FLEXI Documentation

Release 24.10

Institute for Aerodynamics and Gas Dynamics (University of Stuttgart)

Jun 22, 2025

USER GUIDE

FLEXI is a high-order numerical framework for solving PDEs, with a special focus on Computational Fluid Dynamics. FLEXI is based on the Discontinuous Galerkin Spectral Element Method (DGSEM), which allows for high-order of accuracy and fully unstructured hexahedral meshes. The solver is parallelized very efficiently for large-scale applications and scales to 500,000+ cores. Moreover, FLEXI comes with a capable pre- and post-processing suite that enables complex simulation setups up to the finished visualization.

FLEXI has been developed by the Numerics Research Group (NRG) founded by Prof. Claus-Dieter Munz and currently lead by Prof. Andrea Beck at the Institute of Aerodynamics and Gas Dynamics at the University of Stuttgart, Germany.

You can find detailed installation instructions, the extensive documentation and several tutorial cases for FLEXI here.

Numerous people have worked on and with FLEXI over the last years. We would like to thank all these contributors for their efforts they spent on building FLEXI.

In case you have questions regarding FLEXI or want to contribute yourself by either reporting bugs, requesting features or adding something different to the project, feel free to open an issue or pull request.

CHAPTER

ONE

QUICK START GUIDE

This quick start guide allows for a fast installation and setup of **FLEXI** without diving into the general details of the framework, compile options and features. Further information and detailed descriptions are available by following the indicated references.

1.1 Installation and Setup

FLEXI is free and open source (GPLv3). The current version of the code-framework is available online and can be acquired from the GitHub repository by either cloning it or downloading the compressed folder, see *Obtaining the Source Code*.

FLEXI requires the following *packages* to be installed on the system:

- git
- CMake
- Fortran and C/C++ compilers (GNU compilers recommended)
- MPI libraries (OpenMPI recommended)
- LAPACK/OpenBLAS
- HDF5
- FFTW

LAPACK/OpenBLAS, HDF5, and FFTW can be automatically installed by enabling the corresponding compiler options in the CMake configuration.

FLEXI is compiled using CMake. For a compilation in default configuration use:

```
mkdir build
cmake -B build
cmake --build build
```

Custom configurations can be generated with

```
ccmake -B build
```

including the installation of the third-party packages mentioned above (LAPACK/OpenBLAS, HDF5, FFTW), see section *compiler options* for a detailed list. The executables will be generated in `./build/bin/`.

1.2 Mesh Generation

For the generation of high-order meshes the standalone mesh generator **HOPR** is required, creating **FLEXI** compatible mesh files in HDF5 format. Simple, structured meshes can be directly generated in **HOPR** using the integrated mesh generator, while the processing of complex geometries can be a based on external meshes in CGNS or GMSH format. In any case, a parameter file for the mesh generation and modification is required. **HOPR** is available on GitHub and can be compiled using CMake or by simply downloading the provided AppImage. For an in-depth description we refer to the **HOPR** documentation.

1.3 Running FLEXI

FLEXI can be run by executing the generated binary in the build folder `$FLEXIROOT/build/bin/flexi` and providing a parameter file with the simulation-specific definitions. The *feature list* provides an overview of the various features implemented in **FLEXI**, while section *parameter file* contains all options and a short description. For the definition of the initial and boundary conditions we refer to sections *Initial Conditions* and *Boundary Conditions*, respectively. Optionally, simulations can be restarted from an existing state file (i.e. volume solution) by appending it to the argument vector:

```
flexi parameter_flexi.ini [Restart_State.h5]
```

Further details concerning the capabilities of **FLEXI** and the application to small testcases, including e.g., the flow around a *NACA0012* airfoil, are included in the *tutorials*.

1.4 Tools

FLEXI comes with a comprehensive *postprocessing* toolchain, such as, e.g., the *interpolation* between different meshes, the *time averaging* of solution files, and the *animation*. Most importantly, it includes the `posti_visu` tool to convert the solution files from the custom *h5* format to *vtu* files readable by **ParaView**, as covered in the *workflow section*.

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

Generally, **FLEXI** requires the following packages:

- git
- CMake
- Fortran and C/C++ compilers (GNU compilers recommended)
- MPI libraries (OpenMPI recommended)
- ParaView
- LAPACK/OpenBLAS[1]
- HDF5[1]
- FFTW[1]

2.1.1 Installing the Dependencies from the Package Repositories

FLEXI has been tested on various Linux distributions, including Ubuntu and Debian, as well as OpenSUSE, CentOS and Fedora. The required packages for DEB-based and RPM-based Linux distributions can be obtained from the `apt` and the `dnf` environment, respectively. Refer to Table 2.1 for the package names.

Table 2.1: Package names for Linux distributions

Package Installation Command	Debian / Ubuntu `sudo apt-get install`	RHEL / Fedora `sudo dnf install`
git	`git`	`git`
CMake	`cmake-extras` `cmake-curses-gui`	`cmake`
C/C++/Fortran	`g++ gfortran`	`gcc-c++ gcc-gfortran`
MPI	`mpi-default-dev`	`mpich-devel`
ZLIB	`zlib1g-dev`	`zlib-ng-devel`
ParaView	`paraview-dev`	`paraview-devel`
LAPACK/OpenBLAS[1]	`libopenblas-dev`	`openblas-devel`

continues on next page

[1] Package can be automatically installed through **FLEXI** as compiler option.

Table 2.1 – continued from previous page

Package Installation Command	Debian / Ubuntu `sudo apt-get install`	RHEL / Fedora `sudo dnf install`
HDF5[1]	`libhdf5-mpi-dev`	`hdf5-mpich-devel`
FFTW[Page 5, 1]	`libfftw3-dev`	`fftw-devel`

Tip: On RPM-based distributions, you might need to load the MPI module using the command `module load mpi`.

2.1.2 Additional Configuration

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 by entering the following command.

```
ulimit -s unlimited
```

2.2 Obtaining the Source Code

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

- Clone the **FLEXI** repository from GitHub

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

- Download **FLEXI** from GitHub:

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

- Download a release **FLEXI** repository from GitHub

  https://github.com/flexi-framework/flexi/tags

Attention: Cloning **FLEXI** from GitHub may not be possible on some HPC clusters due to restricted internet access. Please refer to the cluster's user instructions for possible remedies, such as establishing a SOCKS proxy on a machine with unlimited internet access, as documented here for the HLRS at the University of Stuttgart.

2.3 Compiling the Code

In order to compile the code, change into the **FLEXI** root directory, create a new sub-folder, and use CMake to configure and compile the code

```
mkdir build
cmake -B build
cmake --build build
```

Custom configuration of compiler options may be done using

```
ccmake -B build
```

For a list of all compiler options see section *Compiler Options*.

The executables `flexi` and `posti_visu` (if enabled) are generated in the sub-directory `build/bin/`.

Note: In the remainder of this user guide, we omit the path to the **FLEXI** executable (and related tools), but assume assume it can be executed directly by typing `flexi`. This can be achieved by defining an *alias* or *symbolic link*, for example.

For Linux beginners, we provide a short explanation on how to achieve this usage behavior. In general, in order to execute a file, the command either has to be in the `PATH` environment variable or you have to enter the full path to it in the terminal. To enable typing only `flexi`, you can add a symbolic link to the **FLEXI** executable in the current directory, e.g., test case folder, by entering

```
ln -s [FLEXI_ROOT]/build/bin/flexi
```

Among the files in the current directory, this symbolic link will be listed as

```
flexi -> [FLEXI_ROOT]/build/bin/flexi
```

2.4 Running the Code

For a first minimal **FLEXI** simulation, navigate to the *cavity* tutorial folder and run **FLEXI**:

```
cd [FLEXI_ROOT]/tutorials/cavity/Basic_Re100
flexi parameter_flexi.ini
```

Convert the output files to the *vtu* format by entering

```
posti_visu cavity_State_0000000.200000000.h5
```

and visualize the generated files using, e.g., **ParaView**. Note that this conversion step requires enabling the `posti_visu` tool by toggling the `POSTI` flag in the CMake configuration (see section *Compiling the Code* above).

CHAPTER

THREE

CODE OVERVIEW

3.1 Feature List

FLEXI currently has the following features implemented, with most of them being covered in Section 5 *Tutorials*.

Equation Systems

- Compressible Euler equations
- Compressible Navier-Stokes equations
- Linear scalar advection and diffusion equations
- Reynolds-averaged Navier–Stokes equations using Spalart–Allmaras turbulence model

Space Discretization

- Discontinuous Galerkin Spectral Element Method (DGSEM) [1, 2]
  - Legendre Gauss nodes
  - Legendre Gauss Lobatto nodes
- Finite Volume (FV) shock-capturing by either
  - Switching to finite volume subcells [3] or
  - Blending the finite volume operator [4]
  - Several shock indicators available

Time Discretization

- Explicit Runge-Kutta (RK) schemes
  - Standard RK schemes
  - Low storage RK schemes [5]
  - Strong stability preserving RK methods [6]

Computational Domain

- Two- or three-dimensional domains
- Curved Meshes
- Nonconforming Meshes via mortar interfaces [7]
- Sponge zone [8]
- Boundary conditions
  - Various subsonic inflow and outflow conditions [9]
  - Exact boundaries (Dirichlet)
  - Periodic boundaries
  - Slip wall (Euler wall)
  - Non-slip walls (Navier-Stokes wall): adiabatic / isothermal

Numerical Scheme

- Classical and split-form DG schemes [10]
- Dealiasing [11]
  - Filtering
  - Overintegration
- Riemann solvers
  - Local Lax-Friedrichs
  - HLL
  - HLLC
  - Roe-Pike
- Lifting methods
  - Bassi Rebay 1 [12]
  - Bassi Rebay 2 [12]
- Time averaging

3.2 Compiler Options

The following table describes the most important configuration options which can be set when building **FLEXI** using CMake. Some options are dependent on others being enabled (or disabled), such that the available ones may change upon reconfiguring.

Table 3.1: Compiler Options

Build Option	Possible Values	Description
`CMAKE_BUILD_TYPE`	Release / Profile / Debug	normal execution / performance profiling using *gprof* / debug compiler for detailed error messages during code development
`CTAGS_PATH`		install directory of *Ctags*, an optional program used to jump between tags in the source files (see e.g. the implementations Exuberant Ctags or Universal Ctags)
`LIBS_BUILD_HDF5`	on / off	will be set to *on* if no pre-built HDF5 installation was found on your machine to build a HDF5 version during compilation
`HDF5_DIR`		specify the directory of a pre-built HDF5 library that was built using the CMake system, this directory should contain the CMake configuration files (e.g. *hdf5-config.cmake*)
`FLEXI_2D`	on / off	set to *on* to run two-dimensional simulations, in this case you have to provide a mesh that consists of only one layer of elements in the third dimension
`FLEXI_EQNSYSNAME`	linearscalaradvection / navierstokes / rans_sa	linear scalar advection-diffusion equation / Navier–Stokes equations / Reynolds-averaged Navier–Stokes equations using Spalart–Allmaras turbulence model
`FLEXI_FV`	off / switch / blend	Finite-Volume (FV) shock-capturing: disabled / by *switching* DG elements in FV subcell representation [3] / by *blending* the FV and the DG operator [4]
`FLEXI_FV_RECONSTRUCTION`	on / off	only available if `FLEXI_FV` is set either to *switch* or *blend*, enables the linear reconstruction of the solution at the FV subcell faces (second-order FV scheme) and is needed for the calculation of parabolic gradients
`FLEXI_LIFTING`	br1 / br2	lifting method to compute the DG gradients in the parabolic terms: first [12] / second [13] method of Bassi and Rebay
`LIBS_USE_MPI`	on / off	define whether to compile with MPI (necessary for parallel execution)
`FLEXI_NODETYPE`	gauss / gauss-lobatto	node-set used to define the basis functions of the DG method, see [1] for details

continues on next page

Table 3.1 – continued from previous page

Build Option	Possible Values	Description
LIBS_USE_PAPI	on / off	enable to use the PAPI library to perform performance measurements (e.g. flop counts)
FLEXI_PARABOLIC	on / off	define whether the parabolic term of the chosen equation system should be included or not, more efficient than simply setting the diffusion coefficient to zero since the gradients do not need to be discretized
FLEXI_POLYNOMIAL_DEGREE	N / {1,2,3,...}	polynomial degree of basis functions: to be set in parameter file / compile with fixed degree for performance (1,2,3,...)
FLEXI_SPLIT_DG	on / off	enable the split form of the DG operator, allows to use kinetic energy or entropy stable flux functions
FLEXI_TESTCASE	default / hit / phill / riemann2d / taylorgreenvortex / channel	some benchmark simulation setups are encapsulated in test cases (separate subfolders) with case-specific initialization, analyze routines, boundary conditions, etc., while the default test case does not include any additions: see section *Tutorials* for more details
FLEXI_VISCOSITY	constant / sutherland / powerlaw	modeling approach for the dynamic viscosity: constant / Sutherland's law / power law
POSTI	on / off	enable to also build the post-processing toolset next to the actual simulation code, the specific tools can be selected once this flag is enabled
POSTI_VISU_PARAVIEW	on / off	enable to build the **ParaView** plugin for the visualization of **FLEXI** simulation data, the ParaView libraries must be available on the systems and environment variable $ParaView_DIR set accordingly
FLEXI_PERFORMANCE	on / off	enables a set of advanced features to improve the performance of FLEXI
FLEXI_PERFORMANCE_OPTILIFT	on / off	enable to lift only the gradients of the variables in the flux function of the selected equation system improves the performance for FLEXI_PARABOLIC=ON, but cannot be used if posti tool-set is built (POSTI=ON)
FLEXI_PERFORMANCE_PGO	on / off	enables profile-guided optimization (PGO) for compilation, currently only supported with GNU compiler the required two-step compilation process is detailed in section *Performance Improvements*

CHAPTER

FOUR

WORKFLOW

This chapter describes the complete process of performing a simulation in **FLEXI**. The process comprises the mesh generation using the high-order pre-processor **HOPR**, the actual simulation of the numerical problem, and the post-processing step using the **POSTI** toolchain. An overview of this workflow and the components of **FLEXI** is given in the flowchart below.

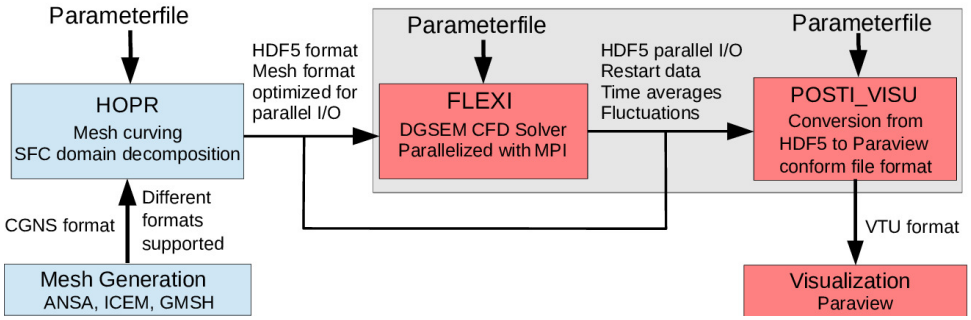

Fig. 4.1: Basic modules and files used by **FLEXI**.

Note that both, **HOPR** and **FLEXI**, use the *HDF5* format to output mesh files and simulation states, respectively. HDF5 is a widely used data model, library, and file format for storing and managing data. It supports an unlimited variety of data types, and is designed for flexible and efficient I/O, and for high volume, complex data. We refer to the HDF5 website for further information on this file format in general.

4.1 Mesh Generation using HOPR

FLEXI obtains its computational meshes from the high-order preprocessor **HOPR** (available under GPLv3 at HOPR project) in HDF5 format. The design philosophy is that all tasks related to mesh organization, different input formats and the construction of high-order geometric mappings are separated from the *parallel* simulation code. The *serial* standalone framework **HOPR** has been developed to generate high-order meshes from input data by external linear mesh generators, supporting different file formats. It also provides a built-in mesh generator for simple, structured meshes. **HOPR** can either be compiled from the source code or be used directly via the provided AppImage, both available on its github page.

The basic command to run **HOPR** is

```
hopr parameter_hopr.ini
```

where the path to the **HOPR** executable has been omitted for simplicity. The test cases provided in chapter *Tutorials* come with both a ready-to-use mesh file and 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 **FLEXI** parameter file.

```
MeshFile=path/to/mesh/file.h5
```

4.2 Build Configuration

Before setting up a simulation, the code must be compiled in the desired configuration. An overview of the most commonly used compiler options is given in section *Compiler Options*. The default configuration solves the three-dimensional Navier–Stokes equations using the pure DG operator (no FV shock-capturing) and does not compile any **POSTI** tools.

4.2.1 Using CMake Presets

The build configurations used for the *Tutorials* are stored as *CMake presets* (human-readable format) in `CMakePresets.json` located in the FLEXI root directory. They can be applied by creating a build folder, reading the desired preset and compiling the code:

```
mkdir build
cmake -B build --preset <preset_name>
cmake --build build
```

Caution: CMake presets were introduced in CMake version 3.19. For earlier versions, **FLEXI** can only be configured manually.

4.2.2 Manual Configuration

To configure the code manually, you can use the CMake GUI, which displays brief instructions and descriptions of the compiler options at the bottom of the window. Note that some compiler options are dependent on others, such that you should always *configure* by hitting the `c` key after changing the value of a compile option. In order to change values, use the arrow keys to select a compile option, and hit the enter key to edit its value; boolean options will toggle with the enter key. Once all desired options are set, *generate* the Makefiles by hitting the `g` key, exit by hitting the `q` key and compile using `make`:

```
mkdir build
ccmake -B build
cmake --build build
```

4.3 Parameter File

The computational setup of the considered test case, including solver settings, initial, and boundary conditions, material properties, data output, is specified via a parameter file. This file is typically named `parameter_flexi_ini` and contains a simple list of parameters, given in the form

```
! This is a comment, e.g. section heading
parameter_name = parameter_value
```

Note that the format is *case-insensitive* (Fortran-style) and that some parameters can also be listed multiple times (so-called `CountOptions`).

To get a list and short description of all possible parameters, grouped thematically, run the **FLEXI** help

```
flexi --help
```

To confine the output to the parameters of a certain section or only one specific parameter, respectively, run

```
flexi --help SECTION
flexi --help PARAMETER
```

4.3.1 Solver Settings

The definition of the numerical solver typically covers the following steps.

- **Set the polynomial degree.**

  Define the polynomial degree `N` 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 dealiasing approach.**

  For under-resolved Navier-Stokes simulations, e.g., in an LES setting, dealiasing is important for numerical stability. Various choices are available and set using either *over-integration* or a *split-form* DG scheme. As the performance penalty of over-integration is substantial, the usage of the split formulation is recommended.

  - `OverintegrationType=1` is a filtering strategy, where the complete operator is first evaluated at `N` (U_t^N) and then filtered to a lower effective degree `NUnder` (U_t^{Nunder}). To use this variant, specify `Nunder` to a value smaller than `N`.
  - `OverintegrationType=2` is a filtering strategy, where the operator in reference space, e.g., JU_t, is first projected to the `NUnder` node set before converting it to physical space $U_t^{Nunder} = JU_t^{Nunder}/J^{Nunder}$. This implementation enforces conservation. To use this variant, specify `Nunder` to a value smaller than `N`.
  - `SplitDG` uses a split formulation, requiring the compiler option `FLEXI_SPLIT_DG` to be turned on. The most commonly used options are the kinetic energy stable formulation by Pirozzoli [14] (`PI`), the entropy conservative formulation by Chandrashekar [15] (`CH`) and a flux differencing form equivalent to the standard DGSEM (`SD`).

- **Choose a Riemann solver.**

  The Riemann solver defines how the inter-element coupling is accomplished. The available variants are listed in section *Parameter File*. 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 `RoeEntropyFix`.

- **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 *Parameter File*. By default, the low-storage fourth order Runge-Kutta scheme by [5] is employed.

4.3.2 Initial Conditions

Both initial and boundary conditions are controlled via the so-called `RefState` and `ExactFunction` constructs.

The `RefState` specifies a state vector in primitive form $(\rho, u, v, w, p)^\intercal$. An arbitrary number of reference states can be defined:

```
RefState=(/1,1,0,0,0.71428571/)
RefState=(/1,0.3,0,0,0.71428571/)
```

In this example, the first state describes a parallel flow in x direction at $Ma = 1$, the second state at $Ma = 0.3$, if an ideal gas with $\kappa = 1.4$ is used.

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

```
IniExactFunc=1
```

The associated state vector to be used is determined by

```
IniRefState=1
```

which, in the above example would imply that the first `RefState` is used for initialization.

Note: The implemented exact functions are specific to the equation system and not documented comprehensively. They can be looked up in the source code, for example in `src/equations/navierstokes/idealgas/exactfunc.f90`.

4.3.3 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 set in the **HOPR** parameter file, if necessary.

FLEXI lists the boundaries and their respective boundary conditions during initialization, for example:

```
|                Name      Type      State      Alpha
|       BC_periodicz-         1          0          3
|       BC_periodicy-         1          0          2
|       BC_periodicx+         1          0         -1
|       BC_periodicy+         1          0         -2
```

(continues on next page)

(continued from previous page)

```
|       BC_periodicx-          1          0          1
|       BC_periodicz+          1          0         -3
```

If we wished to apply a Dirichlet boundary condition with `RefState=2` at the two boundaries in y-direction, we would have to add the following lines to the parameter file

```
BoundaryName=BC_periodicy-
BoundaryType=(/2,2/)
BoundaryName=BC_periodicy+
BoundaryType=(/2,2/)
```

Note that the first entry in the brackets specifies `BC_TYPE`, while the second 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, depending on the `BC_TYPE`.

The currently implemented boundary conditions for the *Navier-Stokes equations* are listed in the table below. See [9] for details on the listed inflow/outflow boundary conditions.

Table 4.1: Boundary conditions.

Boundary Condition	BC_TYP	BC_STATE	Comment
Periodic BC	1	–	Can only be defined in HOPR
Weak Dirichlet	2	`RefState`	
Weak Dirichlet	12	–	Like 2, but using an external state set by `BCStateFile`
Weak Dirichlet	22	`ExactFunction`	Like 2, but using an `ExactFunction`
Wall adiabatic	3	–	
Wall isothermal	4	`RefState`	Isothermal wall, temperature is specified via p and ρ contained in the `RefState`
Wall slip	9	–	Slip, symmetry or Euler wall
Outflow Mach number	23	`RefState`	
Outflow Pressure	24	`RefState`	
Outflow Subsonic	25	`RefState`	
Inflow total pressure / temperature	27	`RefState`	**Special Refstate:** *total* quantities $(T_t, \alpha, \beta, 0, p_t)$

4.3.4 Material Properties

At present, the only available equation of state in the *Navier-Stokes* solver of **FLEXI** is the ideal gas,

$$p = \rho R T$$

with the gas constant R. The heat flux follows Fourier's law

$$\vec{q} = -\lambda \nabla T \qquad \text{with} \quad \lambda = \frac{\kappa R \mu}{(\kappa - 1)\mathrm{Pr}}$$

where λ denotes the heat capacity ratio, Pr the Prandtl number and μ the dynamic viscosity.

These parameters are specified in the parameter file using `R`, `kappa`, `Pr` and `mu0`, respectively.

4.3.5 Data Output

The end time of the simulation is set using `tEnd`. **FLEXI** features several analyze routines, which evaluate the current solution and are invoked every time interval `Analyze_dt`.

Specifically, the following evaluations are possible:

- `CalcErrorNorms=T`: Calculate the L_2 and L_∞ error norms based on the specified `ExactFunc` as reference. This evaluation is used for, e.g., convergence tests.
- `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.
- `CalcBulkState=T`: Calculate the bulk quantities, such as the bulk velocity for the channel flow.
- `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.

The solution itself is dumped to hard drive every `Analyze_dt` as well, unless a multiple of this time interval is specified via `nWriteData`. For example, `nWriteData=10` means that the solution output is performed every tenth analyze time step only.

4.4 Running the Simulation

In general, the simulation is started by running

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

The restart file is optional and allows resuming the simulation from any existing state file.

> **Attention:** When restarting from an earlier time (or zero), all later state file possibly located in the present directory are deleted!

The simulation code is specifically designed for (massively) parallel execution using the MPI library. For parallel runs, the code must be compiled with `LIBS_USE_MPI=ON`. Parallel execution is then controlled using `mpirun`

```
mpirun -np <no. processors> flexi parameter.ini [restart_file.h5]
```

4.4.1 Domain Decomposition

The grid elements are organized along a space-filling curve (SFC), which gives a unique one-dimensional element list. The SFC type is controlled by **HOPR**, with the Hilbert curve set as default. In a parallel run, the mesh is partitioned into as many subdomains as deployed processors simply by splitting the SFC evenly. 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 must not exceed the number of mesh elements.

4.4.2 Choosing the Number of Cores

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

$$PID = \frac{WallTime \times \#Cores}{\#DOF \times \#TimeSteps \times \#RKstages}$$

and measures the wall time per degree of freedom and stage of the time integration scheme. During runtime, the average PID is displayed in the output as

```
CALCULATION TIME PER STAGE/DOF: [ 5.59330E-07 sec ]
```

When compared to the single-core performance, it can be used as a parallel efficiency metric. The PID mainly depends on the processor workload

$$Load = \frac{\#DOF}{\#Cores}$$

and the polynomial degree N. Processor workloads for optimal performance lie in the range $Load = 2000 - 5000$. A recent performance analysis on the HPE Appollo System *HAWK* using AMD EPYC 7742 CPUs is given in [16]

4.5 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 test cases are contained in the folder `src/testcase/` and define standardized interfaces for initialization, source terms and analysis routines

Table 4.2: Test case interfaces.

