A Simple Cartesian-based Two-Dimensional Solver (ASC2D) Based on the Discontinuous Galerkin Finite Element Method

Edmond K. Shehadi

Abstract

This is a brief description about the `ASC2D` code. First and foremost, this is a test code that was developed mainly for investigating some of the non-reflecting boundary conditions (NRBCs). Mainly, this is a two-dimensional code that operates on a multi-zone structured, orthogonal grid. The spatial solver is based on a (nodal) discontinuous Galerkin finite element method (DG-FEM), while the temporal discretization uses explicit time-stepping only. The governing equations are based on the non-linear Euler equations. If users are interested in adding features, please feel free to do so. A lot of the structure of the code is written with readability in mind over performance. In the future, there is a plan to extend this to a hybrid-parallel (MPI+OpenMP), vectorized (SIMD) DG-FEM that solves the Navier-Stokes equations and capable of using a curvilinear single-block grid based on the `Plot3D` format.

Contents

List of Figures

1 Introduction

This is a C++ test code used mainly for investigating and comparing different non-reflecting boundary conditions (NRBCs) in the context of the high-order discontinuous Galerkin finite element method (DG-FEM). The solver generates its own grid, since it is fairly straightforward to assemble a structured, orthogonal grid. Moreover, it has the capability of generating up to nine different zones that are coupled together; hence a multi-zone solver.

This document brieflt describes *some* of the features that currently exist in ASC2D. It does not delve into the physics or the mathematics under the hood. If the interested reader wants to take a look at the theory behind all this, then refer to Shehadi and van der Weide [2022]. Note, unless stated otherwise, all dimensions used in this code are based on SI units.

The remaining portions of this document discuss how to compile and set-up a test case in Section 2. In the likelihood this is a first-time user, it is advisable to simply get comfortable with this software by going through the examples provided by the repository. The remaining five main sections of the dictionary files and rest of this document correspond to

- Section 3 discusses the grid and zone configuration.
- Section 4 discusses the simulation and solver specification.
- Section 5 discusses the initial condition and flow properties.
- Section 6 discusses the boundary condition set up.
- Section 7 discusses the input/output and (post-)processing options.

2 Quick Start

In this section, a quick guide to downloading, installing and running a sample case is detailed. In the below, it is assumed the user already has the open-source version control Git installed along with the build automation Make tool.

2.1 Downloading

First and foremost, this code has been developed and run on explicitly UNIX environments. The first step consists in downloading or cloning a version of this code from the original repository. To do this in the usual manner, assuming you have Git installed, open your terminal, go to the directory you plan to install this code in and type the following command

```
git clone https://github.com/inquisitor101/ASC2D.git
```

This clones a version of the ASC2D code into your current directory. More specifically, once you enter the directory ASC2D/, you should obtain six files: the license, a readme file, this PDF document, a template test-case directory example/, a post-processing programs directory tools/ and the actual code directory asc/, as follows

```
LICENSE  README.md  doc.pdf  example/  tools/  asc/
```

Afterwards, enter the code directory asc/. In it, you should find available the below eight files

```
Makefile  Doxyfile  src/  include/  external/  Dependencies/  obj/  bin/
```

2.2 Installing

To compile the program, first we must set up the required Git submodules, namely the Eigen library. This is done via the two git commands applied from the root directory ASC2D/, as follows

```
git submodule init && git submodule update
```

Then, the actual installation is carried via the below make command from the asc/ directory

```
make
```

The first two git commands will clone the required external Eigen library. In this way, the Eigen library is treated as a submodule to the current code. Here, Eigen is a template library for linear algebra computations. The third (`make`) command is the actual compilation of the code. The executable for this code is found in `bin/ASC2D`. In this Makefile, the `C++17` standard is specified in conjunction with the GNU compiler `g++`.

Note, many options are predefined for use in this Makefile. By default, running `make` uses the optimized compiler with the `OpenMP` multi-thread flag. Also, hard-coded in the Makefile is that it compiles using eight threads, as can be seen from the first line of code in the Makefile: `make -j 8`. You may simply choose to remove or adjust the number of threads. To use a serial implementation of this code without `OpenMP`, simply execute `make standard`. For debugging, one also has two predefined options for compiling. Run `make debug` for a parallelized compilation with debugging options and `make debugSerial` for a serial debug version. Finally, run `make clean` to clean the directory from any previously compiled executable files and objects.

