 sec
  Start 4: ChangeBasis
      4/6 Test #4: ChangeBasis .................. Passed 0.00 sec
  Start 5: SurfInt
      5/6 Test #5: SurfInt ....................... Passed 0.00 sec
  Start 6: ProlongToFace
      6/6 Test #6: ProlongToFace ............... Passed 0.00 sec

100% tests passed, 0 tests failed out of 6
Total Test time (real) = 0.05 sec
```

To manually run the tests after a build use the CTest command
7.2. IMPLEMENTATION OF UNIT TESTS

The manual page of CTest can give you an overview of all available options.

If you don't want to run the test after each build there is a CMake option called FLEXI_UNITTESTS that can be used to turn the tests on and off. This is an advanced option that CCMake will only show if you enter the advanced mode by pressing the t key.

7.2 Implementation of unit tests

All unit tests are implemented in FORTRAN and can be found in the subdirectory unitTests in your FLEXI directory alongside a separate CMakeLists.txt and some binary input and reference files.

7.2.1 CMakeLists.txt

The CMakeLists.txt defines a custom function called add_unit_test which can be used in the CMakeLists.txt to add a single test to the CTest tool. The syntax is

```
add_unit_test(NAME SOURCEFILE.F90)
```

All tests are defined using this function. At the end of the CMakeLists.txt a custom target all_tests is defined which includes all unit tests and will run the ctest command after it has been build.

The whole CMakeLists.txt content is included in the main CMakeLists.txt if the option FLEXI_UNITTESTS is set to ON (default) by CMake.

7.2.2 General unit test structure

The general structure of the unit tests is the same in all cases. They are implemented as FORTRAN programs. The unit test will call a function or subroutine from the FLEXI framework with input either set in the program itself or read from a binary file. The output of this call will then be compared to some precomputed reference results (also stored as binary files) with a certain tolerance to account for differences in e.g. compiler versions and system architecture. If the results are within the given tolerance, the test will be passed, otherwise it will fail by returning a value other than 0.

The programs usually also contain a command line option that can be used to generate the reference solution from a code version that is known to work correctly.
7.2. IMPLEMENTATION OF UNIT TESTS

Have a look at the source code of one of the already implemented unit tests if you want to have a more detailed idea about how to implement your own tests.

7.2.3 Generation of reference mesh data

Some of the unit tests require parts of the mesh data structure to be able to call the functions to be tested. For this purpose, a curved single element is created and all the mesh data stored as a binary file called UnittestElementData.bin. This binary file can then be read during runtime by the unit test programs.

To generate the curved single element mesh, run HOPR with the parameter file provided in the unitTest subdirectory of FLEXI. To generate the binary file, run FLEXI with the following command line argument and the parameter file provided in the unitTest subdirectory:

```
flexi --generateUnitestReferenceData parameter.ini
```
8 Installation guidelines

This chapter contains guidelines to install the code from Github on specific systems.

8.1 Cloning and compiling on machines at the HLRS

Unfortunately, the GitHub server is not available on machines at the HLRS, such as the Hazelhen, due to restricted internet access. The workaround is to use ssh tunnels to access the GitHub repositories. Note that the remote repositories hosted at the GitLab at the Institute of Aerodynamics and Gasdynamics (IAG), no ssh tunnel is required and cloning works straightforwardly.

The following instructions to access the GitHub repositories on HLRS machines is taken from the HLRS wickie page, see https://wickie.hlrs.de/platforms/index.php/Secure_Shell_ssh#Git.

8.1.1 HTTPS

Unfortunately, just using a SSH tunnel as with the SSH and git protocols is not sufficient in this case. Instead, one has to connect via an additional SOCKS proxy on a machine that has unlimited access to the internet, e.g. your local machine.

In order to do so, establish a proxy by using a special feature of OpenSSH:

```
ssh -N -D 1080 localhost
```

This will establish some kind of a "loopback" SSH connection from your local machine to itself which will not execute any command (-N) but act as an SOCKS proxy on port 1080 (-D 1080).

On a second shell, now login to the desired HWW-system and forward a port on the remote machine (e.g. 7777) to the port on your local machine where the newly established SOCKS proxy is listening on (1080):

```
ssh -R 7777:localhost:1080 <system-name>.hww.de
```

By doing so, you have a SOCKS proxy listening on port 7777 of the HWW-system. Hence you can use this proxy for accessing remote git repositories. Unfortunately, the default versions of
8.1. CLONING AND COMPILING ON MACHINES AT HLRS

Installation guidelines

git installed on the HWW-systems are not capable of doing this. You hence have to load an appropriate version first:

```
module load tools/git
```

In order to use the proxy, you can now add "-c https.proxy='socks5://localhost:7777'" to your `git` commands, e.g.:

```
```

In order to avoid typing this in every `git` call, you can also set the respective port to be used whenever `git` talks to a remote repository via HTTPS by

```
git config --global https.proxy 'socks5://localhost:7777'
```

Unfortunately, to connect with GitHub for pulling or pushing, the connection to Hazelhen has to be done via the ssh tunnel.
References


8.1. CLONING AND COMPILING ON MACHINES AT HLRS

Chapter 8. Installation guidelines

387–411.