Interface Name	Description	Example
InitTestcase	Read in test case related parameters from the **FLEXI** parameter file, initialize the corresponding data structures	Prescribed mass flow for `phill` testcase
FinalizeTestcase	Deallocate test case specific data structures	
ExactFuncTestcase	Define test case specific analytic expressions for initial or boundary conditions	
CalcForcing	Compute test case specific source terms	pressure gradient in test case `channel`
TestCaseSource	Add test case specific source terms to equation system	apply pressure gradient in test case `channel`
AnalyzeTestCase	Perform test case specific diagnostics	evaluate dissipation rate of testcase `taylorgreenvortex`

The compiler option `FLEXI_TESTCASE` sets the current test case. 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 \label{missing:phill_testcase}
- `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.

4.6 Post Processing

4.6.1 Overview of Toolchain

FLEXI comes with a post-processing tool-chain that is enabled through the compiler option `POSTI`. This **POSTI** tool-chain converts the **FLEXI** simulation results, stored in a custom data format in HDF5 files, into standardized data formats like *vtu*, which enable further post-processing and are readable by **ParaView**. Additionally, the tool-chain allows computing other quantities of interest derived from the stored variables. Depending on the type of data output, there are different **POSTI** tools that can be used. The data types typically generated by simulations are as follows:

- `StateFile`

  The transient flow state is stored in a so-called *StateFile*. Aside from the solution vector of the conserved variables `(Density, MomentumX, MomentumY, MomentumZ, EnergyStagnationDensity)`, it contains all relevant information to ensure a restart of the associated simulation time.

- `TimeAverage`

  It is possible to carry out a statistical analysis (mean values and fluctuations) of specific variables during the simulation. A large number of variables can be analyzed statistically, including variables derived from the conserved variables. These statistics can be used, for example, to determine the local Reynolds stresses in the simulation. The associated averaging interval corresponds to the output interval. It is also possible to merge consecutive files and increase the effective averaging interval.

- `BaseFlow`

  This file is primarily required for consistent restarts. It stores a moving time average that is used for sponge zones, among others.

- `RecordPoints (RP) / probe data`

  In order to save data with a high temporal resolution without using too much memory, **FLEXI** offers the option of defining recordpoints/point probes. The data of the point samples are stored in the RP files for further processing.

- `CSV files`

  Depending on the selected settings, evaluations are calculated directly by **FLEXI** at runtime and are exported as CSV files. These files require no further processing and can be used directly for analysis.

The most relevant **POSTI** tools are listed with a short description in the Table below.

Table 4.3: Most relevant **POSTI** tools.

POSTI tool	Description
POSTI_AVG2D	Averages a 3D solution file to a 2D solution file, requires an ijk-sorted mesh.
POSTI_MERGETIMEAVERAGES	Merges consecutive TimeAverage files.
POSTI_RP_EVALUATE	Allows to evaluate recordpoints from a solution file after the simulation.
POSTI_RP_PREPARE	Generates the recordpoints file for usage during or after the simulation.
POSTI_RP_VISUALIZE	Converts the recordpoints raw data into post-processable data.
POSTI_SWAPMESH	Swaps the mesh of a solution file for a new mesh with a different spatial resolution.
POSTI_VISU	Converts the volume solution to files readable by e.g. Paraview.

4.6.2 Basic Usage

In the following, the workflow on how to use the **POSTI** tools in general is briefly described at the example of `POSTI_VISU`.

Most **POSTI** tools have a help function that describes how to use the tool and the available parameters. This help can be invoked by running the tool with the flag `--help`, in this case

```
posti_visu --help
```

The `POSTI_VISU` tool reads a separate parameter file as optional first argument, while the files to be visualized are passed as the last argument. The latter can be a single file or several files, specified either as simple space-separated list like `Testcase_State_0.h5 Testcase_State_1.h5` or via standard wildcarding like `Testcase_State_*.h5`. The file must contain the entire volume solution, i.e., can be a StateFile or a TimeAverage file, for example.

For serial execution, the `POSTI_VISU` tool is invoked by entering

```
posti_visu [parameter_postiVisu.ini [parameter_flexi.ini]] <statefiles>
```

The tool also runs in parallel by prepending `mpirun -np <no. processors>` to the above command, as usual, provided the compiler option `LIBS_USE_MPI` is enabled.

```
mpirun -np <no. processors> posti_visu [parameter_postiVisu.ini [parameter_
↪flexi.ini]] <statefiles>
```

The most important runtime parameters to be set in `parameter_postiVisu.ini` are listed in table Table 6.1 in section *POSTI_VISU*.

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

```
NVisu   = 10
varName = MomentumX
varName = VelocityX
```

(continues on next page)

(continued from previous page)

```
varName = Density
varName = Pressure
varName = Temperature
```

CHAPTER

FIVE

TUTORIALS

This chapter provides a detailed overview of flow simulations with **FLEXI**, assuming familiarity with setting compiler options and code compilation. The path to all executables is omitted here. We assume you have either symlinked **flexi**, **hopr**, and all posti tools into the runtime directory or call these executables at their relative location.

Each tutorial directory contains the necessary .ini files - `parameter_hopr.ini`, `parameter_flexi.ini`, `parameter_postiVisu.ini` - as well as the mesh file `*_mesh.h5` in HDF5 format (generated with **HOPR**).

Listing 5.1: Directory tree for a tutorial.

```
tutorial
├── mesh.h5
├── parameter_flexi.ini
├── parameter_hopr.ini
└── parameter_postiVisu.ini
```

Tip: While each tutorial can be run directly in its own directory, we recommend copying each folder to a new directory. This way, you can run the simulations and freely modify the .ini files without altering the original setup.

5.1 Linear Scalar Advection-Diffusion Equation

The three-dimensional linear scalar advection-diffusion (LinAdvDiff) equation implemented in **FLEXI** provides a simple and computationally efficient system of equations for testing and feature development:

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

Here, a scalar solution Φ is advected with a constant three-dimensional velocity $\mathbf{u}$ and experiences diffusion with a constant scalar diffusion coefficient d. This equation is especially useful for evaluating the basic properties of the DG operator, which we will explore in this tutorial. The tutorial is located at `tutorials/linadv`.

5.1.1 Theoretical Background

The dispersion and dissipation properties of the DGSEM operator can be analyzed [17] by examining the evolution of solutions for the one-dimensional linear scalar advection equation for a specific initial wave number in a simplified setup without physical dissipation ($d = 0$). A detailed discussion can be found in the cited paper; here, only a short summary is presented.

Under these preceding conditions, the equation reduces to

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

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

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

where u is a constant scalar transport velocity, $\omega = ku$ the angular frequency, and k the wavenumber. Assuming a uniform mesh with mesh size Δx, we seek numerical solutions of the form

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

where $\mathbf{\Phi}^l$ is a vector containing the degrees of freedom within cell l, and $\hat{\mathbf{\Phi}}$ is a complex amplitude vector, both of size $N+1$. Focusing on the semi-discrete system (only spatially discretized), we can write the system in matrix notation within a single element, yielding an algebraic eigenvalue problem

$$\underline{\underline{A}}\hat{\mathbf{\Phi}} = \Omega\hat{\mathbf{\Phi}},$$

with $\Omega = \frac{\omega\Delta x}{a}$. The matrix $\underline{\underline{A}}$ represents the spatial discretization and is a function of the non-dimensional wavenumber $K = k\Delta x$.

Examining the solutions to this eigenvalue problem reveals relationships for the dissipation and dispersion behavior inherent to the (spatial) numerical scheme for different wavenumbers. In Fig. 5.1, we plot these relationships as functions of 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.

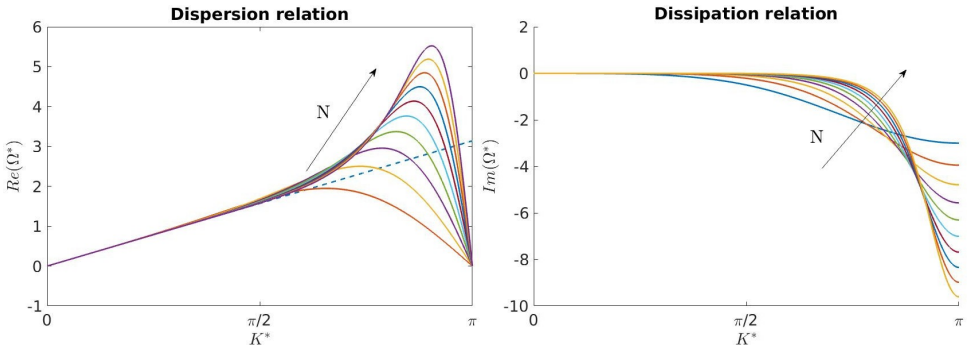

Fig. 5.1: Dispersion and dissipation relationship for $N = \{1,\dots,10\}$ over the modified wavenumber K^*. For the dispersion, the dashed line gives the exact relation.

The quantities are normalized by the number of grid points, such that $K^* = \frac{K}{N+1}$ 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 required to resolve a wave according to the Nyquist theorem, while a normalized wavenumber of 0 corresponds to a constant solution.

As seen in the plot, the dissipation properties of higher-order approximations are significantly improved over those of low-order schemes, as they can preserve higher-frequency waves without significant dissipation. However, there is a sharp increase in dissipation error at the high modes associated with higher-order approximations, which is one of the reasons for scaling down the timestep for larger values of N.

We will now demonstrate these properties through numerical experiments using the linear scalar advection equation.

5.1.2 Build Configuration

In order to use the LinAdvDiff equations, the equation system must be specified during the configuration by setting `EQNSYSNAME=linearscalaradvection`. Since we do not consider diffusion in this tutorial, either turn off the parabolic terms through the build option `FLEXI_PARABOLIC=OFF` or simply specify a zero diffusion coefficient in the parameter file, `DiffC=0`. The required options are set automatically by compiling **FLEXI** with the `linadv` preset using the following commands.

```
cmake -B build --preset linadv
cmake --build build
```

5.1.3 Mesh Generation

We want to perform numerical experiments by solving the one-dimensional linear scalar advection equation. We initialize the simulation with a single wave with a specific angular frequency and observe the behavior of this wave as it evolves, depending on the normalized, non-dimensional wavenumber K^* and the polynomial degree N. Since the analysis in the previous section was based on an infinite domain, the computational domain needs to be large enough to neglect influences from the boundaries. Although periodic boundary conditions could be used as an alternative, this would restrict the choice of wavelengths since they have to fit in the domain and could also adversely affect the stability of the time discretization. Therefore, we opt for an enlarged domain setup and focus on the evolution of the wave in a single element at the center of the domain.

For this tutorial, we create a (quasi-)one-dimensional, equidistant grid by discretizing the interval $x \in [-61, 61]$ with 61 elements, resulting in $\Delta x = 2$, which matches the reference element size. To achieve a (quasi-)one-dimensional simulation, we impose periodic boundary conditions in y and z direction. The boundary conditions in the x-direction are less critical due to the large extent of the domain, so we apply a simple Dirichlet-type boundary conditions (BC type 2) with the analytical wave function as the boundary state. This can be achieved by setting the `BC_STATE`, which means that the initialization function will be used instead of a separate function. Recall that for the LinAdv case, information propagates with the velocity u. Thus, if the boundaries are a distance L away from the cell of interest, solutions can be computed up to time $t = \frac{L}{u}$ without any influence of the boundary conditions.

In the tutorial directory, we provide the necessary mesh file, `CART_1D_mesh.h5`, along with a parameter file for **HOPR** to generate this mesh. You can recreate the mesh by running the following command.

```
hopr parameter_hopr.ini
```

5.1.4 Simulation Parameters

The parameter file to run the simulation is supplied as `parameter_flexi.ini`. The parameters specific to the LinAdvDiff equation system can be found in the `EQUATION` section of the file.

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

The parameter `AdvVel` sets the advection velocity in all three spatial directions. In our one-dimensional simulation, only the first velocity component is non-zero. We set the diffusion coefficient `DiffC` to zero to eliminate physical diffusion, although this is not strictly necessary since the parabolic terms have already been excluded via the build configuration.

The initial condition is derived from the earlier analysis outlined above, with the exact solution given by

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

The amplitude of the wave is given by the real part of this complex expression

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

This function is implemented as `ExactFunc` in the LinAdvDiff equation system and is called by specifying

```
IniExactFunc   = 6
```

in the parameter file. This requires definition of the angular frequency ω, since the wavenumber is given by $k = \frac{\omega}{u}$, so

```
OmegaRef       = 2.
```

Note that the `ExactFunc` function implements the advection velocity and diffusion coefficient as fixed default values, which override the parameters specified in the parameter file. To focus on the behavior of the spatial discretization while minimizing the error from time discretization, we set the CFL number to a small value

```
CFLscale       = 0.1
```

Given that this tutorial involves a quick computation, we can utilize the visualization routines during runtime, eliminating the need for post-processing with the `posti_visu` tool. This is accomplished by enabling the *vtu* output format. With this configuration, the visualization routines are called whenever the analysis routines are executed, with `NVisu` defining the polynomial degree of the output basis. Since we do not specify `Analyze_dt` in the parameter file, the analysis routines will only be invoked at the beginning and end of the simulation.

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

5.1.5 Simulation and Results

We start with simulating a well-resolved wave using a moderate polynomial degree of $N = 4$, which is already configured in the parameter file. For a modified wavenumber of $K^* = \frac{1}{4}\pi$, the dissipation and dispersion relations indicate that we can expect very small errors. Using the relations above, the corresponding angular frequency ω can be calculated as

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

Hence, we adjust the frequency in the parameter file to

```
OmegaRef      = 1.96349540849
```

All other parameters can remain unchanged. The simulation will run until $t = 5$ and is started by executing

```
flexi parameter_flexi.ini
```

After the simulation has completed, we can examine the results by opening the file `LinAdvCosineWave_Solution_0000005.000000000.vtu` with **ParaView**. As noted earlier, we are only interested in the central element within $x \in [-1, 1]$, so we can clip everything else. Moreover, given that we are performing a one-dimensional simulation, we can extract the solution along the x-axis by creating a simple line plot. To see how dispersion and dissipation introduced by the numerics influence our results, we will also overlay the analytical solution introduced above. The result is depicted in figure Fig. 5.2 and shows very little deviation from the analytical solution, as expected, since we are analyzing a well-resolved wave with a low value for K^*.

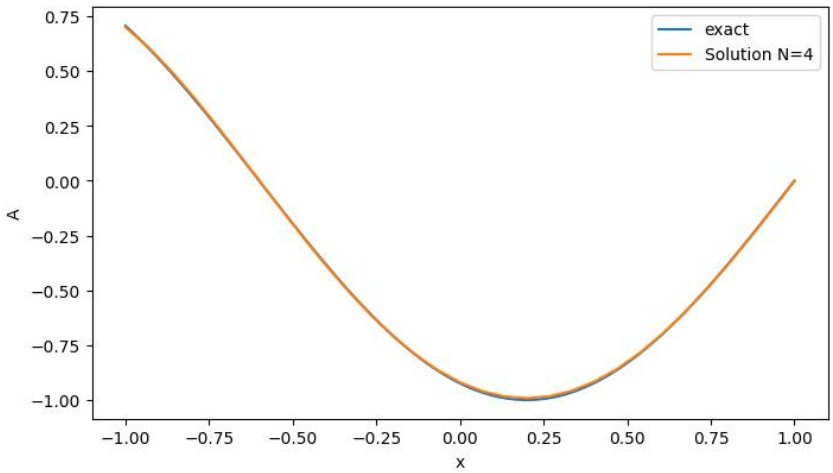

Fig. 5.2: Result for a well-resolved wave with $K^* = \frac{1}{4}\pi$ at $t = 5$ for $N = 4$, in comparison with the exact wave transport.

Of course, it is much more interesting to examine the behavior of waves that are not well-resolved and how the polynomial degree affects their dynamics. To explore this, we run four simulations with polynomial degrees ranging from 2 to 11 and setting the normalized wavenumber to $K^* = 1.6$. Table 5.1 gives an overview of the polynomial degrees and the resulting angular frequencies.

Table 5.1: Angular frequencies needed to attain $K^* = 1.6$.

N	OmegaRef
2	2.4
4	4.0
6	5.6
11	9.6

Next, we will run these four simulations by adjusting the values for `N` and `omegaRef` in the parameter file prior to each run. Remember to give each simulation a unique `ProjectName` to prevent overwriting the results. Additionally, ensure that `NVisu` is set to at least three times the value of N to obtain a meaningful visualization. As we already set `NVisu=30`, simply leave this value unchanged.

The results at $t = 5$, along with the analytical solutions, are shown in Fig. 5.3. Since our comparison uses a constant normalized wavenumber, the actual frequency of the wave increases with the polynomial degree. For $N = 2$ (which should not be considered high-order), we observe both significant dissipation, with an amplitude drop of about 85%, and notable dispersion, that is variations in angular frequency and phase angle. At $N = 4$ (which is at the lower end of what can be considered high-order) there are already some improvements. The amplitude drop is reduced to around 65%, although a phase shift remains clearly visible. This trend continues at higher polynomial degrees. For the highest value of N tested here, we observe only small deviations in the phase angle and the amplitude. These results show how higher-order schemes are able to effectively capture waves with fewer points per wavelength compared to lower-order approximations. This is one of the central aspects why we use such schemes.

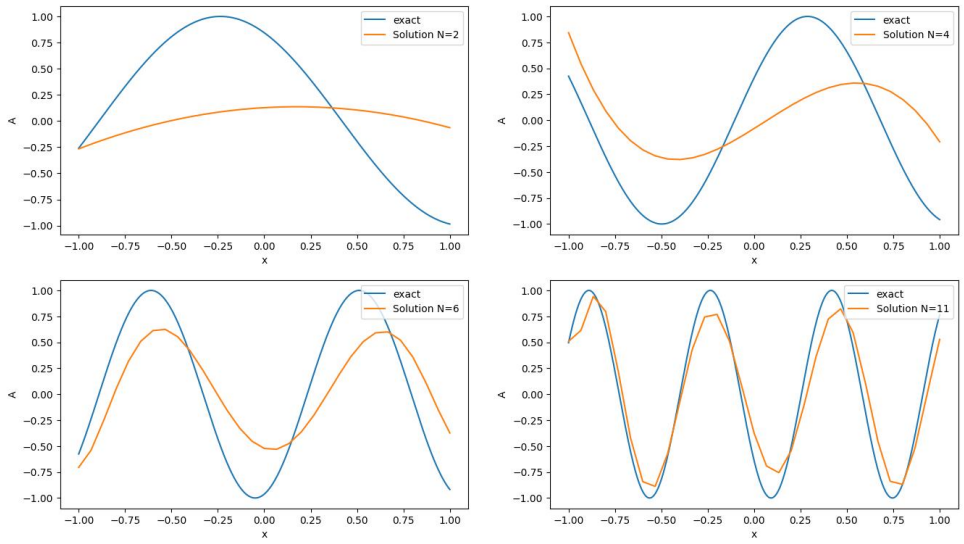

Fig. 5.3: Result for a under-resolved wave with $K^* = 1.6$ at $t = 5$ for $N = 2$ (top left), $N = 4$ (top right), $N = 6$ (bottom left) and $N = 11$ (bottom right).

5.2 Freestream

Unlike the previous tutorial, which dealt with simpler equations, we will now consider the behavior of a compressible fluid flow described by the Navier-Stokes equations. The simplest valid flow solution imaginable is a freestream scenario under conditions of pressure, density, and velocity. For the current setup, we specify a freestream scenario with constant pressure $p = 101325.0$ Pa, density $\rho = 1.225$ kg/m^3 and velocity vector $\boldsymbol{U} = (1, 1, 1)^T$ m/s. This configuration provides a baseline for analyzing how the solver handles simple flow conditions and sets the stage for more challenging simulations.

5.2.1 Mesh Generation

In the tutorial directory, we provide the necessary mesh file, `cartbox_mesh.h5`, along with a parameter file for **HOPR** to generate this mesh. You can recreate the mesh by running the following command.

```
hopr parameter_hopr.ini
```

5.2.2 Build Configuration

FLEXI should be compiled with the `freestream` preset using the following commands.

```
cmake -B build --preset freestream
cmake --build build
```

5.2.3 Simulation Parameters

The parameter file to run the simulation is supplied as `parameter_flexi.ini`. The parameters specific to the Navier-Stokes equation system can be found in the `EQUATION` section of the file.

```
! ============================================================= !
! EQUATION
! ============================================================= !
IniExactFunc  = 1
IniRefState   = 1
RefState      = (/1.225,1.0,1.,1.,101325./)
```

The initial condition is set via the variable vector `RefState` which represents the solution vector $(\rho, u, v, w, p)^T$. **FLEXI** permits for multiple `RefState` vectors, allowing each to be referenced by its corresponding cardinal number for its order number within the .ini file. In this example, only a single `RefState` vector is defined, referenced as 1. Thus, the selection process for the solution vector looks like the following.

- `IniRefState = 1`: the initial condition uses `RefState 1` for the initial flow field solution.
- `IniExactFunc = 1`: the employed exact function routine uses `RefState 1`, such as for calculating the L_2 error norm.

The material properties of the fluid medium, such as the ideal gas constant, are given in Table 5.2 and define the gas behavior in combination with the ideal gas law $p = \rho RT$.

Table 5.2: Material properties for the freestream tutorial.

Variable	Value	Description
mu0	1.8547e-5	Dynamic viscosity μ
R	276	Ideal gas constant R
kappa	1.4	Isentropic coefficient κ

The Discontinuous Galerkin (DG) solution is represented by piecewise polynomials on the computational mesh. In this tutorial, the polynomial degree N is chosen as $N = 3$. The remaining numerical parameters are outlined in Table 5.3.

Table 5.3: Numerical settings for the freestream tutorial.

Variable	Value	Description
N	3	Polynomial degree
MeshFile	cartbox_mesh.h5	Mesh file to be used
tend	1e-6	End time of the simulation
Analyze_dt	1e-6	Time interval for analysis
CFLscale	0.99	Scaling for the theoretical CFL number
DFLscale	0.4	Scaling for the theoretical DFL number

5.2.4 Simulation and Results

We proceed by running the code with the following command.

```
flexi parameter_flexi.ini
```

Running the code prints all output to `STDOUT`. If the run completes successfully, the last lines should appear similar to the following (condensed) output.

```
===============================================================
 INITIALIZATION DONE! [ 0.01 sec ]
===============================================================
---------------------------------------------------------------
 Sys date  :    06.11.2024 14:08:54
#GridCells :    8.0000000E+00
#DOFs      :    5.1200000E+02
#Procs     :    1.0000000E+00
#DOFs/Proc :    5.1200000E+02
 WRITING INITIAL SOLUTION:
---------------------------------------------------------------
 Initial Timestep  :    1.0000000E-06
---------------------------------------------------------------
 Errors of initial solution:
 Sim time   :    0.000E+00
 L_2        :    2.902E-16   2.902E-16   2.902E-16   2.902E-16   3.815E-11
 L_inf      :    6.661E-16   6.661E-16   6.661E-16   6.661E-16   1.164E-10
---------------------------------------------------------------
   Time = 0.0E+00  dt = 0.1E-05  ETA [d:h:m]   0:00:00:00 |>  | [  0.00%]
```

(continues on next page)

(continued from previous page)

```
=============================================================
 FLEXI RUNNING cartbox... [ 0.01 sec ] [ 0:00:00:00 ]
=============================================================
 CALCULATION RUNNING...

-------------------------------------------------------------
 Sys date  :    06.11.2024 14:08:54
 CALCULATION TIME PER STAGE/DOF:            [ 7.05792E-07 sec ]
 EFFICIENCY: CALCULATION TIME [s]/[Core-h]: [ 5.90480E-01 sec/h ]
 Timestep   :    1.063E-04
#Timesteps  :    1.000E+00
 Sim time   :    1.000E-06
 L_2        :    2.902E-16   2.521E-15   2.477E-15   2.513E-15   3.815E-11
 L_inf      :    6.661E-16   1.643E-14   1.421E-14   1.443E-14   1.164E-10
-------------------------------------------------------------
   Time = 0.1000E-05  dt = 0.1000E-05  ETA [d:h:m]:<1 min |==>| [100.00%]
=============================================================
 FLEXI RUNNING cartbox... [ 0.02 sec ] [ 0:00:00:00 ]
=============================================================
=============================================================
 FLEXI FINISHED! [ 0.02 sec ] [ 0:00:00:00 ]
=============================================================
```

Error: If the output does not look like the one above, check for any error messages to diagnose the issue.

After completing the simulation, examine the contents of the working directory. For a successful run, the directory should contain additional generated files named `<PROJECTNAME>_State_<TIMESTAMP>.h5`. Each file stores the solution vector of the conserved variables at each interpolation node at a specific time, corresponding to multiples of `Analyze_dt`.

```
freestream
├── cartbox_mesh.h5
├── cartbox_State_0000000.000000000.h5
├── cartbox_State_0000000.000001000.h5
├── parameter_convert.ini
├── parameter_flexi.ini
├── parameter_hopr.ini
├── parameter_postiVisu.ini
```

5.2.5 Visualization

FLEXI relies on ParaView for visualization. In order to visualize the **FLEXI** solution, its format has to be converted from the HDF5 format into another format suitable for **Paraview**. **FLEXI** provides a post-processing tool *posti_visu* which generates files in VTK format with the following command.

```
posti_visu parameter_postiVisu.ini parameter_flexi.ini cartbox_State_0*
```

5.3 Convergence Test

This tutorial demonstrates how to compute the order of convergence for **FLEXI**. The process is fully scripted, allowing for multiple runs across varied grids and polynomial degrees, with the convergence order calculated automatically. Once the runs are completed, the script generates a plot of the corresponding L_2 error norms and saves it in the directory from which the convergence test was executed. This script is written in Python3.

See also:

The convergence test scripts are provided in the `tools/convergence_test` directory, including the **FLEXI** execution script `tools/convergence_test/execute_flexi.py`.

The convergence test is divided into two parts. The first part examines an inviscid case to determine the order of convergence for the advective terms, applying the Euler equations without any viscous fluxes. In the second part, the viscous convergence test incorporates physical viscosity into the calculation.