2.3 Running a Sample Case

After installing the solver, the next will illustrate how to use it in the framework of a particular test case. Conveniently, three different test cases are located inside the `example/` folder, as such

```
IsentropicVortex/  PressurePulse/  VortexRollup/
```

In each of these folders, a set-up for an illustrative simulation based on the isentropic vortex, pressure pulse and shear-flow vortex roll-up are used. For details on what these are, refer to Shehadi and van der Weide [2022]. Furthermore, in each of these three problems, six different types of simulations are pre-defined. These are

```
cml/  nscbc/  pml/  riemann/  sponge/  supersonic/
```

These correspond to a characteristic matching layer (CML), a Navier-Stokes characteristic boundary condition (NSCBC), a perfectly matched layer (PML), a Riemann extrapolation boundary, a sponge layer and a supersonic layer. These are different methods to impose open or non-reflective boundary conditions.

Furthermore, in each of the above six template simulation folders, there exists eleven files, namely

```
paramDict.cfg sweeper.clean peek.gnuplot anim/ data/ init/ logs/ proc/ surf/ zone/ gnuplot/
```

As an example, let us consider running a vortex roll-up test case. Enter the directory `VortexRollup/` and select one of the sub-directories therein. Note, these are all based on the same test case, but utilize different ways to handle open or non-reflecting BCs. For simplicity, let us use the one based on extrapolation (which is a reflecting BC), named: `riemann/`. To run this case, simply provide the path to the executable followed by the dictionary configuration file (i.e. `paramDict.cfg`), as follows

```
../../../asc/bin/ASC2D paramDict.cfg
```

If you compiled with the parallelized version of `ASC2D`, then before you run the above command, specify the amount of threads you want by entering the command: `export OMP_NUM_THREADS=n`, where `n` is the number of threads required. As an illustration, assuming we intend on running with 4 threads and redirect the output to a file `info.log` inside the `logs/` folder. Then, first specify the number of threads needed as follows

```
export OMP_NUM_THREADS=4
```

followed by the executable with the relevant dictionary and output redirection log file, for example

```
../../../asc/bin/ASC2D paramDict.cfg > logs/info.log 2>&1 &
```

This should run the current test case with the predefined configuration options found in the provided dictionary file `paramDict.cfg`. In turn, the data files are saved in `data/` and the VTK visualization files in `anim/` while the (post-)processed files are saved in `proc/`. The output of the solver can be found in the file `logs/info.log`. To visualize the solution, open the files in `anim/` with `ParaView`.

After running this test case, if one needs to re-run things, it is advisable to clean all local working directories by executing the command

```
./sweeper.clean
```

which will erase the files in all seven directories: `anim/ data/ logs/ proc/ surf/ gnuplot/ zone/`.

3 Grid and Zone Configuration

The internal grid generator is quite primitive and capable of only generating simple structured and orthogonal rectangular geometries. The domain box of the physical/main zone is specified as follows

```
DOMAIN_BOUND = ( WEST, EAST, SOUTH, NORTH )
```

where WEST and EAST correspond to the minimum and maximum x-coordinate of the computational domain, respectively, while SOUTH and NORTH are those of the minimum and maximum of the y-coordinate, respectively. Throughout the entire code design, the grid convention is always such that: bottom is South, top is North, left is West and right is East.

After specifying the physical domain, the next step is to determine how many zones do you need. These zones are by definition wrapped around the main zone, which is always designated as the physical domain. For a visual illustration, see Figure 1. Following which, a zone marker is required to identify which zone index corresponds to which zone. These are primarily needed when specifying different zonal parameters in each zone. Note, these follow the same sequence of indices as their order in the markers input. For the most general, nine-zone, scheme illustrated in Figure 1, the zone markers are

```
ZONE_MARKER = ( ZONE_MAIN, ZONE_WEST, ZONE_EAST, ZONE_SOUTH, ZONE_NORTH, ZONE_CORNER_0, ZONE_CORNER_1, ZONE_CORNER_2, ZONE_CORNER_3 )
```

where it is expect that the first input argument is always that of the physical domain, i.e. ZONE_MAIN.

Now that the physical domain bounding coordinates are specified and the number of zones required is inserted, the next step is to describe the grid in each of these zones. To specify the number elements in each zone in the x- and y-direction use NUMBER_XELEM and NUMBER_YELEM, respectively. Similarly, to specify the type of degrees-of-freedom (DOFs) on the solution nodes use DOF_TYPE. The DOFs have two options: Legendre-Gauss-Lobatto (LGL) or equidistant (EQD). Furthermore, to specify the solution polynomial order and the type of Riemann solver, use POLY_ORDER_SOL and RIEMANN_SOLVER, respectively. Three types of (approximate) Riemann solvers are available and they are: Rusanov, Roe's and Ismail-Roe.

For all the options that require multi-zone arguments (i.e. one parameter per each zone), there are three types of choices to input. First, if for instance one desires to fix the type of DOFs in all zones to LGL, then this can be simply done as follows

```
DOF_TYPE = ( LGL )
```

Second, if one desires to have one value in the physical domain and another value in all the remaining zones, then this may be done as follows

```
DOF_TYPE = ( LGL, EQD )
```

which means that LGL nodes are used in the ZONE_MAIN and all remaining (eight) zones are specified as equidistant nodes (EQD).

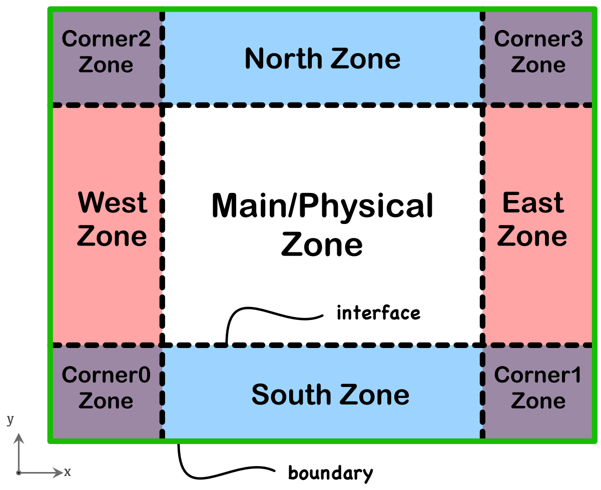

Figure 1: Illustration of a nine-zone set up; Dashed lines are interfaces between zones and green lines denote the terminating boundaries.

Thirdly, if one desires to specify different options in different zones, besides just the physical domain, then they have to do so individually as follows

```
POLY_ORDER_SOL = ( 1, 2, 3, 4, 5, 6, 7, 8, 9 )
```

which corresponds to the below configuration for each zone

- `ZONE_MAIN` having `nPoly = 1`
- `ZONE_WEST` having `nPoly = 2`
- `ZONE_EAST` having `nPoly = 3`
- `ZONE_SOUTH` having `nPoly = 4`
- `ZONE_NORTH` having `nPoly = 5`
- `ZONE_CORNER_0` having `nPoly = 6`
- `ZONE_CORNER_1` having `nPoly = 7`
- `ZONE_CORNER_2` having `nPoly = 8`
- `ZONE_CORNER_3` having `nPoly = 9`

Recall, these configurations are based on the indicial entries input in the `ZONE_MARKER` earlier. Therefore, always keep that in mind else the results might turn out rather confusing and meaningless.

Note, by default the option for `ZONE_CONFORMITY` is always turned on. The current version of the solver cannot handle non-conforming elements. Thus, unless the user intends on adding such a feature, either explicitly mention it is `true` or completely omit this keyword, since the default uses conforming zones and elements anyway.

Given the assumption that the grid is based on conforming elements and zones, specifying the number of elements in each zone is not always needed. As an example, consider a physical domain comprised of 20 elements in the y-direction, respectively. Given the sequence of nine zones input earlier, then `ZONE_WEST` and `ZONE_EAST` will automatically get assigned 20 elements in the y-direction as well. This will overwrite the entries of whatever is given in the `NUMBER_YELEM` entry of these zones, in order to ensure conformity. This also applies for `NUMBER_XELEM`, since the entries corresponding to `ZONE_SOUTH` and `ZONE_NORTH` are automatically overwritten by the number of x-elements taken from `ZONE_MAIN`.

Sometimes it is advantageous to use larger element sizes in the extruded direction, for computational reasons. This can be done via the option `ELEMENT_RATIO`. To elaborate, this is only needed when more than one zone is specified and its arguments are

```
ELEMENT_RATIO = ( H_WEST, H_EAST, H_SOUTH, H_NORTH )
```

where `H_WEST` corresponds to the element expansion ratio in the `ZONE_WEST` and analogously for the remaining directions and elements. Note, these expansion ratios are based on the adjacent element from the neighboring zone and are only expanded in the boundary-normal direction via a geometric expansion:

$$h_{i+1} = r h_i$$

where r is the expansion coefficient and h_i denotes the i^{th} element size in the boundary-normal direction. Note, the element size at the interface is based on the adjacent element from the connecting zone.

If however, for certain applications one desires to investigate the physical phenomenon taking place in a particular region, then this can be achieved relatively easily by refining the grid around this area of interest through a "control point". Note, this is only applicable in the physical domain (i.e. `ZONE_MAIN`). By default, this option is always turned off. However, to use this option, one has to explicitly specify

```
UNIFORM_GRID_RESOLUTION = false
```

and then follow afterwards with the specification of the control point location via

```
BLOCK_INTERFACE_LOCATION = ( x, y )
```

where x and y are the coordinates of the control point.