5.3.1 Manufactured Solution

To compute the order of convergence in **FLEXI**, we apply a benchmark test where an exact analytical solution is known. For this tutorial, we use the method of manufactured solutions. A detailed description is found in Roache [18]. In this method, a smooth function is proposed the solution to the equation system. Since this function generally does not provide a solution to the system of equations, a source term is calculated to force the corresponding solution. The source term is derived analytically and inserted into the equation system, cf. for the continuity equation, we obtain

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

The source term must be added within the time integration loop of the flow solver. In **FLEXI** sine waves are advected in the density using a constant velocity field. The actual source terms to be considered depend on the equation system, that is whether Euler or Navier-Stokes equations are used.

5.3.2 Mesh Generation

In the tutorial directory, we provide the necessary mesh files, along with a parameter files for **HOPR** to generate these meshes. You can recreate any mesh by running the following command.

```
hopr parameter_hopr.ini
```

Non-Conforming Meshes

FLEXI supports non-conforming meshes with mortar interfaces. For the convergence test, parameter files for four mortar meshes are also provided, recognizable through filenames containing the term *MORTAR*. To use the convergence test script, simply open the script file at `tools/convergence_test/convergence_grid` and replace the mesh filenames. After that, the script can be executed in the same manner as for the conforming meshes.

5.3.3 Inviscid Convergence Test

We focus first on the convergence test without viscous terms, i.e. conference for the Euler equations.

Manufactured Solution

The manufactured solution for the Euler equation reads as

$$\rho = A(1 + B\sin(\Omega|\underline{x} - \underline{v}t|)) \quad \text{with} \quad A, B, \Omega, \underline{v} = const.$$

Since we are working with the Euler equations, the source term is zero, $Q(x,t) \equiv 0$. To investigate the order of convergence for a given polynomial degree N, the mesh resolution must be progressively refined. We provide meshes with 1, 2, 4, and 8 elements in each spatial direction together with the corresponding parameter files in the directory `parameter_hopr`. Fig. 5.4 displays an exemplary mesh used for the convergence test and the flow field solution of the density.

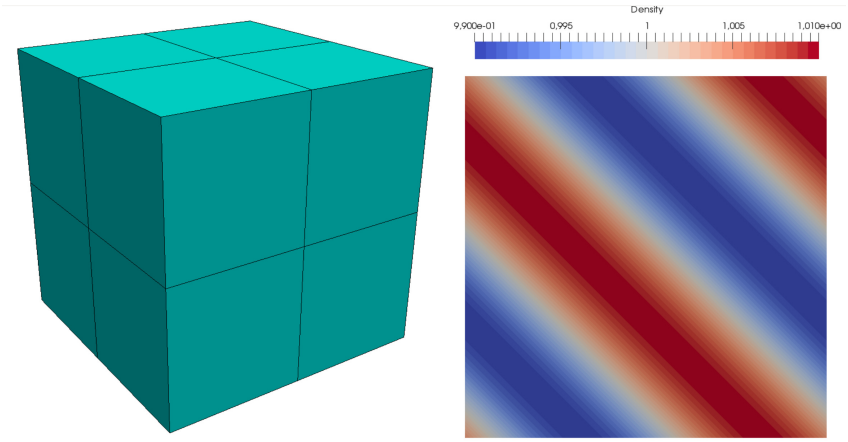

Fig. 5.4: Convergence test: Mesh and flow field solution of the density.

Compiler Options

FLEXI should be compiled with the `convtest_inviscid` preset using the following commands.

```
cmake -B build --preset convtest_inviscid
cmake --build build
```

Simulation Parameters

The inviscid convergence test is run from the parameter file `parameter_convtest_flexi.ini`. Essentially, any valid parameter file can be used since 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 simple baseline test case. The default settings for the time integration are displayed in Table 5.4.

Table 5.4: Numerical settings used for the inviscid convtest.

Variable	Value	Description
N_Analyze	at least $2N$	Number of interpolation nodes for the analyze routines, needed for the calculation of the error norms
IniExactFunc	2	The manufactured solution and the function used to initialize **FLEXI**. It can also be used for Dirichlet BCs.
AdvVel	(/0.3,0.,0./)	constant velocity vector used by specified function
CalcErrorNorms	T	Flag to calculate of L_2 and L_∞ error norms
tend	0.5	End time of the simulation
Analyze_dt	0.5	Time interval for analysis
nWriteData	1	Number of analyze times the state file is written
CFLscale	0.9	Scaling factor for the theoretical CFL number (convective time step restriction)
DFLscale	0.9	Scaling factor for the theoretical DFL number (viscous time step restriction)

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

```
tools/convergence_test
```

Two versions of the script are available. The first script, `convergence_grid`, computes the grid convergence order for a fixed polynomial degree N across progressively refined meshes. The second script, `convergence`, calculates spectral convergence on a fixed mesh by increasing the polynomial degree. 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.

Simulation and Results

We proceed by running the code to investigate the grid convergence with the following command.

```
tools/convergence_test/convergence_grid flexi parameter_convtest_flexi.ini --
↪gnuplot
```

FLEXI outputs its standard log data to the file `ConvTest.log`. Alongside this, a CSV (comma-separated values) file named `ConvTest_convfile_grid.csv` is created, which contains all computed L_2 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 L_2 error of the momentum in x-direction against the number of elements of the meshes. This plot includes a second

curve, representing the theoretical convergence order for the selected polynomial degree, which serves as a benchmark to compare the computed results.

Spectral convergence can be investigated using the following command. Here, the `_grid` of the original command is replaced by `_N`.

```
tools/convergence_test/convergence flexi parameter_convtest_flexi.ini --
↪gnuplot
```

Fig. 5.5 shows the result for grid (left) and spectral (right) convergence.

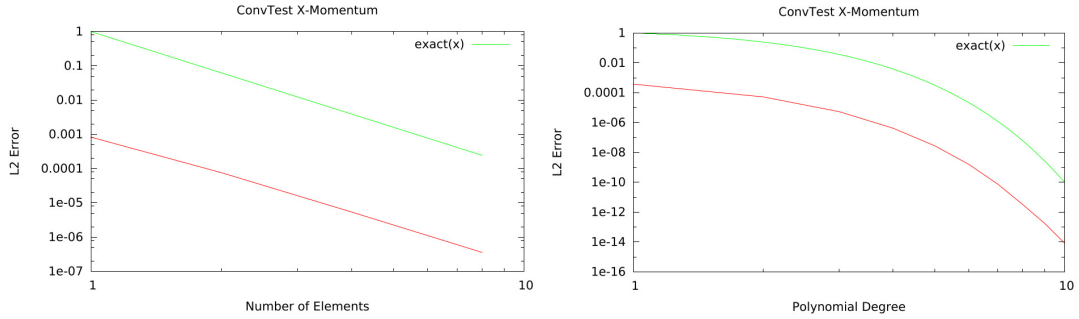

Fig. 5.5: Convergence test: Plot of spectral (right) and grid (left) convergence

5.3.4 Viscous Convergence Test

The second convergence test includes the viscous terms, i.e., convergence for the Navier-Stokes equations.

Manufactured Solution

For this case, another manufactured solution is chosen as

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

The same function is applied to the momentum equations in all spatial directions. The mass specific total energy in this case is $\rho e = \rho\rho$. Since the Navier–Stokes equations are considered, the manufactured solution has a non-zero source term. In **FLEXI**, this source term is added in the routine *CalcSource* in the file

```
src/equations/navierstokes/idealgas/exactfunc.f90
```

Note: This manufactured solution can also be solved without considering the viscous terms. In this case, the source term does not vanish.

Compiler Options

FLEXI should be compiled with the `convtest_viscous` preset using the following commands:

```
cmake -B build --preset convtest_viscous
cmake --build build
```

Simulation Parameters

The viscid convergence test is run from the parameter file `parameter_convtestvisc_flexi.ini`. The default settings for the viscous terms are displayed in Table 5.4.

Table 5.5: Numerical settings used for the viscid convtest.

Variable	Value	Description
IniExactFunc	4	The manufactured solution and the function used to initialize **FLEXI**. It can also be used for Dirichlet BCs.
Viscosity	0.03	Dynamic viscosity μ_0

Simulation and Results

Execution of the viscous convergence tests is analogously to the inviscid case, e.g., with the following command.

```
tools/convergence_test/convergence_grid flexi parameter_convtestvisc_flexi.
↪ini --gnuplot
```

5.4 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., [19], [20]. This tutorial assumes that you have completed the previous tutorial, know how to edit files and post-process 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 :).

This tutorial is divided into two sections. The *Basic* section introduces the setup process and guides you through running simulations, providing a solid foundation for using the code. The *Advanced* section builds on this, offering insights into code modifications that enable more complex simulations and the addition of custom features. If you're mainly interested in running the code as provided, feel free to skip the Advanced section or only explore the parts that interest you.

5.4.1 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 as an example how to set up and run a simulation in **FLEXI**. The computation is conducted in a three-dimensional, square domain with periodic boundary conditions in the "third" direction. The walls of the cavity are modeled 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. Fig. 5.6 shows the resulting velocity field and streamlines for $Re = 400$.

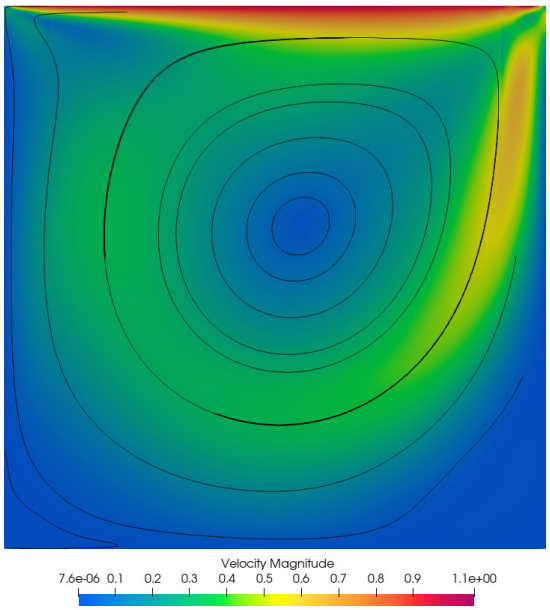

Fig. 5.6: Contours of velocity magnitude for the Re = 400 lid-driven cavity case.

Compiler Options

FLEXI should be compiled with the `cavity` preset using the following commands.

```
cmake -B build --preset cavity
cmake --build build
```

5.4.2 Basic Tutorial | Flow at Re=100

The basic tutorial is contained in the `Basic_Re100` subfolder of the `tutorials\cavity` directory.

Mesh Generation

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 zdirection. Also, we will only use one element in that direction to save computational costs. In the tutorial directory, we provide the necessary mesh files, along with a parameter files for **HOPR** to generate these meshes. You can recreate any mesh by running the following command. A full tutorial on how to run **HOPR** is available at the HOPR documentation.

```
hopr parameter_hopr.ini
```

For this basic tutorial, the simple meshes shown in (Fig. 5.7) will be used.

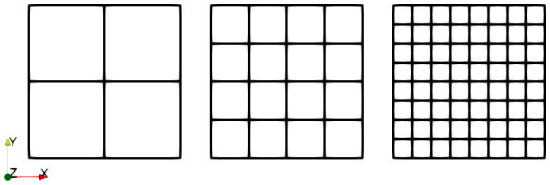

Fig. 5.7: Meshes for the basic lid-driven cavity tutorial.

Simulation Parameters

The parameter file to run the simulation is supplied as `parameter_flexi.ini`. The parameters specific to each topic can be found in the labeled section of the file.

Output

```
! ============================================================= !
! OUTPUT
! ============================================================= !
ProjectName    = Tutorial_Cavity_Re100
OutputFormat   = 0
```

The `ProjectName` parameter defines the prefix for all files generated by the simulation. For example, if the simulation saves the state at time 0.3, it will produce a file named `Tutorial_Cavity_Re100_State_0000000.300000000.h5` which contains the conserved variable values at each node. In this tutorial, the `OutputFormat` parameter is set to 0, which disables on-the-fly output. Although turning on visualization output provides real-time insight, it's generally not recommended because it can significantly slow down the code, especially in parallel executions. Regardless of this setting, HDF5 state files are always generated. For post-simulation visualization, the recommended approach is to use the **posti_visu** tool to create **ParaView**-compatible files. Currently, only the VTK output format is supported (by setting `OutputFormat=3`), as Tecplot output is unavailable due to GPL licensing restrictions.

Interpolation

```
! ============================================================= !
! INTERPOLATION
! ============================================================= !
N             = 3
NAnalyze      = 10
```

The parameter `N` sets the degree of the solution polynomial. In this example, the solution is approximated by a polynomial of degree 3 in each spatial direction. This results in $(N+1)^3 = 64$ 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, i.e., the accuracy of the calculation of error norms or test case specific integrals during the computation. A good rule of thumb is to set $NAnalyze = 2 \times N$.

Numerical Mesh

```
! ============================================================= !
! MESH
! ============================================================= !
MeshFile      = cavity2x2_mesh.h5
useCurveds    = F

BoundaryName  = BC_free
BoundaryType  = (/2,1/)
BoundaryName  = BC_wall_left
BoundaryType  = (/4,1/)
BoundaryName  = BC_wall_right
BoundaryType  = (/4,1/)
! BoundaryName  = BC_wall_lower
! BoundaryType  = (/4,1/)
```

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 must be present in the mesh file and thus match the `BoundaryName` identified used during mesh creation. Each line containing the boundary name must be followed by a line containing the `BoundaryType` identified that what kind of boundary is to be applied to the face. A list of types available for the Navier-Stokes equations can be found in table 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 inflow boundary (Type 2), the full reference state is set at the boundary. Here, `BoundaryType= (/2,1/)` indicates 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. The same reference state is also chosen for the boundaries `BC_wall_left` and `BC_wall_right`. These boundaries selected as isothermal walls (Type 4), so a wall temperature needs to be specified - this is computed from the primitive variables in the associated reference state. For the $z-$oriented faces omitted here,

periodic boundary conditions are selected which must be specified in **HOPR** to pre-compute the correct connectivity information.

Note that the lines for the lower wall boundary are commented out. In this case, the boundary conditions set in **HOPR** will be retained and not overwritten here. 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 should look like the following.

```
 ------------------------------------------------------------
 |           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|           Name   Type  State     Alpha
 |                  |      BC_zminus      1      0         1
 |                  |       BC_zplus      1      0        -1
 |                  |  BC_wall_lower      4      1         0
 |                  |        BC_free      2      1         0
 |                  |   BC_wall_left      4      1         0
 |                  |  BC_wall_right      4      1         0
 .............................................................
```

Equation System

```
! ============================================================ !
! EQUATION
! ============================================================ !
IniExactFunc  = 1
IniRefState   = 2
RefState      = (/1.0,1.,0.,0.,71.4285714286/)
RefState      = (/1.0,0.,0.,0.,71.4285714286/)
mu0           = 0.01
R             = 1
Pr            = 0.72
kappa         = 1.4
```

The equation system in use is set during compilation. However, these equations are unclosed without 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 errors norms and can also be time-dependent. The reference state itself used for the initial function is defined by the parameter `IniRefstate`, in this case, the second one is used. As described above, each `RefState` is given in primitive variables. Here, the second reference state 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 correspond to the values given in Table 5.6. These define the gas dynamics in combination with the ideal gas law $p = \rho RT$.

Important: **FLEXI** does not distinguish between dimensional and non-dimensional quantities. 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 5.6: Material properties for the lid-driven cavity tutorial.

Variable	Value	Description
mu0	0.1	Dynamic viscosity μ
R	1	Ideal gas constant R
kappa	1.4	Isentropic coefficient κ
Pr	0.72	Prandtl number Pr

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

$$Mach = u/c = 1.0/\sqrt{\kappa\frac{p}{\rho}} = 1.0/10.0 = 0.1$$

$$Re = \frac{uL\rho}{\mu_0} = \frac{1.0}{0.01} = 100$$

Temporal Discretization

```
! ============================================================= !
! TIMEDISC
! ============================================================= !
TEnd          = 5.0
Analyze_dt    = 0.1
nWriteData    = 1
CFLscale      = 0.9
DFLscale      = 0.4
```

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 full solution state files in HDF5 format are written to the file system, e.g., in this case `nWriteData` $\times$ `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. However, since the determination of the timestep includes some heuristics, both values might require to be chosen even more conservatively.

Simulation and Results

We proceed by running the code with the following command.

```
flexi parameter_flexi.ini
```

If **FLEXI** was compiled with MPI support, it can also be run in parallel with the following command. Here, <NUM_PROCS> is an integer denoting the number of processes to be used in parallel.

```
mpirun -np <NUM_PROCS> flexi parameter_flexi.ini
```

Important: **FLEXI** uses an element-based domain decomposition approach for parallelization. Consequently, the minimum load per process is *one* grid element, i.e. do not use more processes than grid elements!

Running the code prints all output to STDOUT. If the run completes successfully, the last lines should appear similar to the following (condensed) output. After a successful run, the directory should contain additional generated files named <PROJECTNAME>_State_<TIMESTAMP>.h5. Each file stores the solution vector of the conserved variables at each interpolation node at a specific time, corresponding to multiples of Analyze_dt.

```
-------------------------------------------------------------
 Sys date   :    07.11.2024 13:02:23
 CALCULATION TIME PER STAGE/DOF: [ 4.64195E-07 sec ]
 EFFICIENCY: CALCULATION TIME [s]/[Core-h]: [ 1.23651E+04 sec/h ]
 Timestep   :    4.1665427E-03
#Timesteps  :    1.2000000E+03
 WRITE STATE TO HDF5 FILE... DONE! [ 0.00 sec ]
 Sim time   :    5.000E+00
 L_2        :    1.269E-03   1.810E-01   1.127E-01   5.762E-14   1.568E-01
 L_inf      :    8.090E-03   9.497E-01   3.924E-01   2.020E-13   1.713E+00
 BodyForces (Pressure, Friction) :
 BC_wall_lower   0.00E+00 -7.14E+01  0.00E+00 -2.73E-03  1.80E-05  3.63E-15
  BC_wall_left  -7.13E+01  0.00E+00  0.00E+00  2.77E-04  1.91E-02  3.34E-15
 BC_wall_right   7.14E+01 -8.74E-15 -1.61E-15  9.38E-04 -4.15E-02 -1.78E-15
 Wall Velocities (mean/min/max)  :
 BC_wall_lower   1.138E-02   3.164E-03   2.183E-02
  BC_wall_left   3.604E-02   2.169E-03   1.488E-01
 BC_wall_right   1.114E-01   2.742E-03   3.930E-01
 MeanFlux through boundaries     :
     BC_zminus   3.539E-14  -1.047E-14  -5.820E-15   7.139E+01   8.846E-12
 BC_wall_lower   0.000E+00  -2.737E-03  -7.140E+01   3.633E-15   8.844E-04
       BC_free   3.913E-05  -1.048E-01   7.142E+01   2.564E-15  -4.560E-02
  BC_wall_left   0.000E+00  -7.137E+01   1.914E-02   3.345E-15   1.024E-02
 BC_wall_right   0.000E+00   7.148E+01  -4.159E-02  -3.400E-15   3.555E-02
-------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
   Time = 0.5000E+01  dt = 0.4169E-02  ETA [d:h:m]:<1 min |==>| [100.00%]
===============================================================
 FLEXI RUNNING Tutorial_Cavity_Re100_mesh2x2...! [ 0.38 sec ] [ 0:00:00:00 ]
===============================================================
===============================================================
 FLEXI FINISHED!! [ 0.38 sec ] [ 0:00:00:00 ]
===============================================================
```

Since we start the simulation from a fluid at rest, it will take some iterations / time steps 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. 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. Thus, the velocities at the walls offer a measure of the solution convergence. In Fig. 5.8, the temporal evolution of the velocities at the lower wall are plotted over time for 4 different simulations. For all cases, a sufficiently stationary solution has been achieved at $t_{end} = 5$.

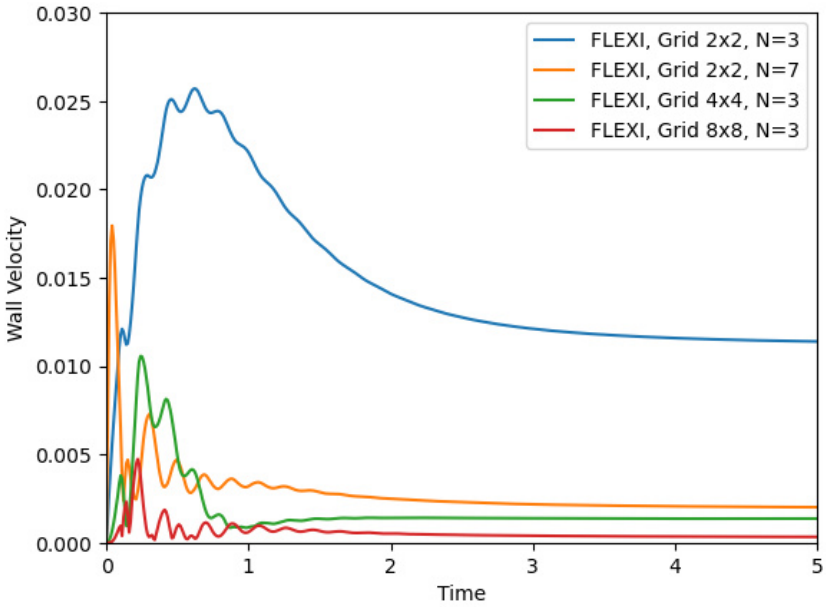

Fig. 5.8: Time evolution of wall velocity at lower wall for the $Re = 100$ lid-driven cavity case.

A contour plot of the velocity magnitude at the end time is given in Fig. 5.9. To generate this plot, convert the *State* files to a **ParaView** format using the **posti_visu** tool.

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. Fig. 5.10 shows the results for the 4 simulations run here, along with published data available in [19], [20].

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 in x- and y-direction. Doubling the grid elements results in visible improvement. Doubling the grid resolution 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 an indication of the excellent convergence properties of high-order schemes for smooth problems.

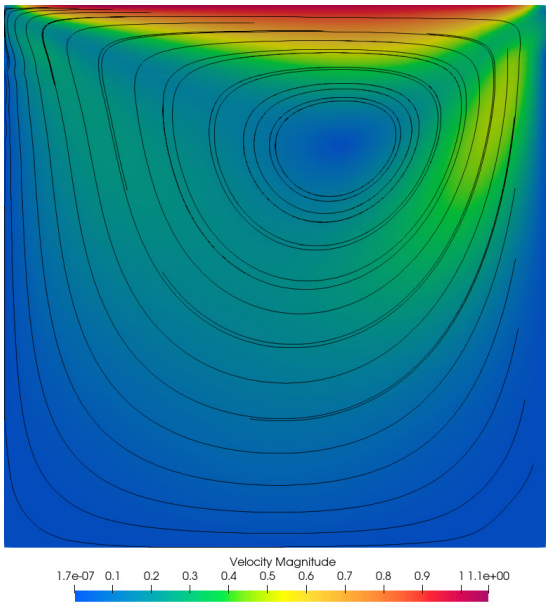

Fig. 5.9: Contours of velocity magnitude for the $Re = 100$ lid-driven cavity case.

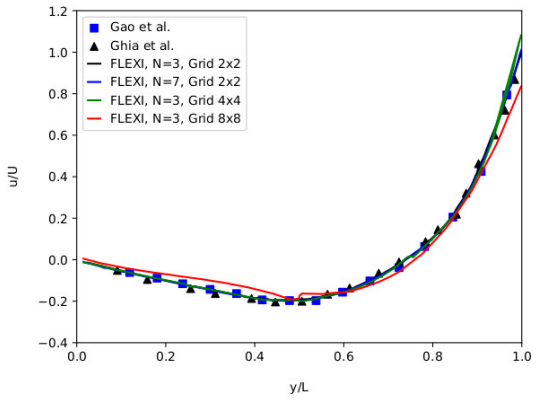

Fig. 5.10: Comparison of centerline velocities for the $Re = 100$ lid-driven cavity case with literature.

5.4.3 Advanced Tutorial | Flow at Re=400

In this section, we build on the concepts covered in the basic tutorial. While the general setup of the simulation remains the same, we increase the Reynolds number, which requires a new, higher-resolution mesh to capture the finer flow details. Additionally, this part introduces basic code customization by showing how to add a new function for custom initial or boundary conditions. Before diving in, it is recommended that you have completed the basic tutorial, have access to at least four computational cores (or be prepared for longer run times), and be comfortable with the modern **Fortran** syntax. The basic tutorial is contained in the `Advanced_Re400` subfolder of the `tutorials\cavity` directory.

Mesh Generation

To account for the increased Reynolds number, the number of elements in the $x-y$-plane is increased to 12×12. Also, a stretching in the $y-$direction is introduced, as depicted in Fig. 5.11. You can generate your own mesh or re-use the provided one, labeled `cavity12x12_stretch_mesh.h5`. In `parameter_hopr.ini`, the following line introduces an exponential stretching.

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

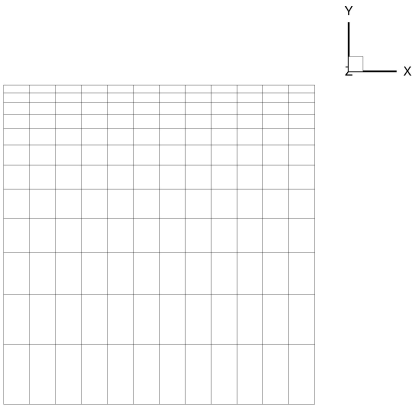

Fig. 5.11: Stretched mesh for the $Re = 400$ lid-driven cavity case.

Custom Initial / Boundary Function

To set up a custom boundary condition for the top boundary, which drives the cavity flow, we define a function for the velocity profile. In this, we follow the suggestions from [20], where the $u(x)$ velocity at the lid is given as

$$u(x) = \begin{cases} c_1x^4 + c_2x^3 + c_3x^2 + c_4x, & \text{if } 0 \leq x < 0.2 \\ d_1x^4 + d_2x^3 + d_3x^2 + d_4x + d_5, & \text{if } 0.8 < x \leq 1.0 \\ 1, & \text{otherwise} \end{cases} \tag{5.1}$$

with

$$\begin{aligned} [c_1, c_2, c_3, c_4] &= 1000 \times [4.9333, -1.4267, 0.1297, -0.0033] \\ [d_1, d_2, d_3, d_4, d_5] &= 10000 \times [0.4933, -1.8307, 2.5450, 1 - .5709, 0.3633] \end{aligned}$$

This assumes the top boundary to be from $x \in [0,1]$, as is the case in our domain. To add this new function to **FLEXI**, locate the file `src/equations/navierstokes/idealgas/exactfunc.f90` and open it in the text editor of your choice. Locate the `SUBROUTINE ExactFunc`, which provides functions with analytic solutions for the boundary condition, initial condition, and analysis routines of the code. The header of the routine you are looking for is shown below.

Listing 5.2: Header of the ExactFunc subroutine.

```
!==================================================================
!> Specifies all the initial conditions. The state in conservative
!> variables is returned. t is the actual time. dt is only needed
!> to compute the time dependent boundary values for the RK scheme
!> for each function resu and the first and second time derivative
!> resu_t and resu_tt have to be defined (is trivial for constants)
!==================================================================
SUBROUTINE ExactFunc(ExactFunction,tIn,x,resu,RefStateOpt)
! MODULES
USE MOD_Preproc        ,ONLY: PP_PI
USE MOD_Globals        ,ONLY: Abort
USE MOD_Mathtools      ,ONLY: CROSS
USE MOD_Eos_Vars       ,ONLY: Kappa,sKappaM1,KappaM1,KappaP1,R
USE MOD_Exactfunc_Vars ,ONLY: IniCenter,IniHalfwidth,IniAmplitude,
↪IniFrequency,IniAxis,AdvVel
USE MOD_Exactfunc_Vars ,ONLY: MachShock,PreShockDens
USE MOD_Exactfunc_Vars ,ONLY: P_Parameter,U_Parameter
USE MOD_Exactfunc_Vars ,ONLY: JetRadius,JetEnd,JetAmplitude
USE MOD_Equation_Vars  ,ONLY: IniRefState,RefStateCons,RefStatePrim
USE MOD_Timedisc_Vars  ,ONLY: fullBoundaryOrder,CurrentStage,dt,RKb,RKc,t
USE MOD_TestCase       ,ONLY: ExactFuncTestcase
USE MOD_EOS            ,ONLY: PrimToCons,ConsToPrim
#if PARABOLIC
USE MOD_Eos_Vars       ,ONLY: mu0
USE MOD_Exactfunc_Vars ,ONLY: delta99_in,x_in
#endif
IMPLICIT NONE
!------------------------------------------------------------------
! INPUT/OUTPUT VARIABLES
INTEGER,INTENT(IN)             :: ExactFunction
REAL,INTENT(IN)                :: x(3)
REAL,INTENT(IN)                :: tIn
REAL,INTENT(OUT)               :: Resu(PP_nVar)
INTEGER,INTENT(IN),OPTIONAL    :: RefStateOpt
!------------------------------------------------------------------
! LOCAL VARIABLES
INTEGER                        :: RefState
REAL                           :: tEval
REAL                           :: Resu_t(PP_nVar),Resu_tt(PP_nVar),ov
REAL                           :: Frequency,Amplitude
REAL                           :: Omega
REAL                           :: Vel(3),Cent(3),a
REAL                           :: Prim(PP_nVarPrim)
```

(continues on next page)

(continued from previous page)

```
REAL                            :: r_len
REAL                            :: Ms,xs
REAL                            :: Resul(PP_nVar),Resur(PP_nVar)
REAL                            :: random
REAL                            :: du, dTemp, RT, r2
REAL                            :: pi_loc,phi,radius
REAL                            :: h,sRT,pexit,pentry
#if PARABOLIC
! needed for blasius BL
INTEGER                         :: nSteps,i
REAL                            :: eta,deta,deta2,f,fp,fpp,fppp,fbar,fpbar,
↪fppbar,fpppbar
REAL                            :: x_eff(3),x_offset(3)
#endif
!================================================================
```

To add equation (5.1) to the code, add a new `CASE` to the routine. In this `CASE`, 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. If the compilation process was not successful, check the compiler output for any error messages to diagnose the issue.

To benefit from this tutorial, it is recommended that you do try to complete this programming task. For reference, the listing below shows an example code for a correct implementation as `CASE(9)`. Note that in the example 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 imposing the Mach number by setting the `IniRefState` accordingly.

Listing 5.3: Example code for a correct implementation from [20].

```
CASE(9) ! Lid driven cavity flow from Gao, Hesthaven, Warburton
        ! "Absorbing layers for weakly compressible flows", 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(:,RefState)
  IF (x(1).LT.0.2) THEN
    prim(VEL1)=1000*4.9333*x(1)**4-1.4267*1000*x(1)**3+0.1297*1000*x(1)**2-0.
↪0033*1000*x(1)
  ELSEIF (x(1).LE.0.8) THEN
    prim(VEL1)=1.0
  ELSE
    prim(VEL1)=1000*4.9333*x(1)**4-1.8307*10000*x(1)**3+2.5450*10000*x(1)**2-
↪1.5709*10000*x(1)+10000*0.3633
  ENDIF
  CALL PrimToCons(prim,Resu)
```

Note: From now on, we refer to the above `CASE(9)` as the case number in question while yours might differ.

Simulation Parameters

To set up the simulation, you can either modify the `parameter_flexi.ini` files from the basic tutorial or use the ones provided in the `Advanced_Re400` directory. We will conduct two simulations for the advanced configuration. The first one uses the constant driving flow boundary conditions as before while the second one utilizes the new custom equation specified as equation `CASE(9)`. In both cases, we need to modify the mesh file, the fluid viscosity (in order to set the Reynolds number), and the end time of the simulation. Here, we chose `TEnd=100` to account for the longer accommodation period required for the simulation to reach a quasi-steady state. As will be seen later, 100 is very conservative, so feel free to lower this value as you see fit.

```
MeshFile      = cavity12x12_stretch_mesh.h5
[...]
mu0           = 0.0025
[...]
TEnd          = 100.0
```

For the custom boundary condition case, the parameter file needs to be adjusted to include 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 thus refers to the equation (`CASE(9)`) which we have programmed above.

```
BoundaryName   = BC_free
BoundaryType   = (/22,9/)
```

Simulation and Results

We proceed by running the code with the following command. If you chose to use the provided .ini files, change the parameter file to `parameter_flexi_custombc.ini` for the second run.

```
flexi parameter_flexi.ini
```

If **FLEXI** was compiled with MPI support, it can also be run in parallel with the following command. Here, <NUM_PROCS> is an integer denoting the number of processes to be used in parallel.

```
mpirun -np <NUM_PROCS> flexi parameter_flexi.ini
```

The evolution of the mean velocity of the lower wall is given in Fig. 5.12. Note the difference in the axis scales compared to Fig. 5.8. It is evident that the flow reaches steady state after about $t = 35$.

In the following, `fig:cavity_re400_velcomp` and Fig. 5.14 show the flow field and comparison of the centerline velocities with published results [19], [20].

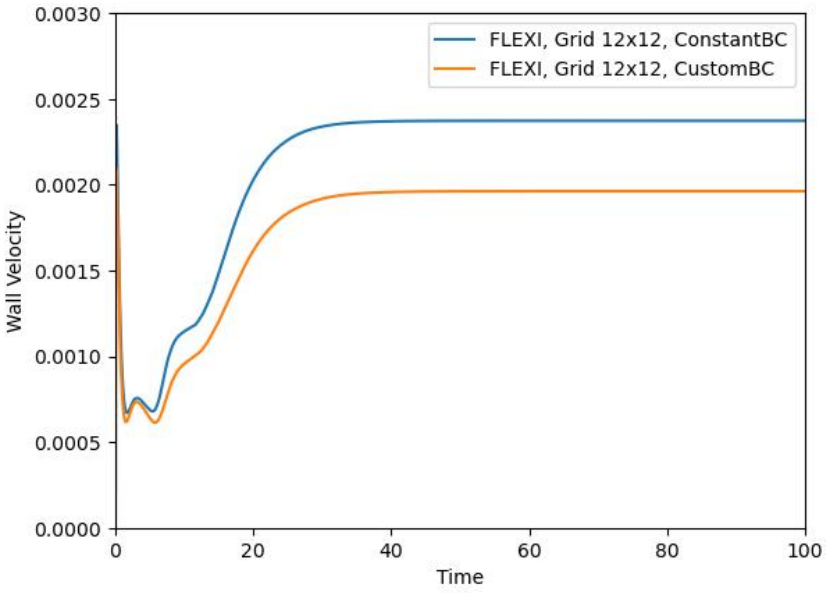

Fig. 5.12: Time evolution of wall velocity at lower wall for the $Re = 400$ lid-driven cavity case.

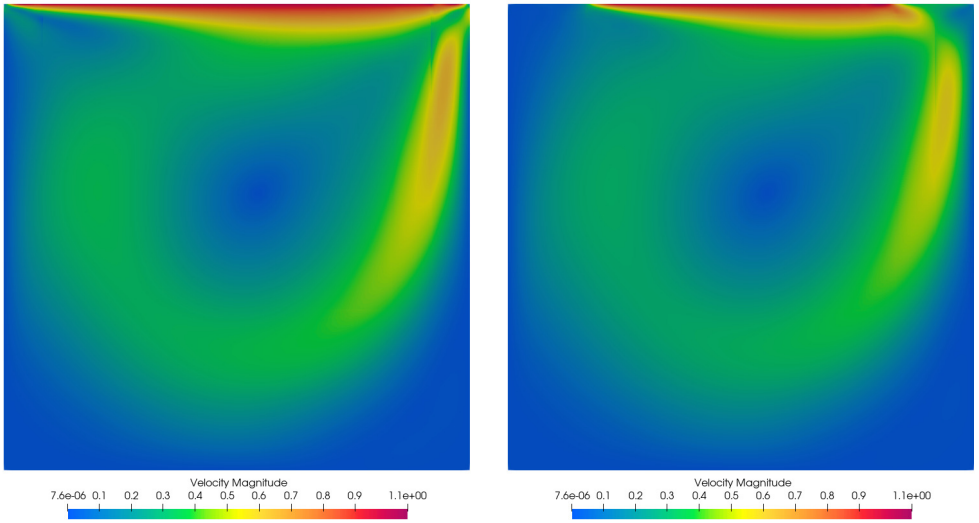

Fig. 5.13: Steady state solution of velocity magnitude of $Re = 400$ lid driven cavity. Left: constant boundary condition, right: custom boundary condition.

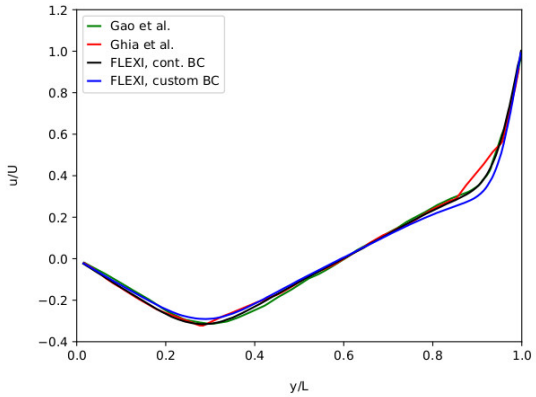

Fig. 5.14: Evolution of wall velocities at the lower wall for $Re = 400$ lid driven cavity simulations.

5.5 Taylor Green Vortex

This tutorial describes how to set up and run a basic test case for turbulent flows, the Taylor–Green vortex (TGV) [21]. The TGV is started from eight Fourier modes with the initial conditions given by Gassner et al. [11]. In this tutorial, we will learn how to avoid catastrophic failure of the code due to non-linear instabilities. This is done by using polynomial dealiasing or entropy/energy stable split flux formulations. In a second step, we add the sub-grid scale (SGS) model of Smagorinsky. The tutorial is located at `tutorials/tgv`.

5.5.1 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 $M = 0.1$. The Reynolds number of the flow is defined as $1/\mu = 6.25 \cdot 10^{-4}$. The domain is set up as a triple periodic box with edge length $L = 2\pi$.

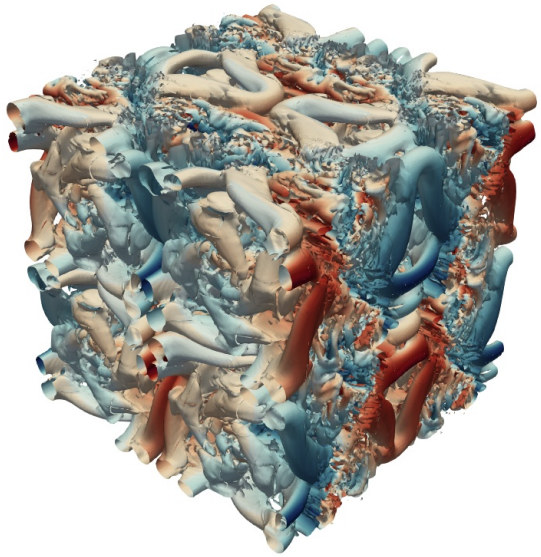

Fig. 5.15: 3D visualization of the Q-criterion of the Taylor–Green vortex.

5.5.2 Mesh Generation

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`), together with the default mesh. Using 4 cells with a polynomial degree of $N = 7$ results in a typical large eddy setup of 32 degrees of freedom (DOF) per direction.

5.5.3 Compiler Options

Depending on the dealiasing strategy used, FLEXI should be compiled either with the `tgv_overintegration`, `tgv_split_lobatto` or `tgv_split_gauss` preset using the following commands

```
cmake -B build --preset tgv_overintegration
cmake --build build
```

```
cmake -B build --preset tgv_split_lobatto
cmake --build build
```

or

```
cmake -B build --preset tgv_split_gauss
cmake --build build
```

respectively.

5.5.4 Simulation Parameters

The parameter file to run the simulation is supplied as `parameter_flexi.ini`.

Interpolation

```
! ============================================================= !
! INTERPOLATION
! ============================================================= !
N             = 7
```

The parameter `N` sets the degree of the solution polynomial. In this example, the solution is approximated by a polynomial of degree 3 in each spatial direction. This results in $(N+1)^3 = 512$ 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 $N \in [3, .., 9]$.

Overintegration

To apply polynomial dealiasing there are the following options in FLEXI.

```
! ============================================================= !
! OVERINTEGRATION (ADVECTION PART ONLY)
! ============================================================= !
OverintegrationType = 0  ! 0:off
                         ! 1:cut-off filter
                         ! 2: conservative cut-off

NUnder              = 7  ! specifies effective polydeg
```

(continues on next page)

(continued from previous page)

```
                              ! (modes > NUnder are thrown away)
                              ! only for types 1 and 2
```

In mode 0, polynomial dealiasing is disabled. FLEXI has two ways of doing polynomial dealiasing. In mode 1, a filter is applied to the time-update $\mathcal{J}U_t$. The filter is formulated as a Galerkin projection of degree N to `NUnder`, the effective resolution is thus `NUnder`. Mode 2 is in principle identical to mode 1, but takes into account non-linear metric terms present in curved meshes. For the linear mesh in this tutorial, the result is identical. Since mode 2 is slightly more computational expensive, we omit it in the present tutorial.

Kinetic/Entropy Stable Formulations

An additional dealiasing technique is provided by entropy/kinetic energy stable split formulations of the DGSEM. In FLEXI, implementations either on `Legendre-Gauss-Lobatto` or on `Legendre-Gauss` integration points are available and depending on the chosen compile option. The respective split flux formulation can be specified by the following option. The most commonly used are `PI`, `CH` and `SD`. While the option `PI` [14] enables the use of a kinetic energy stable formulation, the option `CH` [15] provides an entropy conservative formulation of the DGSEM. Finally, the parameter choice `SD` yields a flux differencing form of the DGSEM which is equivalent to the standard DGSEM formulation.

```
! ============================================================= !
! SPLIT DG
! ============================================================= !
SplitDG             = PI ! PI: kinetic energy preserving formulation
                         ! CH: entropy conserving formulation
                         ! SD: standard DGSEM in flux differencing formulation
```

Riemann Solvers

Besides the inherent filtering properties of the DG operator, the only additional 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:

```
! ============================================================= !
! Riemann
! ============================================================= !
Riemann             =  RoeEntropyFix ! Riemann solver to be used:
                                     ! LF, HLLC, Roe,
                                     ! RoeEntropyFix, HLL, HLLE, HLLEM
```

Attention: Be aware that from the above listed Riemann solvers, only the implementations of `LF`, `Roe` and `RoeEntropyFix` are compatible for the use with split flux formulations.

Sub-Grid Scale Model

To add sub-grid scale (SGS) model by Smagorinsky, set the parameter `eddyViscType = 1`. Here, `CS` is the Smagorinsky constant which is usually chosen around `CS = 0.1` for isotropic turbulence (such as in the TGV).

```
! ============================================================= !
! LES MODEL
! ============================================================= !
eddyViscType        = 0   ! Choose LES model, 1:Smagorinsky
CS                  = 0.1 ! Smagorinsky constant
PrSGS               = 0.6 ! turbulent Prandtl number
```

5.5.5 Simulation and Results

We proceed by running the code with the following command.

```
flexi parameter_flexi.ini
```

If **FLEXI** was compiled with MPI support, it can also be run in parallel with the following command. Here, <NUM_PROCS> is an integer denoting the number of processes to be used in parallel.

```
mpirun -np <NUM_PROCS> flexi parameter_flexi.ini
```

Important: **FLEXI** uses an element-based domain decomposition approach for parallelization. Consequently, the minimum load per process is *one* grid element, i.e. do not use more processes than grid elements!

This test case generates an analysis output file named <PROJECTNAME>_TGVAnalysis.csv, which we use to examine the results. Instead of focusing on flow visualization, this tutorial centers on analyzing key quantities directly from this analysis output data. Among other interesting quantities, the analysis file contains the incompressible dissipation rate, stored in the second column of the file. 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. We will use this quantity in the tutorial to verify your results. You can visualize the `csv` file in your favored plotting tool, e.g., within ParaView using the option `Line Chart View`.

$$\mathrm{DR}_S = \frac{2\mu}{\rho_0 \|\Omega\|} \int_\Omega S_{ij} S_{ij} \, d\mathbf{x}$$

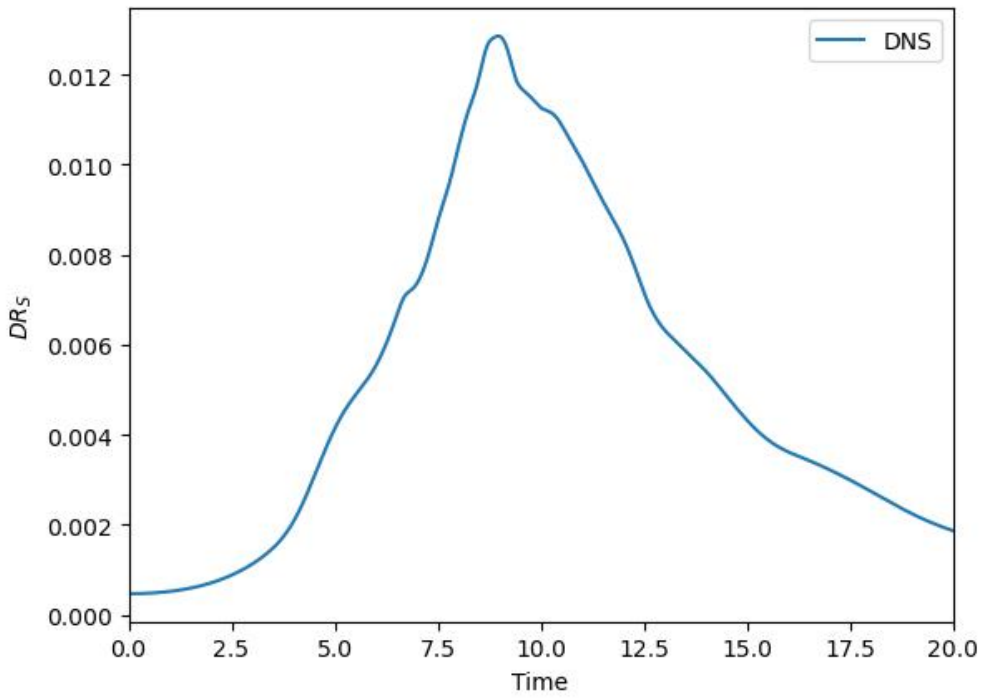

Fig. 5.16: Incompressible dissipation rate of the Taylor–Green vortex over time.

Part I: Crashing Simulation

First, we run FLEXI without any kind of dealiasing technique. For this, use the FLEXI version compiled with the preset `tgv_overintegration`. We will find that the code crashes, once scale production becomes relevant. The same holds for the split form DGSEM if used with the `SD` split flux and the preset `tgv_split_lobatto` or `tgv_split_lobatto`- You can compare your result to the `crash_no_dealiasing.csv` file in the tutorial folder.

Part II: Overintegration

We now use overintegration by changing the respective settings in the `parameter_flexi.ini` file as described above. Set `OverintegrationType = 1` and specify `N = 11` and `NUnder = 7`. You can compare your result to the `les_overintegration.csv` file in the tutorial folder.

Part III: Split Formulation

We now use the split DGSEM formulation as a dealiasing technique. Please be aware to use FLEXI either compiled with the preset `tgv_split_lobatto` or `tgv_split_gauss`. Set `SplitDG = PI` and specify `N = 7`. Don't forget to switch off overintegration. You can compare your result to the `les_split.csv` file in the tutorial folder.

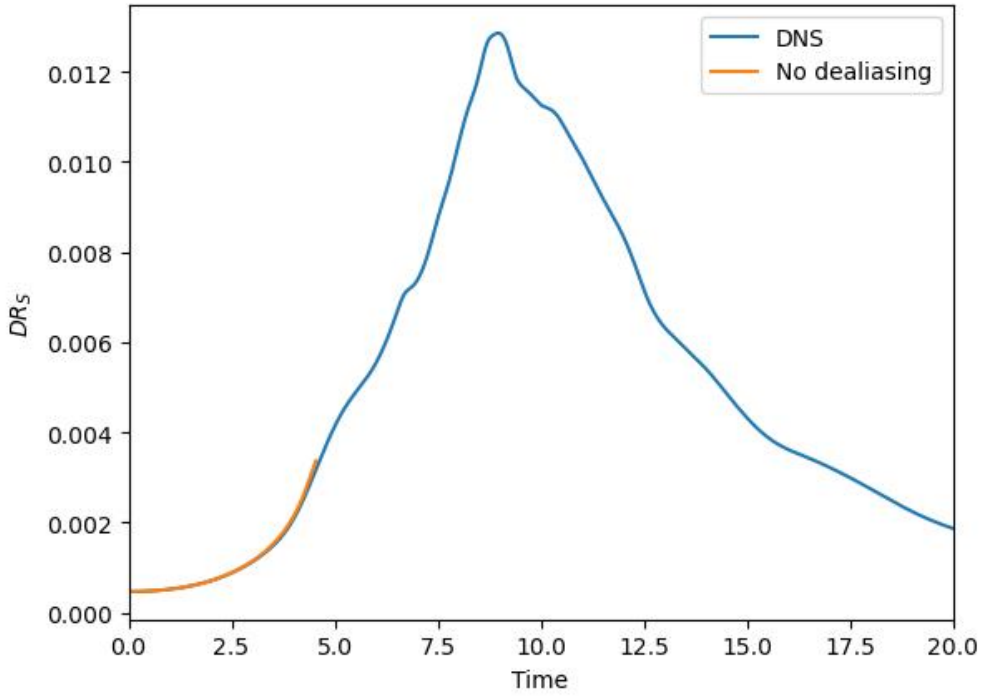

Fig. 5.17: Incompressible dissipation rate of the Taylor–Green vortex over time without any dealiasing technique.

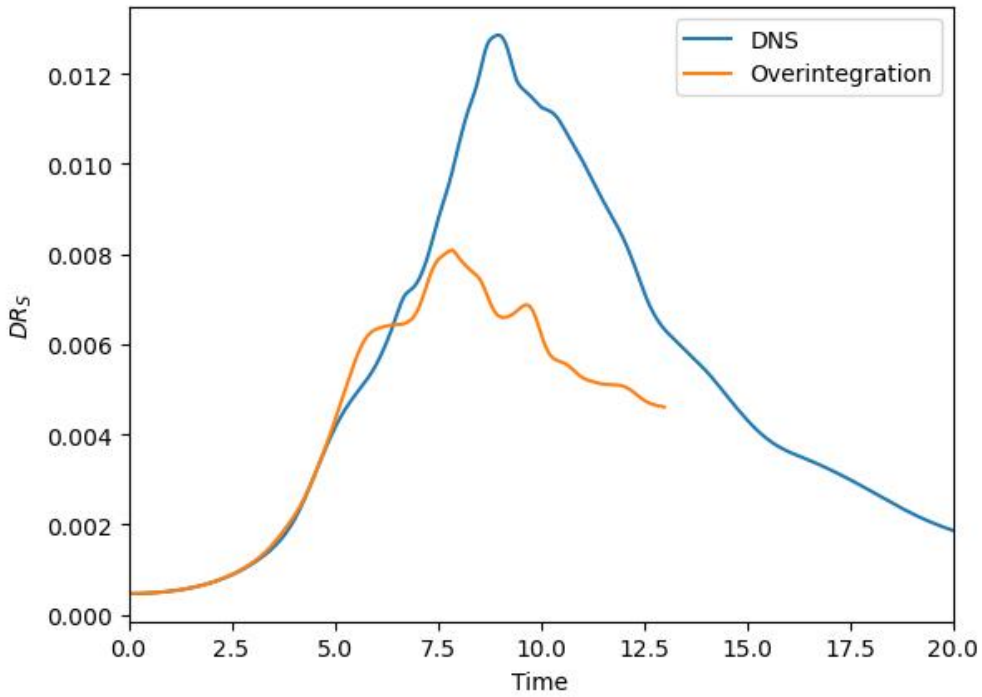

Fig. 5.18: Incompressible dissipation rate of the Taylor–Green vortex over time with overintegration.