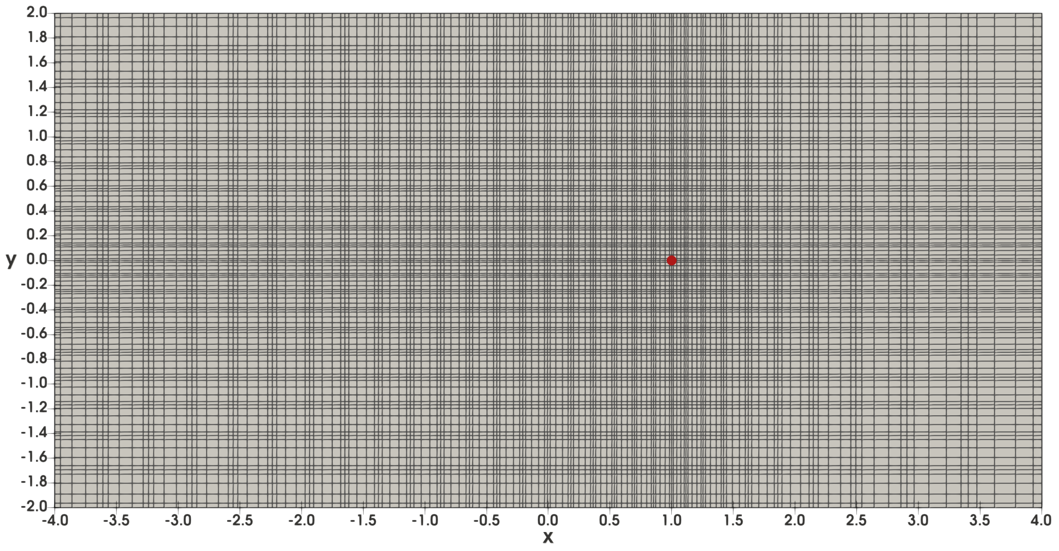

Figure 2: Sketch of a physical domain with non-uniform elements refined in the vicinity of a control point $(1.0, 0.0)$.

After that, an expansion ratio for each of the four blocks in the physical domain is provided. The blocks are defined as min/max blocks per each dimension. Their corresponding expansion ratios are specified by

```
BLOCK_EXPANSION_RATIO = ( RATIO_XMIN, RATIO_XMAX, RATIO_YMIN, RATIO_YMAX )
```

where `RATIO_XMIN` corresponds to the x-dimension element expansion ratio on the block left of the control point, while `RATIO_XMAX` is that of the block right of the control point. Likewise also for the y-dimension element expansion ratios in entries `RATIO_YMIN` and `RATIO_YMAX` for the blocks below and above the control point, respectively. Additionally, the user needs to explicitly specify the number of elements in each of these blocks through the options `NUMBER_XELEM_BLOCK` and `NUMBER_YELEM_BLOCK`.

As an example, consider a domain size of $[-4.0, 4.0] \times [-2.0, 2.0]$ in a single zone environment with 30×20 elements, each based on a `nPoly = 5` solution situated on LGL nodes. Assume some interesting physics (e.g. vortex shedding or shear flow interface) is located at the control point $(1.0, 0.0)$, which requires a finer grid resolution to capture and investigate. This implies that the four blocks are defined as

- `BLOCK_XMIN`: $\forall\, y,$ such that: $x \in [-4.0, \quad 1.0]$.
- `BLOCK_XMAX`: $\forall\, y,$ such that: $x \in [\ \ 1.0, \quad 4.0]$.
- `BLOCK_YMIN`: $\forall\, x,$ such that: $y \in [-2.0, \quad 0.0]$.
- `BLOCK_YMAX`: $\forall\, x,$ such that: $y \in [\ \ 0.0, \quad 2.0]$.

An illustration of this set up is presented in Figure 1, using the below parameters

```
BLOCK_INTERFACE_LOCATION = ( 1.0, 0.0 )
NUMBER_XELEM_BLOCK       = ( 20, 10 )
NUMBER_YELEM_BLOCK       = ( 10, 10 )
BLOCK_EXPANSION_RATIO    = ( 0.95, 1.20, 0.90, 1.10  )
```

Note that the number of elements of both block blocks in each dimension must be the same as the original total number of elements in that dimension in the physical domain. In other words, in the above example of Figure 2 the total number of elements in the x-direction is 30. Accordingly, the total number of elements in both blocks of `NUMBER_XELEM_BLOCK` is $20 + 10 = 30$ too. This is also obligatory for the number of elements in the y-direction too.