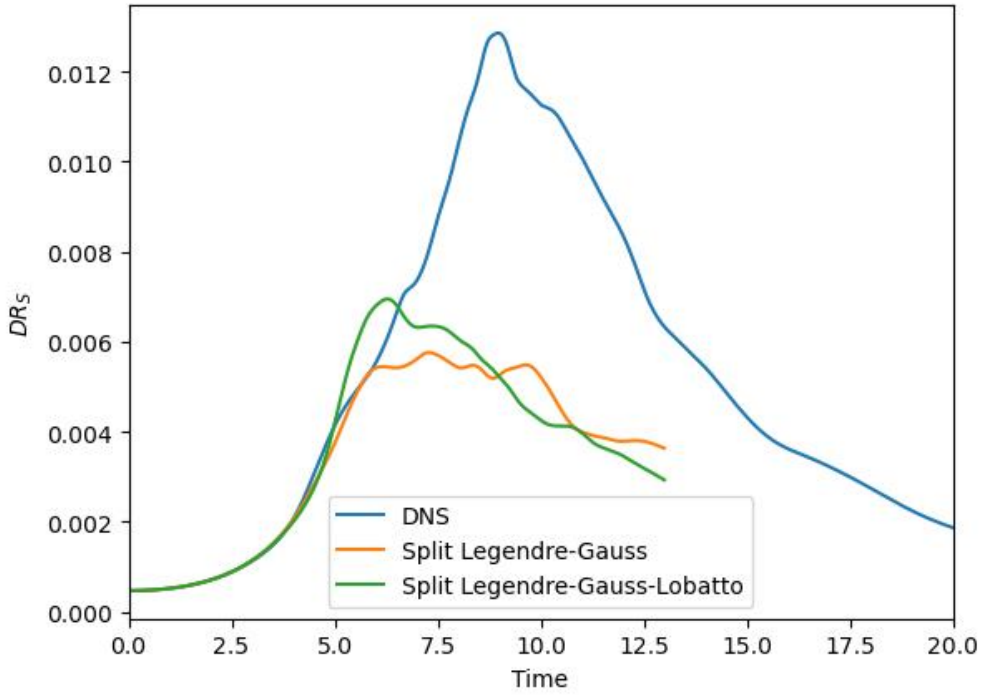

Fig. 5.19: Incompressible dissipation rate of the Taylor–Green vortex over time with a kinetic energy stable formulation.

Part VI: Explicit LES model

To see the effect of adding explicit eddy viscosity, we activate the LES model (Smagorinsky) as described above via `eddyViscType = 1`. To obtain the reference result in `les_smago.csv`, set `CS = 0.1`. Don't forget to switch back to the compiler preset `tgv_overintegration` and deactivate overintegration `OverintegrationType = 0`. Use a polynomial degree of `N=7`.

Part V: Have Fun!

Feel free to play around with the effect of the constant in the Smagorinsky model or compare the different dealiasing techniques with respect to accuracy or compute time. Most important, have fun with FLEXI.

5.6 SOD Shock Tube

The Sod shock tube example [22] is one of the most basic test cases to investigate the shock capturing capabilities of a CFD code. An initial discontinuity is located in the middle of the one dimensional domain $x \in [0, 1]$. The left and right states are given by $\rho = 1, v = 0, p = 1$ and $\rho = 0.125, v = 0, p = 0.1$. The tutorial is located at `tutorials/sod`. These states are already set as `RefState` in the `parameter_flexi.ini` file.

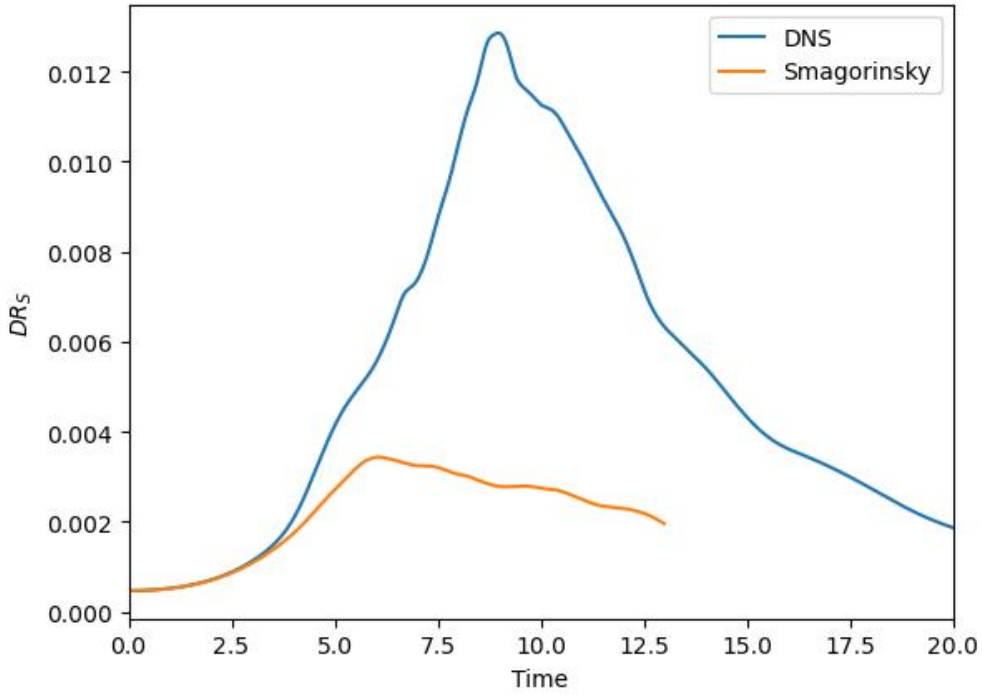

Fig. 5.20: Incompressible dissipation rate of the Taylor–Green vortex over time with the Smagorinsky model as a sub grid scale model.

5.6.1 Mesh Generation

In the tutorial directory, we provide the necessary mesh files, along with a parameter files for **HOPR** to generate these meshes. You can recreate any mesh by running the following command. A full tutorial on how to run **HOPR** is available at the [HOPR documentation][hopr].

```
hopr parameter_hopr.ini
```

5.6.2 Build Configuration

This example requires the Finite Volume (FV) shock capturing and the Euler equations. In this tutorial, we will investigate the shock capturing based on switching the DG representation to FV sub-cells. Therefore, [FLEXI][flexi] should be compiled either with the `sod` preset using the following command

```
cmake -B build --preset sod
cmake --build build
```

5.6.3 Simulation Parameters

The parameter file to run the simulation is supplied as `parameter_flexi.ini`.

Finite Volume Shock Capturing

The options for the Finite Volume shock capturing are contained in the 'FV-Subcell' section.

```
! ============================================================= !
! FV-Subcell
! ============================================================= !
IndicatorType        = Persson
IndVar               = 1       ! first conservative (density)
                               ! used for indicator evaluation
IndStartTime         = 0.001   ! until this time FV is used in
                               ! the whole domain

FV_LimiterType       = MinMod
FV_IndUpperThreshold = -3.     ! upper threshold (if IndValue
                               ! above this value, switch to FV)
FV_IndLowerThreshold = -4.     ! lower threshold (if IndValue
                               ! below this value, switch to DG)
```

The `IndicatorType` parameter sets the type of indicator function used to detect DG elements containing discontinuities. For this case, the Persson indicator [23] is applied, an element-local indicator that compares the different modes of the DG polynomial. If the relative content in the highest mode is high compared to the amount in the lower modes, the DG polynomial may show oscillatory behavior. All indicator functions return a high value for "troubled" with discontinuities and low values for smooth elements. The variable `IndVar` specifies the index within the conservative variable vector used to evaluate the indicator function. Typically, pressure (index 6) is a good choice; however, for this test case, density is used instead. `IndStartTime` specifies a time during which the actual indicator function is overwritten by a high value to force the use of FV elements everywhere. This helps capture initial discontinuities placed exactly at element boundaries, as element-local indicators like Persson can not detect these discontinuities. `FV_LimiterType` sets the limiter used in the second-order FV reconstruction. `FV_IndUpperThreshold` and `FV_IndLowerThreshold` define thresholds for deciding in which elements are represented by the DG method and where the FV sub-cell scheme should be used. While a single threshold can theoretically suffice, it often leads to continuous switching between the DG and the FV schemes. To avoid this, switching to FV only occurs if the indicator exceeds the upper threshold. Switching back to DG only happens if the indicator value falls below the lower threshold.

5.6.4 Simulation and Results

We proceed by running the code with the following command.

```
flexi parameter_flexi.ini
```

This test case generates 5 state files name `sod_State_>TIMESTAMP>.h5` for $t = 0.0, 0.05, 0.10, 0.15, 0.20$.

Visualization

FLEXI relies on ParaView for visualization. In order to visualize the **FLEXI** solution, its format has to be converted from the HDF5 format into another format suitable for **Paraview**. **FLEXI** provides a post-processing tool *posti_visu* which generates files in VTK format with the following command.

```
posti_visu parameter_postiVisu.ini parameter_flexi.ini sod_State_0*
```

posti_visu generates two types of files which can be loaded into **ParaView**. *vtu* files contain either DG or the FV part of the solution. The *vtm*-files combine the DG and FV *vtu*-file of every timestamp. It is thus recommended to load the *vtm*-files into **ParaView**.

For this one-dimensional test case, apply the `Plot Over Line` filter in **ParaView**. When setting up the filter, choose the `X Axis` as the line direction to create a line plot of the variable values along this axis. The resulting plot should resemble the one shown in Fig. 5.21.

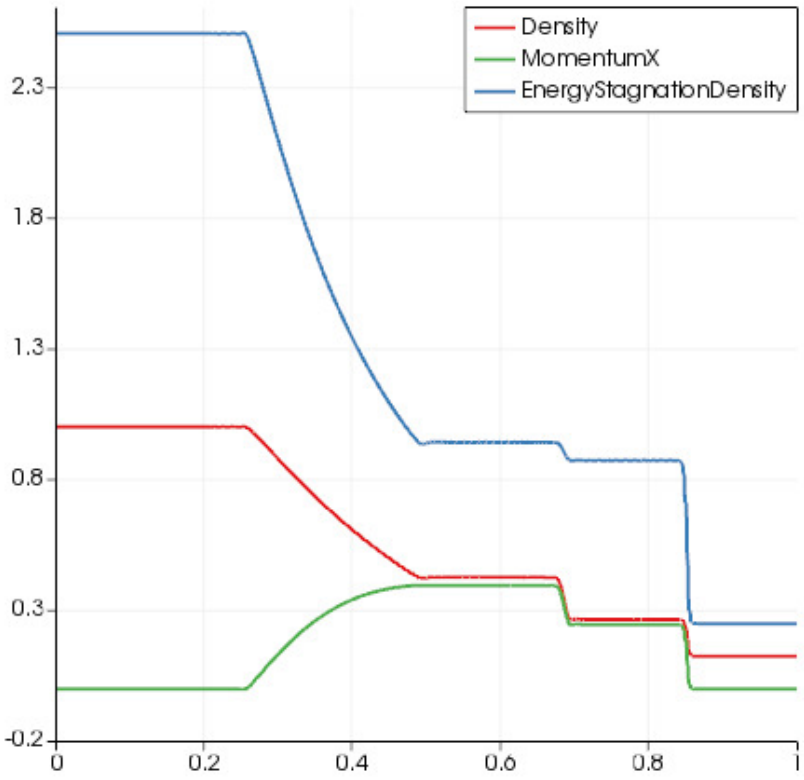

Fig. 5.21: Solution of the Sod shock tube at $t = 0.2$.

5.7 Double Mach Reflection

The Double Mach Reflection is a classical test case to investigate the abilities of a numerical scheme to represent shocks and contact discontinuities. It was proposed by Colella and Woodward [24] and considers a Mach 10 oblique shock wave that hits a reflecting wall. The initial conditions are given by the Rankine-Hugoniot conditions

$$(\rho, v_1, v_2, p) = \begin{cases} (8.0, 8.25 \cdot \cos(30°), -8.25 \cdot \sin(30°), 116.5) & x < x_0 + \sqrt{\frac{1}{3}}y \\ (1.4, 0.0, 0.0, 1.0) & x \geq x_0 + \sqrt{\frac{1}{3}}y \end{cases}, \tag{5.2}$$

where the wall at the bottom starts at $x_0 = \frac{1}{6}$ and the computational domain $\Omega = [0,4] \times [0,1]$ is discretized by an equidistant Cartesian mesh. This tutorial is located in the folder `tutorials/dmr`.

5.7.1 Mesh Generation

In the tutorial directory, we provide the necessary mesh files, along with a parameter files for **HOPR** to generate these meshes. You can recreate any mesh by running the following command. A full tutorial on how to run **HOPR** is available at the [HOPR documentation][hopr].

```
hopr parameter_hopr.ini
```

5.7.2 Flow Simulation

This example requires the finite volume (FV) shock capturing. Two variants of the shock capturing are implemented in **FLEXI**, which are both based on the FV sub-cell approach. This approach subdivides each DG element into FV sub-cells, where each cell corresponds to one degree of freedom of the initial DG element. Thus, the total number of degrees of freedom is constant throughout the simulation, while the DG and the FV operator are chosen independently for each individual element.

The first variant of the FV sub-cell approach **switches** DG elements into the FV sub-cell representation such that the element-local solution is either a smooth polynomial (DG representation) or piecewise constant (FV sub-cell), as detailed in [3]. The second variant of the FV sub-cell approach **blends** the DG operator with the FV operator, while the solution itself always remains a DG polynomial, see [4] for more details. In the following, both approaches will be applied to simulate the Double Mach Reflection.

Finite Volume Switching

In the switching-based shock capturing, the element-local solution is either given in DG representation (`FV_Elems=0`) or interpolated to piecewise constant FV sub-cells (`FV_Elems=1`).

Build Configuration

The FV switching is enabled through the build option `FLEXI_FV=SWITCH`, the corresponding CMake preset `cmd_fvswitch` is applied by running

```
cmake -B build --preset dmr_fvswitch
cmake --build build
```

Simulation Parameters

The simulation setup is defined in `parameter_flexi_switch.ini` and includes parameters for the shock capturing via switching.

```
! ============================================================= !
! FV-Subcell
! ============================================================= !
IndicatorType        = Jameson
```

(continues on next page)

(continued from previous page)

```
IndVar               = 6       ! sixth variable (pressure)
                               ! used for indicator evaluation
FV_LimiterType       = 1       ! MinMod
FV_IndUpperThreshold = 0.010   ! upper threshold (if IndValue
                               ! above this value, switch to FV)
FV_IndLowerThreshold = 0.005   ! lower threshold (if IndValue
                               ! below this value, switch to DG)
FV_toDG_indicator    = T
FV_toDG_limit        = -5.5
FV_IniSupersample    = T
```

The `IndicatorType` parameter sets the type of indicator function used to detect DG elements containing discontinuities. For this case, the Jameson indicator [23] is applied, an adaptation of the switching function of the Jameson-Schmidt-Turkel scheme [25] to FV sub-cells. In contrast to the Persson indicator, it is not element-local and therefore more robust for traveling discontinuities. All indicator functions return a high value for "troubled" with discontinuities and low values for smooth elements. The variable `IndVar` specifies the index within the variable vector used to evaluate the indicator function. Typically, pressure (index 6) is a good choice and also used here. `FV_toDG_indicator` enables an additional Persson indicator [23] for the switch from FV to DG. When an FV element is marked for transition to DG, it is temporarily converted to a DG element and the Persson indicator is evaluated for this DG polynomial to test if the polynomial is oscillating. Only in the case of a non-oscillatory solution, the solution is converted to DG. Otherwise, the element retains the FV representation. This improves the simulation stability when indicator functions defined on the FV sub-cells are used. In the given case, the Jameson indicator only considers oscillations between adjacent degrees of freedom (DOFs), which may miss certain high-frequency oscillations in the polynomial. The `FV_toDG_limit` parameter is then used as threshold for additional Persson indicator, restricting the FV to DG transition to indicator values below this choice. During initialization, by default the solution is initialized as DG polynomials for all elements. In a second, step, the indicator function is evaluated to identify troubled elements which are then converted to an FV representation. This can cause issues if discontinuities lie inside DG elements which leads to strongly oscillating polynomials and invalid solutions, e.g., negative density, even after converting these oscillating polynomials to FV. The `FV_IniSupersample` option 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 $(N+1)$ equidistant points per dimension inside the sub-cell and then taking the arithmetic mean value.

Simulation and Results

We proceed by running the code with the following command.

```
flexi parameter_flexi_switch.ini
```

This test case generates 11 state files name `dmr_SWITCH_State_<TIMESTAMP>.h5` for $t = 0.0, 0.02, \ldots, 0.20$.

Visualization

FLEXI relies on ParaView for visualization. In order to visualize the **FLEXI** solution, its format has to be converted from the HDF5 format into another format suitable for **Paraview**. **FLEXI** provides a post-processing tool *posti_visu* which generates files in VTK format with the following command.

```
posti_visu parameter_postiVisu.ini parameter_flexi_switch.ini dmr_SWITCH_
↪State_0000000.*
```

posti_visu generates two types of files which can be loaded into **ParaView**. *vtu* files contain either DG or the FV part of the solution. The *vtm*-files combine the DG and FV *vtu*-file of every timestamp. It is thus recommended to load the *vtm*-files into **ParaView**. The result at $t = 0.2$ should look like in figure Fig. 5.22.

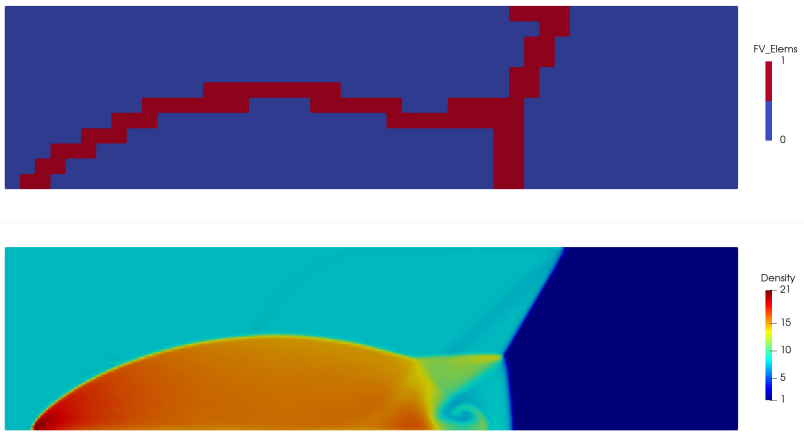

Fig. 5.22: Distribution of DG and FV elements (top) and density (bottom) of Double Mach Reflection at $t = 0.2$.

Finite Volume Blending

Next, we will investigate the blending approach. In this tutorial we use an entropy-stable split formulation to ensure the stability of the FV blending approach, with more details on the split formulation given later in section *Plane Turbulent Channel Flow*. For the FV sub-cell blending, the elements are not switched completely to the FV operator, but instead the DG operator R_{DG} and FV operator R_{FV} are blended as

$$R = \alpha R_{FV} + (1 - \alpha) R_{DG}$$

with the blending coefficient α. Instead of switching between a DG and a FV discretization, the blending allows a continuous transition between the DG and FV operators. The blending factor is computed based on the indicator proposed by [4], which is parameter-free and does not require any parameters to be tuned by the user.

Build Configuration

The FV blending is enabled through the build option `FLEXI_FV=BLEND`. The FV blending requires selecting the Gauss-Lobatto node set by setting `FLEXI_NODETYPE=GAUSS-LOBATTO` and to enable the split-form DG with `FLEXI_SPLIT_DG=ON`. **FLEXI** should be compiled using the `dmr_fvblend` present.

```
cmake -B build --preset dmr_fvblend
cmake --build build
```

Simulation Parameters

The simulation setup is defined in `parameter_flexi_blend.ini` and the simulation parameters specific to the FV blending approach are summarized below.

```
! ============================================================= !
! FV-Subcell
! ============================================================= !
IndicatorType         = Jameson
IndVar                = 6       ! sixth variable (pressure)
                                ! used for indicator evaluation
FV_LimiterType        = 1       ! MinMod
FV_alpha_min          = 0.01    ! Lower bound for alpha (all
                                ! elements below threshold are
                                ! treated as pure DG)
FV_alpha_max          =  0.5    ! Maximum value for alpha. All
                                ! blending coefficients exceeding
                                !  this value are clipped
FV_alpha_ExtScale     =  0.5    ! Scaling factor by which the
                                ! blending factor is scaled when
                                ! propagated to its neighbor.
                                ! Has to be between 0 and 1.
FV_nExtendAlpha       =    1    ! Number of times this propagation
                                ! of the blending should be performed
                                ! (number of element layers).
                                ! Higher values correspond to a wider
                                ! sphere of influence.
FV_doExtendAlpha      =    T    ! Blending factor is prolongated
                                ! into neighboring elements
```

Simulation and Results

Similarly to the switching-based approach, we proceed by running the code with the following command.

```
flexi parameter_flexi_blend.ini
```

This test case generates 11 state files name `dmr_BLEND_State_<TIMESTAMP>.h5` for $t = 0.0, 0.02, \ldots, 0.20$.

Visualization

FLEXI relies on ParaView for visualization. In order to visualize the **FLEXI** solution, its format has to be converted from the HDF5 format into another format suitable for **Paraview**. **FLEXI** provides a post-processing tool *posti_visu* which generates files in VTK format with the following command.

```
posti_visu parameter_postiVisu.ini parameter_flexi_blend.ini dmr_BLEND_State_
↪0000000.*
```

As the solution is always represented by DG polynomials. *posti_visu* generated only the DG part of the solution. Thus, load the generated *vtm*-files into **ParaView**. The result at $t = 0.2$ should look like in figure Fig. 5.23.

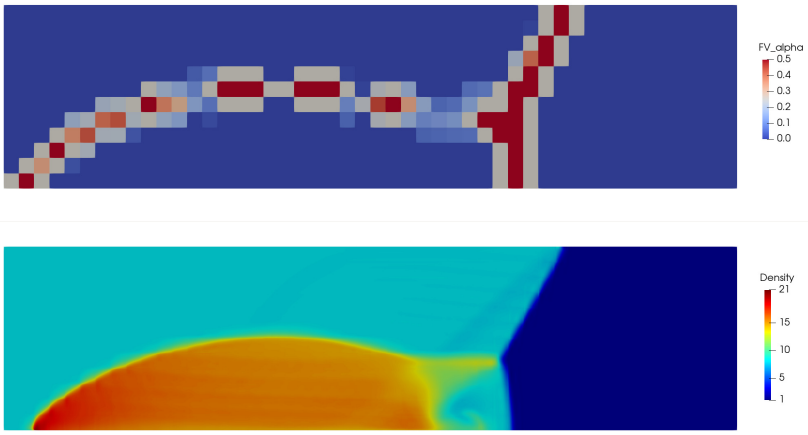

Fig. 5.23: Distribution of the blending factor α between the DG and FV operators (top) and density (bottom) of Double Mach Reflection at $t = 0.2$.

5.8 Plane Turbulent Channel Flow

This tutorial describes how to set up and run the test case of a turbulent flow in a plane channel geometry. We will learn how to use the split-form DG method to guarantee non-linear stability of the turbulent flow simulation. In a second step, we add the sub-grid scale (SGS) model of Smagorinsky combined with Van Driest type damping to run stable wall-bounded turbulent flows with explicit small scale dissipation. This tutorial is located in the folder `tutorials/channel`.

5.8.1 Flow description

The flow is calculated in a plane channel with half-height $\delta = 1$, streamwise (x-coordinate) length 2π and span (z-coordinate) width π 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 a constant density of $\rho = 1$ is used. We superimpose sinus perturbations in the u, v and w velocity components which lead to rapid production of turbulent flow structures. Since the wall friction would slow down the flow over time, a source term imposing a constant 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 $Ma = 0.1$, matching the

Moser channel test case [26]. In this setting, the wall friction velocity τ will always be equal to 1. We can define a Reynolds number based on the channel half-height and the wall friction velocity as $Re_\tau = 1/\nu$.

5.8.2 Build Configuration

FLEXI should be compiled with the `channel` preset using the following commands.

```
cmake -B build --preset channel
cmake --build build
```

5.8.3 Mesh Generation

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 as `parameter_hopr.ini`. The default mesh uses 4 cells with a polynomial degree of $N = 5$, corresponding to a Large Eddy Simulation (LES) setup of 24 DOFs per direction.

5.8.4 Simulation Parameters

The simulation setup is defined in `parameter_flexi.ini`. In this tutorial, we are not interested in the flow visualization of the instantaneous state files. Instead, we 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.

Interpolation / Discretization Parameters

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

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, the `FLEXI_SPLIT_DG` option has to be switched `ON` in combination with the `FLEXI_NODETYPE=GAUSS-LOBATTO`. **FLEXI** provides several distinct split-flux formulations. Therefore, a specific split flux formulation has to be chosen during runtime. In this tutorial, the pre-defined split-flux formulation by Pirozzoli [14] is used, which results in a kinetic energy preserving DG scheme.

Sub-Grid Scale Modeling

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

The `eddyViscType` defines the SGS model in use, with 0 corresponding to no model (implicit LES) and 1 the model by Smagorinsky [27]. The Smagorinsky constant `CS` is 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 behavior of the eddy viscosity when approaching a wall, Van Driest-type damping has to be switched on.

5.8.5 Simulation and Results

We proceed by running the code with the following command.

```
flexi parameter_flexi.ini
```

If **FLEXI** was compiled with MPI support, it can also be run in parallel with the following command. Here, <NUM_PROCS> is an integer denoting the number of processes to be used in parallel.

```
mpirun -np <NUM_PROCS> flexi parameter_flexi.ini
```

Important: **FLEXI** uses an element-based domain decomposition approach for parallelization. Consequently, the minimum load per process is *one* grid element, i.e. do not use more processes than grid elements!

Once the simulation has completed, the generated state files can be post-processed via the `posti_channel_fft` tool which was build by the `POSTI_CHANNEL_FFT` CMake option. To run the post-processing, the standard command is

```
posti_channel_fft parameter_channel_fft.ini <State1 State2 ...>
```

The `parameter_channel_fft.ini` is provided in the tutorial folder. The selection of the specific *State* files to be used is left to 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 files. One contains the mean velocity profiles as well as the Reynolds stress profiles while the other two files contain turbulent energy spectra. To visualize those files, you can run the python script `plotChannelFFT.py`, provided in the `tools/testcases` folder with the following command in your simulation directory.

```
python tools/testcases/plotChannelFFT.py -p <PROJECTNAME> -t <POSTITIME>
```

Here, <PROJECTNAME> specifies the project name specified in the `parameter_flexi.ini` file and <POSTITIME> is the timestamp of your output files from the `posti_channel_fft` tool.

Part I: Split-DG without Explicit LES Model

First, we run **FLEXI** without an SGS model. This configuration is called implicitly modeled 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 center of the channel are given in Fig. 5.24.

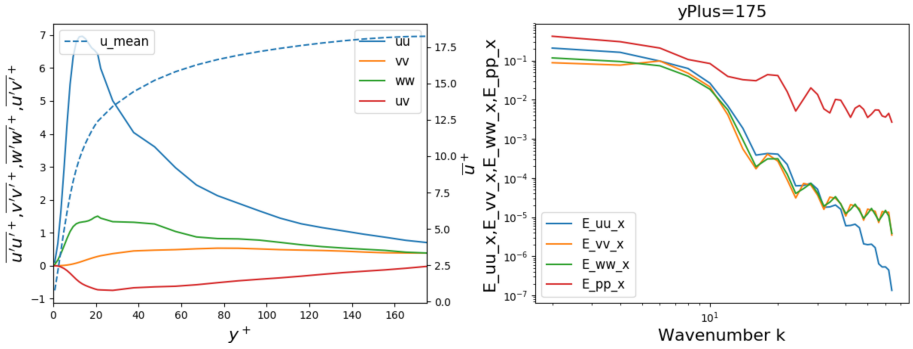

Fig. 5.24: Mean velocity and Reynolds stress profiles (left) as well as turbulent energy spectra close to the center of the channel (right) of an implicit LES at $Re_\tau = 180$.

Part II: SplitDG with Explicit LES Model

In a second step, we run **FLEXI** with the SGS model by Smagorinsky and Van Driest damping. These options need to be enabled in the parameter file as described above. The resulting mean velocity and Reynolds stress profiles as well as turbulent energy spectra close to the center of the channel are given in Fig. 5.25. In comparison to the previous simulation, you might recognize the effect of the explicit damping on the Reynolds stress profile $\overline{u'u'}$, most evident close to the maximum. To further study the influence of Smagorinsky's model, play around with the spatial resolution both in terms of grid resolution and 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. [26].

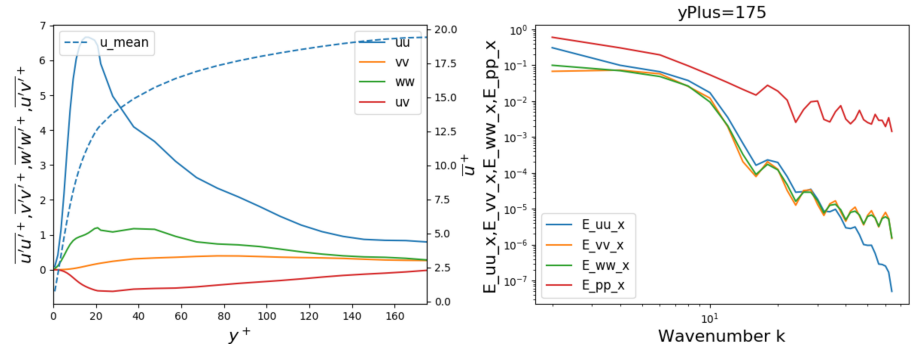

Fig. 5.25: Mean velocity and Reynolds stress profiles (left) as well as turbulent energy spectra close to the centre of the channel (right) of a LES with Smagorinsky's model and van Driest damping at $Re_\tau = 180$.

5.8.6 Performance Improvements

FLEXI comes with some advanced optimizations in order to increase its computational efficiency for compute-intensive simulations. As these optimizations require user intervention, they are disabled by default and appear once the CMake flag `FLEXI_PERFORMANCE=ON` is set. The first option `FLEXI_PERFORMANCE_OPTLIFT` optimizes the computation of the parabolic terms of the applied equation system by omitting terms not relevant for the lifting procedure. However, POSTI is not available if this option is enabled.

Link-Time Optimization

The option `FLEXI_PERFORMANCE_PGO=ON` enable link-time optimization (LTO), sometimes called profile-guided optimization (PGO). For LTO/PGO, the executable is first instrumented with profiling tools by the compiler and then executed on a relatively simple test case. The generated profiling data can be used by the compiler to identify bottlenecks and hotspots in the code that cannot be identified from the static source code analysis. Consequently, the executable is compiled a second time using the gathered profiling data to perform these additional optimizations. In **FLEXI**, this two-step compilation works as follows. First, **FLEXI** is compiled with the following options.

```
cmake -B build -DFLEXI_PERFORMANCE=ON -DFLEXI_PERFORMANCE_OPTLIFT=ON -DFLEXI_
↪PERFORMANCE_PGO=ON -DCMAKE_BUILD_TYPE=Profile
cmake --build build
```

For this first step, **FLEXI** is compiled with `CMAKE_BUILD_TYPE=Profile` build type in order to activate the profiling. Then, **FLEXI** has to be executed on a simple test case. Here, the freestream tutorial Section 5.2 provides a good starting step. Finally, **FLEXI** is compiled a second time, but this time with the build type set to `CMAKE_BUILD_TYPE=Release`.

```
cmake -B build -DFLEXI_PERFORMANCE=ON -DFLEXI_PERFORMANCE_OPTLIFT=ON -DFLEXI_
↪PERFORMANCE_PGO=ON -DCMAKE_BUILD_TYPE=Release
cmake --build build
```

This setting incorporates the generated profiling data into the compilation process. Now, **FLEXI** can be executed as usual and should show a considerable performance improvement in comparison to the previous simulations.

Important: Link-Time Optimization (LTO)/Profile-Guided Optimization (PGO) is currently only supported for the GNU compiler. Furthermore, this two-step compilation process has to be performed each time either the code or the compile options, i.e. the **FLEXI** executable, are changed.

5.9 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 temporal probes, here called "record points", together with the associated **posti_visualizerecordpoints** tool. Furthermore, we introduce the **posti_dmd** tool and use it for dynamic mode decomposition (DMD). This tutorial is located in the folder `tutorials/cylinder`.

5.9.1 Flow Description

The setup considered consists of a 2D rectangular domain with the primary flow in x-direction, from left to right. Being a 2D plane, it corresponds to the "look-down" view upon the domain and the cylinder. The imposed flow is sufficiently fast for the wake of the cylinder to turn from a laminar flow to a street of shed vortices. It is the goal of the dynamic mode decomposition to analyze the primary oscillation frequencies present in these shed vortices.

5.9.2 Build Configuration

FLEXI should be compiled with the `cylinder` preset using the following commands.

```
cmake -B build --preset cylinder
cmake --build build
```

5.9.3 Mesh Generation

We use a curved mesh with 1000 cells for the tutorial. The mesh is deformed to align with the cylinder boundary in the center. Furthermore, we utilize stretching in wall-normal direction to increase the resolution in the crucial near-wall region and relax it towards the domain boundaries. In case you want to generate other meshes, the parameter file for **HOPR** is included in the tutorial directory as `parameter_hopr.ini`.

5.9.4 Simulation Parameters

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

Material Properties

The material properties of the considered fluid are defined in the equation of state section. We chose the Prandtl number Pr and the isentropic coefficient κ to their default values of $Pr = 0.72$ and $\kappa = 1.4$, respectively. Thus, they are omitted in the parameter file.

```
! ============================================================ !
! Equation of State
! ============================================================ !
```

(continues on next page)

(continued from previous page)

```
R                = 17.857   ! ideal gas constant
Mu0              = 0.005    ! dynamic viscosity
```

Based on the ideal gas law, we get

$$Ma = 1/\sqrt{\kappa p/\rho} = 0.2$$

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

$$\mu = \rho u D/Re = 1/Re = 0.005$$

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.

5.9.5 Simulation and Results

We proceed by running the code with the following command.

```
flexi parameter_flexi.ini
```

If **FLEXI** was compiled with MPI support, it can also be run in parallel with the following command. Here, <NUM_PROCS> is an integer denoting the number of processes to be used in parallel.

```
mpirun -np <NUM_PROCS> flexi parameter_flexi.ini
```

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

Evaluation of the 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 the following.

```
CalcBodyForces   = T        ! Calculate body forces (BC 4/9)
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 file. In Fig. 5.26 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.

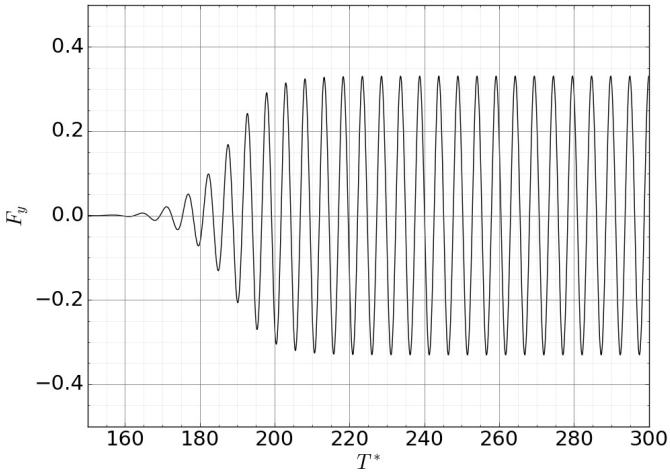

Fig. 5.26: Resulting lift forces on the cylinder.

Evaluation of the Separation Angle

The mean separation angle is evaluated using the record point tool as described in *Recordpoints*. The simulation setup already contains the record points set and output of the record points during the simulation is enabled by the default parameter file. The record points set contains probes distributed along a plane within the boundary layer of the upper cylinder side. For the calculation of the separation angle, we want to use the **Plane_doBLProps** functionality within the **posti_visualizerecordpoints** tool. In addition to the namesake visualization functionality, this tool has options to analyze the boundary layer properties such as the wall friction to estimate the separation point. The required parameters are already set in the `parameter_visualizeRecordpoints.ini` file. Thus, you can directly invoke the tool by running the following command.

```
posti_visualizerecordpoints parameter_visualizeRecordpoints.ini Cylinder_
↪Re200_RP_*
```

After executing the tool, you will get a file named `Cylinder_RP_BLProps_upperSide_BLPla000001.vts` which can be visualized with **ParaView**. Since the data were are interested in is one-dimensional, you won't be able to see the data in the default render view. Instead, choose the `Plot Over Line` filter and **ParaView** should automatically apply the correct plot range. As separation occurs when the skin friction falls to zero, you can estimate the separation angle to $110°$ by plotting **tau_w** over the circumference and finding the intersection with the $y = 0$ axis.

5.9.6 Dynamic Mode Decomposition

The dynamic mode decomposition (DMD) is an algorithm to divide a temporal series into a set of modes which are associated with a frequency and growth/decay rate. Thus, DMD allows us to determine temporal dependencies such as the Strouhal frequency. The DMD in **FLEXI** is implemented according to Schmid et al. [28] and available through the posti tool **posti_dmd**.

As the DMD operates in the frequency domain, we need a higher temporal resolution of the written state files. Thus, change the time `TEnd` to 310 and `nWriteData` to 1. Then, restart the **FLEXI** simulation from the latest state file with the following command.

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

Once the simulation concludes, execute the DMD on the generated state files. The `parameter_dmd.ini` is pre-configured to perform a DMD on the density, thus we can invoke the following command.

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

Attention: Dynamic Mode Decomposition (DMD) is performed in the frequency domain and requires the complete solution to be loaded into memory. Depending on the available memory, you might have to decrease the number of input state files.

During execution, two additional files `Cylinder_Re200_DMD_0000300.000000000.h5` and `Cylinder_Re200_DMD_Spec_0000300.000000000.dat` are generated. The first file contains a field representation of the different modes, the second file contains the Ritz spectrum of the modes. Thus, the field can be visualized in **ParaView** after conversion to VTK format with the following command.

```
posti_visu parameter_postivisuDMD.ini Cylinder_Re200_DMD_0000300.000000000.h5
```

The new file `Cylinder_Re200_Solution_0000300.000000000.vtu` now contains four modes available for visualization. Fig. 5.27 shows the global, the first, the second, and the third harmonic mode. The first mode is the mode of the considered Strouhal number.

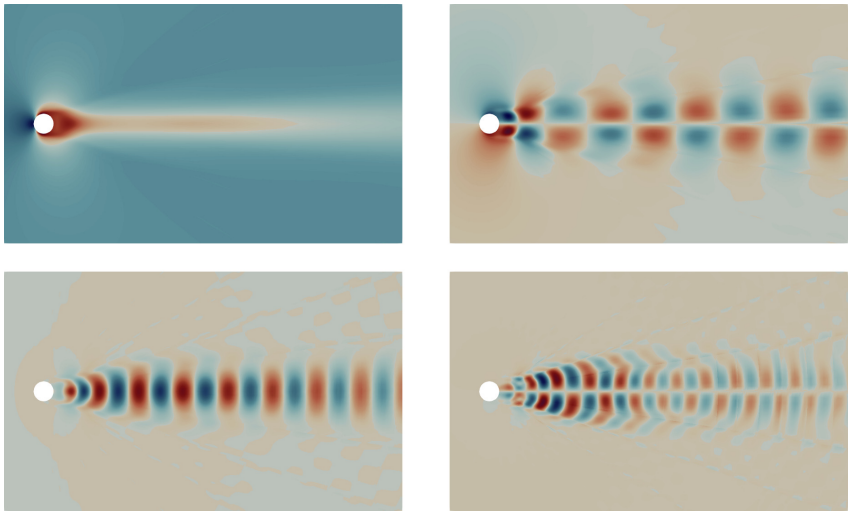

Fig. 5.27: DMD modes of the density field. Top left global mode, top right first harmonic, bottom left second harmonic, bottom right third harmonic.

The Ritz spectrum in DMD is a set of complex numbers that represent the eigenvalues of a low-rank approximation of the Koopman operator. These eigenvalues provide information about the frequencies and growth rates of the dominant modes present in the flow data. **FLEXI** contains a Python script `tools/plot_RitzSpectrum.py` which we can execute on the DMD data file to obtain the Ritz spectrum.

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

The result is a Ritz spectrum as shown in Fig. 5.28. Here, the abscissa shows the frequency of the modes and the ordinate the growth/decay factor. Modes with $\omega_r < 0$ are damped. The modes placed directly on

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

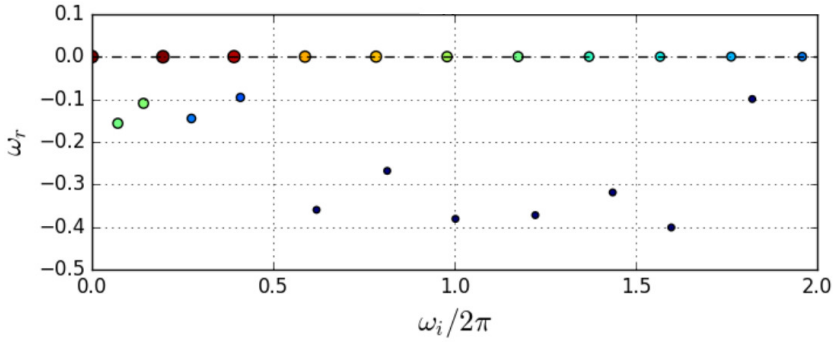

Fig. 5.28: Ritz spectrum.

5.10 Flow Around a NACA0012 Airfoil

In this tutorial, the simulation around a NACA0012 airfoil at $Re = 5000$ and $Ma = 0.4$ is considered. First, we explain how to set the main flow parameters. Next, we 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. The tutorial is located at `tutorials/naca0012`.

5.10.1 Flow Description

A NACA0012 airfoil is placed in a semi-circular domain, extruded in the downstream direction. The chord length is normalized to 1 with an inflow velocity of 1 at an angle of attack (AoA) of $8°$. The viscosity and internal energy are scaled to obtain the desired Reynolds and Mach number, respectively. Fig. 5.29 displays the Mach number distribution along the domain centerline in the vicinity of the airfoil.

5.10.2 Mesh Generation

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. Obtain the executable and run the following command which creates the mesh file `NACA0012_652_Ng2_mesh.h5` in HDF5 format.

```
hopr parameter_hopr.ini
```

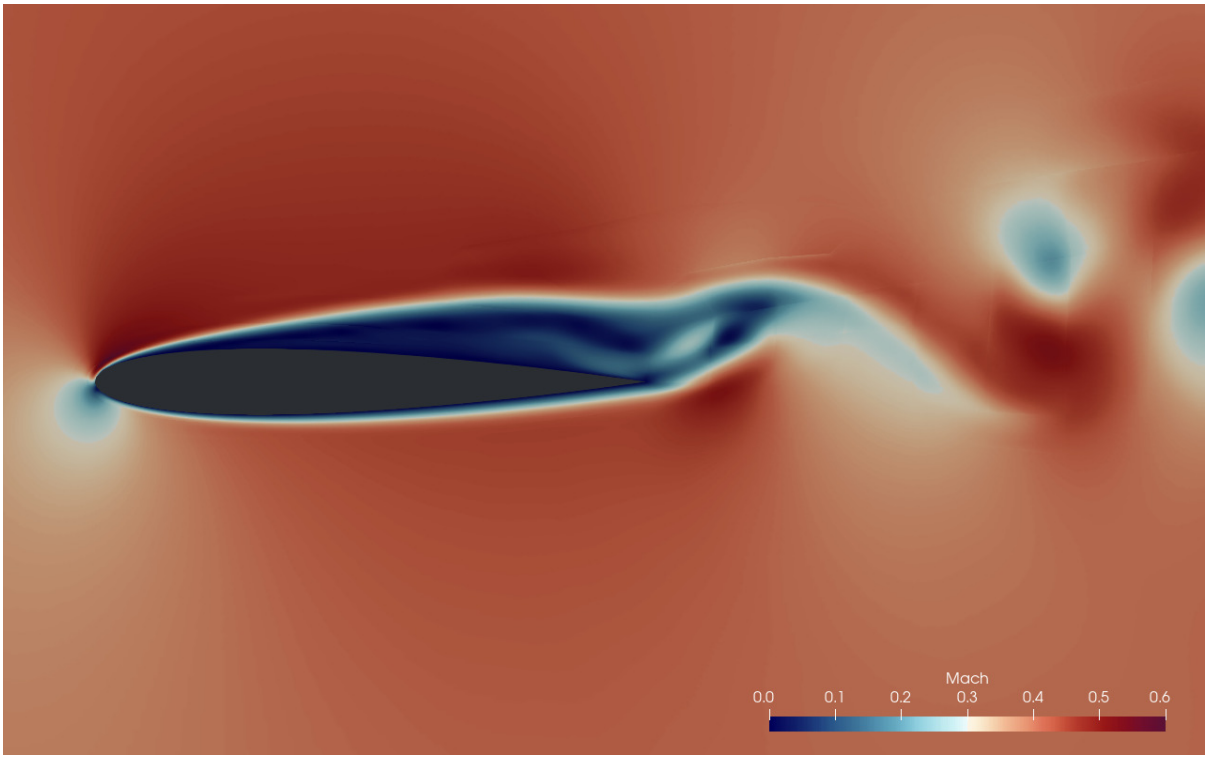

Fig. 5.29: Mach number distribution around the NACA0012 airfoil.

5.10.3 Build Configuration

FLEXI should be compiled with the `naca0012` preset using the following commands.

```
cmake -B build --preset naca0012
cmake --build build
```

5.10.4 Simulation Parameters

The simulation setup is defined in `parameter_flexi.ini`. The initial condition is selected via the variable vector `RefState` which represents the vector of primitive solution variables $(\rho, u, v, w, p)^T$.

Material Properties

```
! ============================================================= !
! Equation of State
! ============================================================= !
RefState          = (/1.,0.990268069,0.139173101,0.,4.4642857/)
IniExactFunc      = 1           ! Exact function for initial solution
IniRefState       = 1           ! Refstate used  for initial solution
kappa             = 1.4         ! Heat capacity ratio / isentropic exponent
R                 = 2.857142857 ! Specific gas constant
Pr                = 0.720       ! Prandtl number
mu0               = 0.0002      ! Dynamic Viscosity
```

The chosen velocity vector $(u, v)^T$ yields an angle of attack of $\alpha = 8°$ and a velocity magnitude of 1. The `RefState` are numbered in the order they are supplied in the parameter file. Material properties are selected as described above. 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

$$\mu = \rho u C/Re = 1/Re = 0.0002.$$

Numerical Setup

The DG method in **FLEXI** represents the solution on the mesh using piecewise polynomials. The polynomial degree in this tutorial is chosen as $N = 3$. The remaining numerical settings for the NACA0012 tutorial are summarized below.

```
N                = 3        ! Polynomial degree
MeshFile         =  NACA0012_652_Ng2_mesh.h5
TEnd             = 10       ! End time of the simulation
Analyze_dt       = 0.01     ! Time interval for analysis
CFLscale         = 0.9      ! Scaling for the theoretical CFL number
DFLscale         = 0.9      ! Scaling for the theoretical DFL number
```

Boundary Conditions

Additionally, the setup requires the specification of the boundary conditions for all domain boundaries.

```
! ============================================================ !
! BOUNDARY CONDITIONS
! ============================================================ !
BoundaryName      = BC_wall
BoundaryType      = (/3,1/)     ! Adiabatic wall condition
BoundaryName      = BC_inflow
BoundaryType      = (/2,0/)     ! Weak Dirichlet condition
BoundaryName      = BC_outflow
BoundaryType      = (/2,0/)     ! Weak Dirichlet condition
BoundaryName      = BC_zminus
BoundaryType      = (/1,0/)     ! Periodicity condition
BoundaryName      = BC_zplus
BoundaryType      = (/1,0/)     ! Periodicity condition
```

The freestream boundaries are set to weak Dirichlet conditions (`BCType=2`) using the same reference state as the initialization. The airfoil boundaries are set to adiabatic walls (`BCType=3`). The boundary conditions in z-direction are not relevant for this quasi-2D example and are realized as periodic boundaries (`BCType=1`). All boundary conditions are summarized in the boundary conditions section.

5.10.5 Simulation and Results

We proceed by running the code in parallel. Here, <NUM_PROCS> is an integer denoting the number of processes to be used in parallel.

```
mpirun -np <NUM_PROCS> flexi parameter_flexi.ini
```

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.

```
CalcBodyForces    = T           ! Compute body forces at walls
WriteBodyForces   = T           ! Write body forces to file
```

CalcBodyForces activates the integration of the pressure and viscous forces at each Analyze_dt. WriteBodyForces enforces output of the forces to a <PROJECTNAME>_BodyForces_<BOUNDARYNAME>.csv. In addition to being relevant to the airfoil performance, 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. Fig. 5.30 shows the x- and y-components of the force acting on the airfoil until $TEnd = 10$. The lift and drag coefficients can be easily calculated by rotating these forces from the computational reference frame to the one of the freestream.

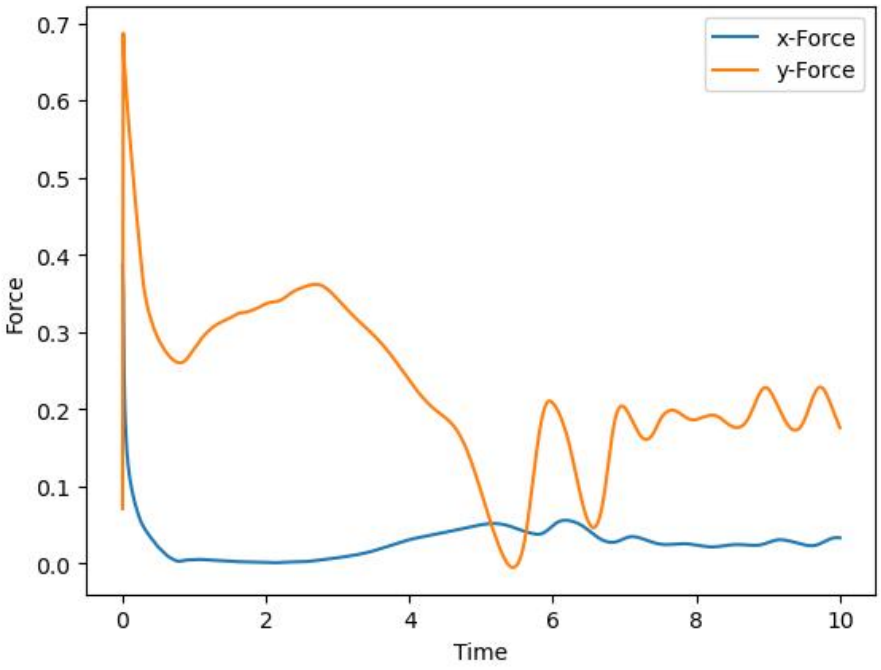

Fig. 5.30: Resulting forces on the NACA0012 airfoil.

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.

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.

```
CalcWallVelocity  = T           ! Compute velocities at wall boundaries
WriteBodyForces   = T           ! Write wall velocities to file
```

`CalcWallVelocity` activates the integration of the wall velocities at each `Analyze_dt`. `WriteWallVelocity` enforces output of the wall velocities to a `<PROJECTNAME>_WallVel_<BOUNDARYNAME>.csv`.

During the computation, we get output like the following.

```
 Wall Velocities (mean/min/max)  :
       BC_wall   2.661973831E-02   2.303391807E-04   6.206912250E-01
```

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 these 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.

Visualization

FLEXI relies on ParaView for its visualization. To visualize the **FLEXI** solution, it must be converted from the HDF5 format into a format suitable for **Paraview**. **FLEXI** provides a post-processing tool *posti_visu* which generates files in VTK format when running the following command.

```
mpirun -np 4 posti_visu parameter_postiVisu.ini parameter_flexi.ini NACA0012_
↪Re5000_AoA8_State_0000000.0*
```

Fig. 5.31 shows a visualization of the density distribution at $t = 10$. 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 **FLEXI** to remove this artificial source.

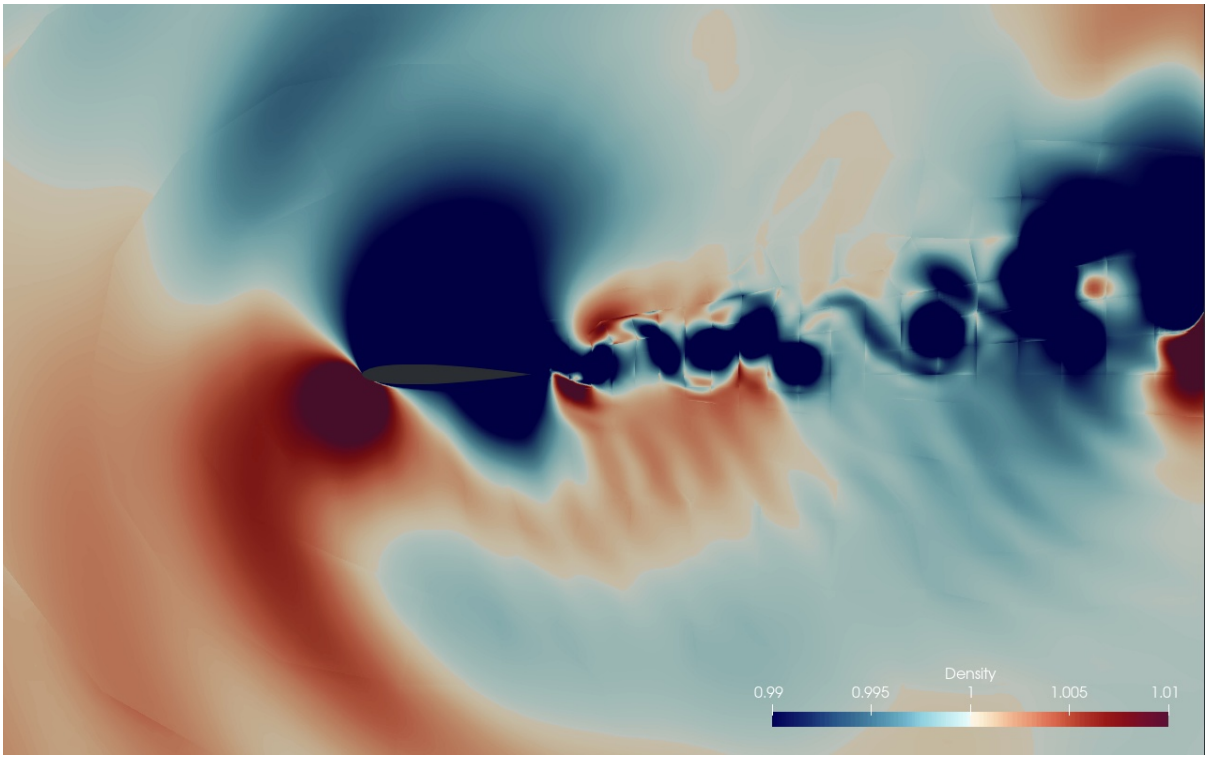

Fig. 5.31: Density field around the NACA0012 airfoil.

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. The associated flags in the parameter file are given in `SPONGE` section of the `parameter_flexi.ini` file. See [8] for the background of our sponge zone implementation.

```
! ============================================================ !
! SPONGE
! ============================================================ !
SpongeLayer           = T           ! Enables the dissipative source term
SpongeShape           = 1           ! Shape of sponge: 1: Cartesian
damping               = 1.0         ! Damping factor of sponge
SpongeXStart          = (/2.0,0,0/) ! Coordinates of start position of sponge
                                    ! - ramp (for SpongeShape=1)
SpongeDistance        = 3.0         ! Length of the sponge ramp
                                    ! - ramp (for SpongeShape=1)
SpongeDir             = (/1,0,0/)   ! Direction vector of the sponge ramp
                                    ! - ramp (for SpongeShape=1)
SpongeBaseFlow        = 4           ! Type of baseflow to be used for sponge
                                    ! - 4: moving average (Pruett baseflow)
tempFilterWidthSponge = 2.0         ! Temporal filter width used to advance
                                    ! Pruett baseflow in time
SpongeViz             = T           ! Write a visualization file of the␣
↪sponge strength
```

The source term is of the form

$$\tilde{U}_t = U_t - d\sigma(\vec{x})\,(U - U_B)\,. \tag{5.3}$$

First, `damping` determines the strength of the source term, i.e., d in eq. (5.3). 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 $\mathcal{O}(1)$, $d = 0.1 \ldots 2$.

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 wider sponge ramp, so that the source term is ramped up more gradually. We choose a parallel ramp by setting `SpongeShape=1`. The ramp's start position, thickness, and direction are controlled by the parameters `SpongeXStart`, `SpongeDistance` and `SpongeDir`, respectively. These parameters govern the shape function $\sigma(\vec{x})$ which smoothly ramps the source term from 0 to 1. 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`.

Caution: The sponge zone is not a physical region but a boundary condition. Place the active source regions far enough downstream of the airfoil to ensure they do not influence the near-field solution.

Next, select the desired base flow, (U_B). For the current configuration, the moving average (`SpongeBaseFlow=4`) is appropriate. It produces a mean field slowly progressing in time, which adapts to the airfoil's surrounding flow. The parameter `tempFilterWidthSponge` determines the effective time window for this average and should be set slightly longer than the largest time scales to be damped. In this example, we use `tempFilterWidthSponge=2.0`, chosen based on the frequency of oscillations in the body forces, as shown in Fig. 5.30.

The moving average base flow requires an initial field. One option is to provide an initial flow field from a file using the `SpongeRefFile` parameter. If this parameter is not set, If this parameter is not set, the code initializes the base flow with the current solution field. Therefore:

- For a new simulation, the base flow is initialized with `IniExactFunc`, which also initializes the solution.
- For a restarted simulation, as in this example, the base flow is initialized using the state file provided for the restart.

Important: When using the moving average base flow, the code creates **baseflow*.h5* files for restarting with the saved base flow state. If these files match the current project name, they are automatically loaded. However, if `SpongeRefFile` is also specified, the base flow restarts from that file instead, which may unintentionally reset the flow state.

5.10.6 Restarting the Simulation

If the simulation is interrupted or needs to extend beyond `TEnd=10`, **FLEXI** can be restarted easily. With the current settings, the solution is saved every 0.1 time units, so to continue the simulation, update `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
```

To restart the simulation, ensure the state files are in the current folder, then run

```
mpirun -np 4 flexi parameter_flexi.ini NACA0012_Re5000_AoA8_State_0000010.
↪000000000.h5
```

You can also adjust the polynomial degree `N` during restart, allowing a lower initial degree for faster convergence, then a higher degree for improved accuracy. The code will automatically project the solution onto the new polynomial basis at startup, but **restart is only possible with the same mesh file**.

5.10.7 Two-dimensional Computation

The laminar flow around this airfoil is inherently two-dimensional. However, so for this simulation was run using a three-dimensional code by imposing periodic boundary conditions with 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 avoid these inefficiencies, **FLEXI** provides an option for true two-dimensional calculations. Set the flag `FLEXI_2D=ON` during configuration, which enables two-dimensional mode. Navigate to your build directory, set the `FLEXI_2D` flag in CMake, reconfigure, and recompile. You must also use a mesh with only one element in the third dimension. Since the tutorial mesh meets this requirement, you can start two-dimensional calculations immediately after recompiling with `FLEXI_2D` enabled.

Once compiled, you can rerun the code with the same parameter file using the new executable. All settings for two-dimensional and three-dimensional computations remain the same. For compatibility, vector parameters (like momentum in `RefState`) should still specify three dimensions; however, the third dimension will simply be ignored. Pay attention to how much faster your code runs when using the two-dimensional version.

By default, **FLEXI** saves *State* files in a three-dimensional format by extruding the solution, ensuring compatibility with existing post-processing tools. To save space, you can set `Output2D=T` to write two-dimensional files. Note that, regardless of dimensionality, arrays retain three dimensions (with one dimension of size 1 for two-dimensional simulations), and all variables, including the spanwise momentum, are still present but set to zero without performing calculations.

5.10.8 Record Points (Probes)

To track the evolution of flow variables near the airfoil's upper and lower surfaces, recording the entire flow field in *State* files can be costly, especially for high temporal resolution or large fields. **FLEXI** offers *record points* (commonly also known as probes) to sample specific flow locations at high temporal resolutions (even down to a single time step) to reduce storage costs.

Setting up record points involves three steps

- Creating the record points
- Running a simulation with active record points
- Visualizing the results

The first and third steps are performed with pre- and post-processing tools, respectively. Enabling record points in a simulation just requires setting the respective options in **FLEXI**.

Record Points Preparation

The coordinates of the record points are defined using the **POSTI** tool *preparerecordpoints*, which is built when the corresponding CMake option is enabled. The tool takes a single parameter file as an input, so run it as follows:

```
posti_preparerecordpoints parameter_recordpoints.ini
```

A sample parameter file is included in the NACA0012 tutorial folder. Define the mesh and project name in this file. The options `NSuper` and `maxTolerance` adjust the record point search algorithm; setting `NSuper` to at least twice the mesh's polynomial degree is a good starting point. The remainder of the parameter file is the actual definition of the record points.

Record points are organized in *sets*, with each set a part of a named *group*. In the example, two sets are created - one for the suction and one for the pressure side of the airfoil -, each assigned to a separate group. FLEXI supports several differing set types, such as single points, lines, or planes. For the NACA example, we use the *boundary layer plane* set type. This set defines a special type of plane by projecting a spline onto the nearest boundary and distributing a user-specified number of points along this line. The plane is then created through extrusion along the boundary's normal vector over a specified distance, with optional stretching to cluster points near the wall. The definition of one of the groups looks like this:

```
GroupName           = suctionSide
BLPlane_GroupID     = 1
BLPlane_nRP         = (/20,30/)
BLPlane_nCP         = 2
BLPlane_CP          = (/0.9,0.014,0.5/)
BLPlane_height      = 0.05
BLPlane_CP          = (/0.999,0.001,0.5/)
BLPlane_height      = 0.05
BLPlane_fac         = 1.04
```

Each *set* of record points must be assigned to a group for identification purposes. The assignment is specified with the *GroupID* (e.g., `BLPlane_GroupID`), here set to 1 for assignment to the first group. After defining the group, configure the parameters of the set. For a boundary layer plane, set the number of points along wall-tangential and wall-normal directions using `BLPlane_nRP`. Then, specify the spline

projected onto the boundary. Set the number of control points in the boundary (`BLPlane_nCP`), the coordinates of each control point (`BLPlane_CP`), the height in the wall-normal direction at this point (`BLPlane_height`) and the stretching factor applied in the wall-normal direction (`BLPlane_fac`). For details on defining other set types, refer to their respective parameter descriptions.

When you run the *preparerecordpoints* tool, it computes the physical coordinates of the record points based on your definitions. The tool then identifies the mesh elements containing each record point and calculates the coordinates in the reference element using Newton's method for interpolation. The results are saved in a file named `<PROJECTNAME>_RPSet.h5`. If you enable the *doVisuRP* option, a visualization of the record points is generated for viewing in ParaView.

Record Points Usage

To utilize record points during a simulation, simply set a few options in the **FLEXI** parameter file. The essential options are given in the following.

```
RP_inUse            = T
RP_DefFile          = NACA0012_RPSet.h5
RP_SamplingOffset   = 1
```

With the `RP_inUse` option, the record points system is enabled or disabled. The `RP_DefFile` option specifies the name of the file created by the *preparerecordpoints* tool containing the record points definitions (e.g., `NACA0012_RPSet.h5`). The `RP_SamplingOffset` option determines the sampling interval for the solution, allowing you to specify that the solution should be sampled at every `RP_SamplingOffset` timestep.

When you run the simulation, in addition to the *State* files, corresponding *RP* files are generated, containing the conservative variable values at each record point for every sampling timestep.

Record Points Post-Processing

Record points data is post-processed using the **visualizerecordpoints** tool. It can merge multiple *RPSet.h5* files to create an extended time series. In addition to visualizing the time series, a multitude of post-processing options is available. These options allow you to compute the mean and fluctuating components of the solution or perform spectral analysis using FFTs. For boundary layer planes, different turbulent quantities, such as skin friction, can be computed directly.

For example, we want to calculate the mean flow $\bar{U}$ and the temporal fluctuations U) at our two record point planes. A sample parameter file for the tool is provided in the tutorial folder. The options set in the parameter file are

```
ProjectName         = NACA0012
RP_DefFile          = NACA0012_RPSet.h5
GroupName           = suctionSide
GroupName           = pressureSide
OutputTimeAverage   = T
doFluctuations      = T
```

In this configuration, we specify a name for the project (`ProjectName`) and provide the path to the file containing the record point definitions (`RP_DefFile`). The `GroupName` options correspond to the names we defined while using the *preparerecordpoints* tool. To evaluate the data, we calculate the temporal

average and the fluctuations, which are determined using the equation

$$U' = U - \bar{U}.$$

We did not specify any specific variables for visualization, meaning all conservative variables stored in the *RPSet.h5* files will be utilized. If you wish to visualize a specific or derived variable (e.g., pressure), you can simply set it in the parameter file as follows

```
VarName              = Pressure
```

You can execute the tool using the command

```
posti_visualizerecordpoints parameter_visualizeRecordpoints.ini NACA0012_
↪Re5000_AoA8_RP_*
```

This takes all the time samples recorded during the simulation as input. For each of the planes, a separate *.vts* file containing the temporal average is generated. The fluctuations are time-resolved data, and due to limitations in the VTK file format, each time step must be written to a single file. To prevent the creation of numerous files in the working directory, these files can be organized into a subfolder named *timeseries*. In this case, *.pvd* files for the fluctuations are created in the working directory. These files can be opened with ParaView, containing the complete time series along with the correct time values. If you to not require ParaView compatibility, the complete solution can be written in a single HDF5 file by setting

```
OutputFormat         = HDF5
```

CHAPTER

SIX

TOOLS OVERVIEW

This section gives an overview over the additional tools contained in the **FLEXI** repository. It also provides references to the tutorials where they are used as reference.

6.1 POSTI Tools

The different **POSTI** tools are used to further post-process the simulation results obtained with **FLEXI**. They can be compiled together with **FLEXI** given the according `cmake` option. A list and description for the input parameters of the associated **POSTI** tools can be displayed with the command

```
[posti_toolname] --help
```

6.1.1 POSTI_VISU

`POSTI_VISU` converts **FLEXI** StateFiles, TimeAverage, and BaseFlow files from the HDF5 format to the ParaView readable `.vtu` (single) or `.pvtu` (parallel) format.

The `POSTI_VISU` tool reads a separate parameter file as optional first argument, while the files to be visualized are passed as the last argument. Without specifying a separate parameter file, the parameters stored in the userblock of the files are used and only the conservative variables are visualized. The latter can be a single file or several files, specified either as simple space-separated list like `Testcase_State_0.h5 Testcase_State_1.h5` or via standard wildcarding like `Testcase_State_*.h5`. The file must contain the entire volume solution, i.e., can be a StateFile or a TimeAverage file, for example.

For serial execution, the `POSTI_VISU` tool is invoked by entering

```
posti_visu [parameter_postiVisu.ini [parameter_flexi.ini]] <statefiles>
```

The tool also runs in parallel by prepending `mpirun -np <no. processors>` to the above command, as usual, provided the compiler option `LIBS_USE_MPI` is enabled.

```
mpirun -np <no. processors> posti_visu [parameter_postiVisu.ini [parameter_
↪flexi.ini]] <statefiles>
```

Important: ParaView can only read state files up to 2 GB in single mode (`.vtu`). Furthermore, the MPI-parallel HDF5 implementation internally uses a signed 32-bit integer, restricting the maximum chunk size to 2 GB per thread. When post-processing with activated `LIBS_USE_MPI` flag, especially with large cases and large files as is often the case with TimeAverage files, the file size of approximately 2 GB per

core must not be exceeded. In this case, the number of cores used must be increased for MPI-parallel executable **POSTI** tools, or **POSTI** must be compiled with `LIBS_USE_MPI=OFF`.

The **POSTI_VISU** tool has a help function that describes the available parameters. This help can be invoked by running the tool with the flag `--help`

```
posti_visu --help
```

The most important runtime parameters to be set in `parameter_postiVisu.ini` are listed in the table below.

Table 6.1: POSTI_VISU parameters.

Parameter	Possible Values	Description
NodeTypeVisu	VISU / GAUSS / GAUSS-LOBATTO / VISU-INNER	Node type of visualization basis; the default *VISU* uses equidistant nodes which include the boundary points of the elements.
NVisu	1 / 2 / 3 / ...	Polynomial degree used to sample the solution for visualization; if left unspecified, it defaults to using the number of collocation points per elements, i.e. $N+1$ per dimension. For high-quality visualization, it is usually advisable to choose a value higher than N in order to keep interpolation errors small.
VarName	Density / VelocityX / ...	Names of the variables to be visualized, parameter can be specified multiple times to visualize more than one variable and set to both conservatives (e.g. *Density*) and primitives (e.g. *VelocityX*). If left unspecified, it defaults to visualizing the five conservative variables.
BoundaryName	Density / WallFriction / y+ / ...	Name of the boundary to visualize. Some variables can only be visualized on the boundary like WallFriction / y+.

In the following all available variables that can be used for visualization are listed.

```
Density, MomentumX, MomentumY, MomentumZ, EnergyStagnationDensity, VelocityX,␣
↪VelocityY, VelocityZ, Pressure, Temperature, VelocityMagnitude,␣
↪VelocitySound, Mach, EnergyStagnation ,EnthalpyStagnation ,Entropy ,
↪TotalTemperature ,TotalPressure ,PressureTimeDeriv ,VorticityX ,
↪VorticityY ,VorticityZ ,VorticityMagnitude ,NormalizedHelicity ,Lambda2␣
↪ ,Dilatation ,QCriterion ,Schlieren ,WallFrictionX ,WallFrictionY ,
↪WallFrictionZ ,WallFrictionMagnitude ,WallHeatTransfer ,x+ ,y+ z+
```

The practical application of `POSTI_VISU` can be practiced in the following tutorials. *Linear Scalar Advection-Diffusion Equation*, *Freestream*, *Lid-driven Cavity*, *SOD Shock Tube*, *Double Mach Reflection*, *Flow Around a Cylinder*, *Flow Around a NACA0012 Airfoil*

6.1.2 POSTI_SWAPMESH

The POSTI_SWAPMESH tool interpolates the solution of a StateFile or a TimeAverage file from one mesh to another, or from one polynomial degree to another. To do so, the parametric coordinates of the interpolation points of the new state are searched in the old mesh. For non-equal elements, a Newton algorithm is used to find the parametric coordinates of the interpolation points. Based on the found parametric coordinates, a high-order interpolation to the interpolation points in the new mesh is performed. Non-conforming meshes are allowed. A reference state can be given for areas in the target mesh which are not covered by the original mesh. The project name and therefore the file name is based on the original project name with '_newMesh' appended, the original file is therefore not overwritten.

For serial execution, the POSTI_SWAPMESH tool is invoked by entering

```
posti_swapmesh parameter_postiSwapmesh.ini <statefiles>
```

The tool also runs in parallel using OpenMP. To run in parallel, the environment variable OMP_NUM_THREADS=XXX needs to be set with the number of threads to be used, provided the compiler option LIBS_USE_OPENMP is enabled. In this case, the parallel execution is the same as the single execution.

A list of parameters used by the POSTI_SWAPMESH tool is listed in the table below. An example of the POSTI_SWAPMESH tool can be found in

```
./flexi/ini/swapmesh
```

Table 6.2: POSTI_SWAPMESH parameters.

Parameter	Possible Values	Description
MeshFileOld	none / MeshFileName.h5	Old mesh file (if different than the one found in the state file)
MeshFileNew	MeshFileName.h5	New mesh file
useCurvedsOld	T/F	Controls usage of high-order information in old mesh. Turn off to discard
useCurvedsNew	T/F	Controls usage of high-order information in new mesh. Turn off to discard
NInter	1 / 2 / 3 / ...	Polynomial degree used for interpolation on new mesh (should be equal or higher than NNew) - the state will be interpolated to this degree and then projected down to NNew
NNew	1 / 2 / 3 / ...	Polynomial degree used in new state files
NSuper	1 / 2 / 3 / ...	Polynomial degree used for supersampling on the old mesh, used to get an initial guess for Newton's method - should be higher than NGeo of old mesh
maxTolerance	value ≥ 0	Tolerance used to mark points as invalid if outside of reference element more than maxTolerance
printTroublemakers	T/F	Turn output of not-found points on or off
RefState	complete conservative state vector	If a RefState is defined, this state will be used at points that are marked as invalid - without a RefState, the program will abort in this case
abortTolerance	value ≥ 0	Tolerance used to decide if the program should abort if no RefState is given
ExtrudeTo3D	T/F	Perform an extrusion of a one-layer mesh to the 3D version Layer which is used in extrusion
ExtrudePeriodic	T/F	Perform a periodic extrusion of a 3D mesh to a mesh with extended z length

6.1.3 Recordpoints

For investigations with a high temporal resolution, such as frequency analyses, it is generally not practical to write complete state files with a high output frequency. Among other things, this would generate a considerable memory requirement and unnecessarily slow down the simulation due to frequent I/O operations. Numerical probes, here called record points, can therefore be used within **FLEXI** for high-frequency outputs in time. These represent point samples and can sample the flow variables with high temporal resolution at defined points in the domain. Multiple record points can be combined to form geometric shapes such as lines or surfaces.

The **POSTI** tools available for this are:

- POSTI_RP_PREPARE: Definition of the points in the flow domain
- POSTI_RP_VISUALIZE: Visualization of the variables recorded at runtime
- POSTI_RP_EVALUATE: Subsequent evaluation of record points on existing volume solutions

To use the **POSTI** tools, the compile flag `POSTI` and the compile flag associated with the respective tool must be activated.

POSTI_RP_PREPARE

The POSTI_RP_PREPARE tool uses its own parameter file. This specifies the grouping of the recordpoints (individual points, lines, planes, ...) and the associated mesh file for which the recordpoints will be defined. The parameters that can be used are documented in the table listed below. The available parameters can also be listed by using the help function

```
posti_preparerecordpoints --help
```

The tool can be executed as follows:

```
posti_preparerecordpoints parameter_recordpoints.ini
```

After successful execution, an additional `h5` file is written with the name of the specified project name with postfix `_RPSet`. In order to record the data during the simulation, this file must be specified in the parameter file when executing **FLEXI**. If the `doVisuRP` option is used, the defined recordpoints are also written to a `vtm` file that can be viewed with Paraview.

To collect the date at the recordpoints locations during the simulation, the recordpoints functionality needs to be activated in the ini file of the FLEXI. This is done by setting `RP_inUse = T`.

The following exemplary options can be added to an existing parameter file:

```
RP_inUse             = T
RP_DefFile           = *_RPSet.h5
RP_SamplingOffset    = 10
RP_MaxMemory         = 100
```

Here, the specified `RP_DefFile` contains the element-local parametric recordpoint coordinates. This is the file generated by `posti_preparerecordpoints`. The option `RP_SamplingOffset` defines the multiple of timestep at which recordpoints are evaluated. Additionally, the `RP_MaxMemory` can be set. It defines the maximum memory in MiB to be used for storing recordpoint state history. If the memory is exceeded before regular I/O level, states are written to the file.

Table 6.3: POSTI_RP_PREPARE parameters.

Parameter	Possible Values	Description
ProjectName		Name used to identify the recordpoint file
MeshFile	MeshFileName.h5	Name of the mesh file
NSuper	1 / 2 / 3 / ...	Number of Newton start values per element per direction.
maxTolerance	value ≥ 0	Tolerance in parameter space at the element boundaries, required to mark a recordpoint as found.
doVisuRP	T/F	Write output file to visualize recordpoints.
GroupName		Name of the RP group (one for each group!)
Line_GroupID		ID of a straight line group, defined by start and end coordinates and the number of points along that line, used to allocate the definition to a specific group
Line_nRP		Number of RPs on line
Line_xstart		Coordinates of start of line

continues on next page

Table 6.3 – continued from previous page

Parameter	Possible Values	Description
Line_xEnd		Coordinates of end of line
Circle_GroupID		ID of a circular group, used to allocate the definition to a specific group
Circle_nRP		Number of RPs along circle
Circle_Center		Coordinates of circle center
Circle_Axis		Axis vector of circle
Circle_Dir		Vector defining the start point on the circle
Circle_Radius		Radius of the circle
Circle_Angle		Angle from the start point, 360° is a full circle
CustomLine_GroupID		ID of a custom line, defined by an arbitrary number of RPs, used to allocate the definition to a specific group
CustomLine_nRP		Number of points on the custom line
CustomLine_x		Coordinates of the points on the custom line
Point_GroupID		ID of a point group, used to allocate the definition to a specific group
Point_x		Coordinates of the single point
Plane_GroupID		ID of a plane group, defined by the corner points and the number of points in both directions, used to allocate the definition to a specific group
Plane_nRP		Number of points in the plane
Plane_CornerX		Coordinates of the 4 corner points (x1,y1,z1,x2,y2,z2,...)
Box_GroupID		ID of a box group, defined by the corner points and the number of points in both directions, used to allocate the definition to a specific group
Box_nRP		Number of points in the box
Box_CornerX		Coordinates of the 8 corner points (x1,y1,z1,x2,y2,z2,...)
Sphere_GroupID		ID of a spherical group, with points on the circumference, used to allocate the definition to a specific group
Sphere_nRP		Number of points on the spere in phi and theta direction
Sphere_Center		Coordinates of sphere center
Sphere_Axis		Axis vector of sphere
Sphere_Dir		Vector defining the start point on the sphere
Sphere_Radius		Radius of the sphere
Sphere_Angle		Phi angle of the sphere (360° is a full sphere)
BLPlane_GroupID		ID of a boundary layer group - works like a plane group, but the plane is created by projecting the points of a spline to the nearest boundary and extruding the plane along the normal with a stretching factor, used to allocate the definition to a specific group
BLPlane_nRP		Number of RPs along and normal to the boundary
BLPlane_nCP		Number of control points defining the spline (at least two)

continues on next page

Table 6.3 – continued from previous page

Parameter	Possible Values	Description
BLPlane_CP		Coordinates of the spline control points
BLPlane_fac		Factor of geometrical stretching in wall-normal direction
BLPlane_height		Wall-normal extend of the plane for each control point
BLBox_GroupID		ID of a boundary layer group - works like a box group, but the box is created by projecting the points of a spline to the nearest boundary and extruding the box along the normal with a stretching factor, used to allocate the definition to a specific group
BLBox_nRP		Number of RPs along and normal to the boundary
BLBox_nCP		Number of control points defining the spline (at least two). Defined once per BLBox
BLBox_nSP		Number of shifted splines (at least one), linear interpolation between the splines in shifted direction (linear extrusion)
BLBox_CP		Coordinates of the spline control points. nCPxnSP needed.
BLBox_fac		Factor of geometrical stretching in wall-normal direction.
BLBox_height		Wall-normal extend of the box for each control point

Exemplary applications of `POSTI_RP_PREPARE` can be found in the following tutorials: *Flow Around a Cylinder*, *Flow Around a NACA0012 Airfoil* Sample parameter files can also be found here.

POSTI_RP_VISUALIZE

During the runtime of the simulation, `ProjectName_RP_*.h5` files are written. These files contain the raw data collected during the simulation. Using the `posti_visualizerecordpoints` tool, the raw data can be further post-processed. The tool takes one or more of the recordpoint files and combines them into a single time series. From the conservative variables that are stored during the simulation, all available derived quantities can be computed. Additionally, several more advanced post-processing algorithms are available. This includes calculation of time averages, FFT, and PSD values and specific boundary layer properties.

The `posti_visualizerecordpoints` tool is designed for single execution only and can be executed as follows:

```
posti_visualizerecordpoints parameter_visuRP.ini projectname_RP_*.h5
```

The parameters that can be used are documented in the table listed below. The available parameters can also be listed by using the help function

```
posti_visualizerecordpoints --help
```

Table 6.4: POSTI_RP_VISUALIZE parameters.

Parameter	Possible Values	Description
ProjectName		Name of the project
GroupName		Name(s) of the group(s) to visualize, must be equal to the name given in preparerecordpoints tool
VarName		Variable name to visualize
RP_DefFile		Path to the *RPset.h5 file
usePrims	T / F	Set to indicate that the RP file contains the primitive and not the conservative variables
meshScale		Specify a scalar scaling factor for the RP coordinates
OutputTimeData	T / F	Should the time series be written? Not compatible with TimeAvg and FFT options!
OutputTimeAverage	T/ F	Should the time average be computed and written?
doFluctuations	T / F	Should the fluctuations be computed and written?
equiTimeSpacing	T / F	Set to interpolate the temporal data to equdistant time steps (always done for operations requiring FFTs)
OutputPoints	T / F	General option to turn off the output of points
OutputLines	T / F	General option to turn off the output of lines
OutputPlanes	T / F	General option to turn off the output of planes
OutputBoxes	T / F	General option to turn off the output of boxes
doFFT	T / F	Calculate a fast Fourier transform of the time signal
doPSD	T / F	Calculate the power spectral density of the time signal
nBlocks		Specify the number of blocks over the time signal used for spectral averaging when calculating spectral quantities
SamplingFreq		Instead of specifying the number of blocks, the sampling frequency in combination with the block size can be set - the number of blocks will then be calculated.
BlockSize		Size of the blocks (in samples) if sampling frequency is given
CutoffFreq		Specify smallest considered frequency in spectral analysis
hanning	T / F	Set to use the Hann window when performing spectral analysis
doTurb	T / F	Set to compute a temporal FFT for each RP and compute turbulent quantities like the kinetic energy over wave number
Box_doBLProps	T / F	Set to calculate seperate boundary layer quantities for boundary layer planes
Box_BLvelScaling		Choose scaling for boundary layer quantities. 0: no scaling, 1: laminar scaling, 3: turbulent scaling

continues on next page

Table 6.4 – continued from previous page

Parameter	Possible Values	Description
Plane_doBLProps	T / F	Set to calculate seperate boundary layer quantities for boundary layer planes
Plane_BLvelScaling		Choose scaling for boundary layer quantities. 0: no scaling, 1: laminar scaling, 3: turbulent scaling
RPRefState		Refstate required for computation of e.g. cp.
RefState		State(s) in primitive variables (density, velx, vely, velz, pressure).
Box_LocalCoords	T / F	Set to use local instead of global coordinates along boxes
Box_LocalVel	T / F	Set to use local instead of global velocities along boxes
Plane_LocalCoords	T / F	Set to use local instead of global coordinates along planes
Plane_LocalVel	T / F	Set to use local instead of global velocities along planes
Line_LocalCoords	T / F	Set to use local instead of global coordinates along lines
Line_LocalVel	T / F	Set to use local instead of global velocities along lines
Line_LocalVel_vec		Vector used for local velocity computation along line
doFilter	T / F	Set to perform temporal filtering for each RP
FilterWidth		Width of the temporal filter
FilterMode		Set to 0 for low pass filter and to 1 for high pass filter
TimeAvgFile		Optional file that contains the temporal averages that should be used
SkipSample		Used to skip every n-th RP evaluation
OutputFormat		Choose the main format for output. 0: ParaView, 2: HDF5
doEnsemble	T / F	Set to perform ensemble averaging for each RP
EnsemblePeriod		Periodic time to be used for ensemble averaging
UseNonDimensionalEqn	T / F	Set true to compute R and mu from bulk Mach Reynolds (nondimensional form.
kappa	1.4	Heat capacity ratio / isentropic exponent
R	287.058	Specific gas constant
Pr	0.72	Prandtl number
mu0	0.0	Dynamic Viscosity
Ts	110.4	Sutherland's law for variable viscosity: Ts
Tref	273.15	Sutherland's law for variable viscosity: Tref
ExpoSuth	1.5	Sutherland's law for variable viscosity: Exponent

Exemplary applications of `POSTI_RP_PREPARE` can be found in the following tutorials: *Flow Around a Cylinder*, *Flow Around a NACA0012 Airfoil* Sample parameter files can also be found here.

POSTI_RP_EVALUATE

The POSTI_RP_EVALUATE tool can be used to extract data for a given simulation at the defined positions for a given RP_DefFile. For this purpose, the recordpoints can be defined as described in section *POSTI_RP_PREPARE* and the data can be extracted. It is possible to extract data not only form StateFiles but also e.g. from TimeAverageFiles. By default, the data set `DG_Solution` is read, which can be used to extract data from StateFiles. To extract data from TimeAverageFiles, a corresponding data set like `Mean` or `Fluc` needs to be specified in the parameter file option `RecordpointsDataSetName`. Using the following command the data can be extracted at the recordpoint positions:

```
posti_evaluaterecordpoints [parameter.ini] <solutionfiles>
```

The tool also runs in parallel by prepending `mpirun -np <no. processors>` to the above command, as usual, provided the compiler option `LIBS_USE_MPI` is enabled.

```
mpirun -np <no. processors> posti_evaluaterecordpoints [parameter.ini]
↪<solutionfiles>
```

After the execution of the `posti_evaluaterecordpoints` tool, a `ProjectName_RP_*.h5` file is written. This file is similar to the recordpoint files written during runtime. Therefore, these files can be visualized as described in section *POSTI_RP_VISUALIZE*.

Important: The MPI-parallel HDF5 implementation internally uses a signed 32-bit integer, restricting the maximum chunk size to 2 GB per thread. When post-processing with activated `LIBS_USE_MPI` flag, especially with large cases and large files as is often the case with TimeAverage files, the file size of approximately 2 GB per core must not be exceeded. In this case, the number of cores used must be increased for MPI-parallel executable **POSTI** tools, or **POSTI** must be compiled with `LIBS_USE_MPI=OFF`.

The parameters for this POSTI tool are listed in the table below.

Table 6.5: POSTI_RP_EVALUATE parameters.

Parameter	Possible Values	Description
RP_inUse	T / F	Set true to compute solution history at points defined in recordpoints file
RP_DefFile	Project-Name_RPSet.h5	File containing element-local parametric recordpoint coordinates and structure
RP_MaxMemory	100	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
RP_SamplingOffset	1	Multiple of timestep at which recordpoints are evaluated
RecordpointsDataSet-Name	DG_Solution / Mean / Fluc / ...	If no state files are given to evaluate, specify the data set name to be used here

The available parameters can also be listed by using the help function

```
posti_evaluaterecordpoints --help
```

CHAPTER

SEVEN

PARAMETER FILE

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 = F         ! 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
```

- the order of defined variables is with one exception generally indifferent, but it is preferable to group similar variables
- the order is only necessary for combined parameters for a setting, e.g., when changing boundary conditions or using multiple sponge zones. For example, if you want to change a specific boundary by addressing its name, the associated boundary type must be defined in the correct order:

```
BoundaryName=inflow          ! BC_Name defined in mesh file
BoundaryType=(/2,0,0,0/)
BoundaryName=outflow         ! BC_Name defined in mesh file
BoundaryType=(/2,0,0,0/)
```

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

MPI	Default	Description
GroupSize	0	Define size of MPI subgroups, used to e.g. perform grouped IO, where group master collects and outputs data.

IO_HDF5	Default	Description
gatheredWrite	F	Set true to activate gathered HDF5 IO for parallel computations. Only local group masters will write data after gathering from local slaves.

Interpolation	Default	Description
N		Polynomial degree of computation to represent to solution

Restart	Default	Description
ResetTime	F	Override solution time to t=0 on restart.
FlushInitialState	F	Check whether (during restart) the statefile from which the restart is performed should be deleted.

Output	Default	Description
NVisu		Polynomial degree at which solution is sampled for visualization.
NOut	-1	Polynomial degree at which solution is written. -1: NOut=N, >0: NOut
ProjectName		Name of the current simulation (mandatory).
Logging	F	Write log files containing debug output.
ErrorFiles	T	Write error files containing error output.
OutputFormat	None	File format for visualization: None, ParaView.
ASCIIOutputFormat	CSV	File format for ASCII files, e.g. body forces: CSV
doPrintStatusLine	F	Print: percentage of time, ...
WriteStateFiles	T	Write HDF5 state files. Disable this only for debugging issues. NO SOLUTION WILL BE WRITTEN!
WriteTimeAvgFiles	T	Write HDF5 time average files. Disable this only for debugging. NO TIME AVERAGE FILES WILL BE WRITTEN!

Attention: OpenMPI v5.x changed the output buffering behaviour. To see the status line on newer versions of OpenMPI, pass `--output :raw` to `mpirun`.

Mesh	Default	Description
MeshFile		(relative) path to meshfile (mandatory).
useCurveds	T	Controls usage of high-order information in mesh. Turn off to discard high-order data and treat curved meshes as linear meshes.
interpolateFromTree	T	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.
meshScale	1.0	Scale the mesh by this factor (shrink/enlarge).
meshdeform	F	Apply simple sine-shaped deformation on Cartesian mesh (for testing).
crossProductMetrics	F	Compute mesh metrics using cross product form. Caution: in this case free-stream preservation is only guaranteed for N=3*NGeo.
debugmesh	0	Output file with visualization and debug information for the mesh. 0: no, visualization, 3: Paraview binary
BoundaryName		Names of boundary conditions to be set (must be present in the mesh!).For each BoundaryName a BoundaryType needs to be specified.
BoundaryType		Type of boundary conditions to be set. Format: (BC_TYPE,BC_STATE)
writePartitionInfo	F	Write information about MPI partitions into a file.
NGeoOverride	-1	Override switch for NGeo. Interpolate mesh to different NGeo.<1: off, >0: Interpolate

Equation of State	Default	Description
UseNonDimensionalEqn	F	Set true to compute R and mu from bulk Mach Reynolds (nondimensional form).
BulkMach		Bulk Mach (UseNonDimensionEqn=T)
BulkReynolds		Bulk Reynolds (UseNonDimensionEqn=T)
kappa	1.4	Heat capacity ratio / isentropic exponent
R	287.058	Specific gas constant
Pr	0.72	Prandtl number
mu0	0.0	Dynamic Viscosity
Ts	110.4	Sutherland's law for variable viscosity: Ts
Tref	273.15	Sutherland's law for variable viscosity: Tref
ExpoSuth	1.5	Sutherland's law for variable viscosity: Exponent

Equation	Default	Description
IniRefState		Refstate required for initialization.
RefState		State(s) in primitive variables (density, velx, vely, velz, pressure).
BCStateFile		File containing the reference solution on the boundary to be used as BC.

Riemann	Default	Description
Riemann	RoeEntropy-Fix	Riemann solver to be used: LF, HLLC, Roe, RoeEntropyFix, HLL, HLLE, HLLEM
RiemannBC	Same	Riemann solver used for boundary conditions: Same, LF, Roe, RoeEntropyFix, HLL, HLLE, HLLEM

Exactfunc	Default	Description
IniExactFunc		Exact function to be used for computing initial solution.
AdvVel		Advection velocity (v1,v2,v3) required for exactfunction CASE(2,21,4,8)
IniAmplitude		Amplitude for synthetic test case
IniFrequency		Frequency for synthetic test case
MachShock	1.5	Parameter required for CASE(10)
PreShockDens	1.0	Parameter required for CASE(10)
IniCenter		Shu Vortex CASE(7) (x,y,z)
IniAxis		Shu Vortex CASE(7) (x,y,z)
IniHalfwidth	0.2	Shu Vortex CASE(7)
JetRadius	1.0	Roundjet CASE(5,51,33)
JetEnd	10.0	Roundjet CASE(5,51,33)
JetAmplitude	1.0	Roundjet CASE(5,51,33)
Ramping	1.0	Subsonic mass inflow CASE(28)
P_Parameter	0.0	Couette-Poiseuille flow CASE(8)
U_Parameter	0.01	Couette-Poiseuille flow CASE(8)
AmplitudeFactor	0.1	Harmonic Gauss Pulse CASE(14)
HarmonicFre-quency	400.0	Harmonic Gauss Pulse CASE(14)
SigmaSqr	0.1	Harmonic Gauss Pulse CASE(14)
delta99_in		Blasius boundary layer CASE(1338)
x_in		Blasius boundary layer CASE(1338)

Filter	Default	Description
FilterType	None	Type of filter to be applied. None, CutOff, Modal, LAF
NFilter		Cut-off mode (FilterType==CutOff or LAF)
LAF_alpha	1.0	Relaxation factor for LAF, see Flad et al. JCP 2016
HestFilterParam		Parameters for Hesthaven filter (FilterType==Modal)

Overintegration	Default	Description
Overintegra-tionType	none	Type of overintegration. None, CutOff, ConsCutOff
NUnder		Polynomial degree to which solution is filtered (Overintegra-tionType == 1 or 2

Lifting	Default	Description
doWeakLifting	F	Set true to perform lifting in weak form.
doConservativeLifting	F	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.

BaseFlow	Default	Description
doBaseFlow	F	Switch on to calculate a baseflow.
BaseFlowFile	none	FLEXI file (e.g. baseflow, TimeAvg) from which baseflow is read.
BaseFlowRefState		Specify which refstate should be used in no baseflowfile is given.
SelectiveFilter	(/ -999, -999, -999 /)	Filter Mean to another polynomial degree.
TimeFilterWidthBaseFlow	1.0	Temporal filter width of exponential, explicit time filter.

Sponge	Default	Description
SpongeLayer	F	Turn on to use sponge regions for reducing reflections at boundaries.
damping		Damping factor of sponge. U_t=U_t-damping*(U-U_base) in fully damped regions.
SpongeShape		Set shape of sponge: (1) ramp : Cartesian / vector-aligned, (2) cylindrical
nSpongeVertices		Define number of vertices per Polygon sponge Zone defining the Polygon
SpongeVertex		Sponge Vertex that defines polygon
SpongeDistance		Length of sponge ramp. The sponge will have maximum strength at the end of the ramp and after that point.
SpongeXStart		Coordinates of start position of sponge ramp (SpongeShape=ramp) or center (SpongeShape=cylindrical).
SpongeXEnd		Coordinates of second point to define Cartesian aligned cube.
SpongeDir		Direction vector of the sponge ramp (SpongeShape=ramp)
SpongeRadius		Radius of the sponge zone (SpongeShape=cylindrical)
SpongeAxis		Axis vector of cylindrical sponge (SpongeShape=cylindrical)
SpongeViz	F	Turn on to write a visualization file of sponge region and strength.
WriteSponge	F	Turn on to write the sponge region and strength to the first state file.
SpongeBaseFlow	1	Type of baseflow to be used for sponge. (1) constant: fixed state,(2), exactfunction: exact function, (3) file: read baseflow file, (4) pruett: temporally varying, solution adaptive Pruett baseflow
SpongeRefState		Index of refstate in ini-file (SpongeBaseFlow=constant)
SpongeExact-Func		Index of exactfunction (SpongeBaseFlow=exactfunction)
SpongeRefFile		FLEXI solution (e.g. TimeAvg) file from which sponge is read.
tempFilterWidth-Sponge		Temporal filter width used to advance Pruett baseflow in time.

TimeDisc	Default	Description
TimeDiscMethod	CarpenterRK4-5	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, eulerimplicit, cranknicolson2-2, esdirk2-3, esdirk3-4, esdirk4-6
TEnd		End time of the simulation (mandatory).
TStart	0.0	Start time of the simulation (optional, conflicts with restart).
CFLScale		Scaling factor for the theoretical CFL number, typical range 0.1..1.0 (mandatory)
DFLScale		Scaling factor for the theoretical DFL number, typical range 0.1..1.0 (mandatory)
dtmin	-1.0	Minimal allowed timestep (optional)
dtkill	-1.0	Kill FLEXI if dt gets below this value (optional)
maxIter	-1	Stop simulation when specified number of timesteps has been performed.
NCalcTimeStepMax	1	Compute dt at least after every Nth timestep.

Implicit	Default	Description
adaptepsNewton	F	Adaptive Newton eps by Runge-Kutta error estimation
EpsNewton	1.0E-03	Newton tolerance, only used if adaptepsNewton=F
nNewtonIter	50	Maximum amount of Newton iterations
EisenstatWalker	F	Adaptive abort criterion for GMRES
gammaEW	0.9	Parameter for Eisenstat Walker adaptation
EpsGMRES	1.0E-03	GMRES Tolerance, only used of EisenstatWalker=F
nRestarts	10	Maximum number of GMRES Restarts
nKDim	30	Maximum number of Krylov subspaces for GMRES, after that a restart is performed
Eps_Method	2	Method of determining the step size of FD approximation of A*v in GMRES, 1: sqrt(machineAccuracy)*scaleps, 2: take norm of solution into account
scaleps	1.0	Scaling factor for step size in FD, mainly used in Eps_Method=1
FD_Order	1	Order of FD approximation (1/2)
PredictorType	0	Type of predictor to be used, 0: use current U, 1: use right hand side, 2: polynomial extrapolation, 3: dense output formula of RK scheme
PredictorOrder	1	Order of predictor to be used (PredictorType=2)

Analyze	Default	Description
CalcErrorNorms	T	Set true to compute L2 and LInf error norms at analyze step.
AnalyzeToFile	F	Set true to output result of error norms to a file (CalcErrorNorms=T)
analyze_dt	0.0	Specifies time interval at which analysis routines are called.
nWriteData	1	Interval 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.
PIDkill	-1.0	Kill FLEXI if PID gets above this value (optional)
NCalcPID	1	Compute PID after every Nth timestep.

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

RecordPoints	Default	Description
RP_inUse	F	Set true to compute solution history at points defined in recordpoints file.
RP_DefFile		File containing element-local parametric recordpoint coordinates and structure.
RP_MaxMemory	100	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.
RP_SamplingOffse	1	Multiple of timestep at which recordpoints are evaluated.

BIBLIOGRAPHY

[1] David A. Kopriva and Gregor Gassner. On the quadrature and weak form choices in collocation type discontinuous galerkin spectral element methods. *Journal of Scientific Computing*, 44:136–155, 2010.

[2] Florian Hindenlang, Gregor J Gassner, Christoph Altmann, Andrea Beck, Marc Staudenmaier, and Claus-Dieter Munz. Explicit discontinuous galerkin methods for unsteady problems. *Computers & Fluids*, 61:86–93, 2012.

[3] Matthias Sonntag and Claus-Dieter Munz. Efficient parallelization of a shock capturing for discontinuous galerkin methods using finite volume sub-cells. *Journal of Scientific Computing*, 70(3):1262–1289, 2017.

[4] Sebastian Hennemann, Andrés M Rueda-Ramírez, Florian J Hindenlang, and Gregor J Gassner. A provably entropy stable subcell shock capturing approach for high order split form dg for the compressible euler equations. *Journal of Computational Physics*, 426:109935, 2021.

[5] M. Carpenter and C. Kennedy. Fourth-order 2N-storage Runge-Kutta schemes. Technical Report NASA TM 109111, Langley Research Center, Hampton, Virginia, 1994.

[6] Jens Niegemann, Richard Diehl, and Kurt Busch. Efficient low-storage runge–kutta schemes with optimized stability regions. *Journal of Computational Physics*, 231(2):364–372, 2012.

[7] David A Kopriva, Stephen L Woodruff, and M Yousuff Hussaini. Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method. *International journal for numerical methods in engineering*, 53(1):105–122, 2002.

[8] David Flad, Hannes Frank, Andrea D Beck, and Claus-Dieter Munz. A discontinuous galerkin spectral element method for the direct numerical simulation of aeroacoustics. In *20th AIAA/CEAS Aeroacoustics Conference*, 2740. 2014.

[9] Jan-Reneé Carlson. Inflow/outflow boundary conditions with application to fun3d. Technical Report, Langley Research Center, Hampton, Virginia, 2011.

[10] Gregor J. Gassner, Andrew R. Winters, and David A. Kopriva. Split form nodal discontinuous Galerkin schemes with summation-by-parts property for the compressible Euler equations. *J. Comput. Phys.*, 327:39–66, December 2016. doi:10.1016/j.jcp.2016.09.013.

[11] Gregor J Gassner and Andrea D Beck. On the accuracy of high-order discretizations for underresolved turbulence simulations. *Theoretical and Computational Fluid Dynamics*, 27(3-4):221–237, 2013.

[12] F. Bassi and S. Rebay. A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 131:267–279, 1997.

[13] F. Bassi, S. Rebay, G. Mariotti, S. Pedinotti, and M. Savini. A high-order accurate discontinuous finite element method fir inviscid an viscous turbomachinery flows. In R. Decuypere and G. Dibelius, editors, *Proceedings of 2nd European Conference on Turbomachinery, Fluid and Thermodynamics*, 99–108. Technologisch Instituut, Antwerpen, Belgium, 1997.

[14] Sergio Pirozzoli. Generalized conservative approximations of split convective derivative operators. *Journal of Computational Physics*, 229:7180–7190, 2010.

[15] Praveen Chandrashekar. Kinetic energy preserving and entropy stable finite volume schemes for compressible euler and navier-stokes equations. *Communications in Computational Physics*, 14:1252–1286, 2013.

[16] Marcel Blind, Min Gao, Daniel Kempf, Patrick Kopper, Marius Kurz, Anna Schwarz, and Andrea Beck. Towards exascale cfd simulations using the discontinuous galerkin solver flexi. In *High Performance Computing in Science and Engineering '23 (in press)*. 2024.

[17] Gregor Gassner and David A Kopriva. A comparison of the dispersion and dissipation errors of gauss and gauss–lobatto discontinuous galerkin spectral element methods. *SIAM Journal on Scientific Computing*, 33(5):2560–2579, 2011.

[18] Patrick J. Roache. Code Verification by the Method of Manufactured Solutions. *Journal of Fluids Engineering*, 124(1):4–10, November 2001.

[19] UKNG Ghia, Kirti N Ghia, and CT Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.

[20] Zhen Gao, Jan S. Hesthaven, and Tim Warburton. Efficient absorbing layers for weakly compressible flows. 2016. URL: https://infoscience.epfl.ch/handle/20.500.14299/97200.

[21] Geoffrey Ingram Taylor and Albert Edward Green. Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences*, 158(895):499–521, February 1937. doi:10.1098/rspa.1937.0036.

[22] Gary A Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of computational physics*, 27(1):1–31, 1978.

[23] Per-Olof Persson and Jaime Peraire. Sub-Cell Shock Capturing for Discontinuous Galerkin Methods. In *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics, 2006. doi:10.2514/6.2006-112.

[24] Phillip Colella and Paul R Woodward. The piecewise parabolic method (ppm) for gas-dynamical simulations. *Journal of computational physics*, 54(1):174–201, 1984.

[25] Antony Jameson, Wolfgang Schmidt, and Eli Turkel. Numerical solution of the euler equations by finite volume methods using runge kutta time stepping schemes. In *14th fluid and plasma dynamics conference*, 1259. 1981.

[26] Robert D Moser, John Kim, and Nagi N Mansour. Direct numerical simulation of turbulent channel flow up to re= 590. *Physics of fluids*, 11(4):943–945, 1999.

[27] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164, March 1963. doi:10.1175/1520-0493(1963)091<0099:gcewtp>2.3.co;2.

[28] Peter J Schmid, Knud Erik Meyer, and Oliver Pust. Dynamic mode decomposition and proper orthogonal decomposition of flow in a lid-driven cylindrical cavity. In *8th International Symposium on Particle Image Velocimetry*, 25–28. 2009.

[29] Xiangxiong Zhang and Chi-Wang Shu. On positivity-preserving high order discontinuous galerkin schemes for compressible euler equations on rectangular meshes. *Journal of Computational Physics*, 229(23):8918–8934, 2010. doi:10.1016/j.jcp.2010.08.016.