4 Simulation and Solver Specification

In this section, the simulation set-up and general solver options are specified. The simulation start time is specified via START_TIME and the end time is done through FINAL_TIME. As a reminder, these are based on SI units and in this case, seconds. The time step is adjusted via the TIME_STEP option. Additionally, there exists an upper bound on the maximum number of temporal iterations done through MAX_ITER. Note, the simulation will stop once either of the maximum number of iterations or the simulation end time is reached first.

There also exists an option to use adaptive time stepping. This is specified via ADAPT_TIME which is by default false. If adaptive time stepping is used, then the CFL number is specified via CFL_NUMBER. Note, if a negative value is specified in CFL_NUMBER, then a fixed time step is used based on the input of TIME_STEP. The solver will complain if CFL_NUMBER is negative (i.e. a fixed time step is specified) while ADAPT_TIME is turned on.

In this current version of ASC2D, only two explicit time-marching schemes are available: LSRK4 and SSPRK3. These correspond to the fourth-order low-storage Runge-Kutta scheme and the third-order strong-stability-preserving scheme, respectively. Either of those time integration options is specified via the TIME_MARCHING keyword.

Besides the temporal configuration, the user must also specify the integration rule (based on a single dimension) to use. This is done through INTEGRATION_FACTOR and the default is taken as three. This means that the integration assumes the solution is thrice the original polynomial order. For example, if nPoly = 5, and the integration rule is three, then the quadrature nodes per dimension integrate exactly a polynomial order of nPoly = 15. This over-integration is needed, since the Euler equations are non-linear and irrational. If no over-integration is used then the solution is aliased. On the other hand, if the over-integration is very high, then the computations are needlessly expensive.

In the likelihood of a multi-zone arrangement, the user needs to specify the type of equations solved in each zone. This is done via the TYPE_BUFFER_LAYER option. As of now, this has three possible inputs

- NONE, which implies the Euler equations are solved.
- SPONGE, which indicates this is a sponge or absorbing layer.
- PML, which corresponds to a perfectly matched layer.

Furthermore, the user has the possibility of filtering the solution through FILTER_SOLUTION per each zone. The default is NONE, which uses no filters. As of now, the only option available is to use EXPONENTIAL, which is an exponential filter. If an exponential filter is selected, its below characteristics should be specified

```
FILTER_CHARACTERISTICS = ( nf, Nc, s, alpha )
```

where nf corresponds to the frequency, Nc is the number of unaffected or unfiltered modes, s and alpha are coefficients that determine the strength of the filter. For more information about this type of filter, see Nordström and Winters [2021]. Note, by filtering the solution, the *nodal* DG-FEM formulation is converted into a *modal* one where the filter is applied and then converted back into its *nodal* form.

5 Initial Condition and Flow Properties

In the current form of ASC2D, many different initial conditions may be specified. However, the first step is to make sure if this is a restart simulation or not. This can be done by setting the RESTART_SOLUTION option to either true or false. If this is a restart simulation, make sure to specify the restart file via RESTART_FILENAME. Additionally, one is required to specify the VTK visualization file output in OUTPUT_VTK_FILENAME and the solution data in OUTPUT_SOL_FILENAME. The solution data uses the format required in order to be used as an initial restart condition, as long as its corresponding file is linked in RESTART_FILENAME.

In the case where the user does not want to use a restart condition and instead resort to a predefined initial condition, then five current initializations exist. These are specified in TYPE_IC. Note, if a restart solution is used, then the entry in this field is automatically ignored. The five possible initial conditions are

- GAUSSIAN_PRESSURE
- ISENTROPIC_VORTEX
- ENTROPY_WAVE
- VORTEX_ROLLUP
- ACOUSTIC_PLANE_WAVE

Note, with the exception of the VORTEX_ROLLUP, all the remaining initializations are dimensional. For most of these initializations, refer to the explanation in Shehadi and van der Weide [2022]. The acoustic plane wave is taken analogous to the one in Maeda and Colonius [2017]. Another remark is that the TYPE_IC may be specified differently in each zone – however this is not recommended.

Most of these initializations require two flow properties: the background Mach number and the flow angle – taken with respect to the x-direction. These may be prescribed in the options FREESTREAM_MACH and FLOW_ANGLE. Note, the flow angle is defined in degrees (not radians!).

Some of the initializations also require additional information to tune the Gaussian profiles and, whenever applicable, the time-dependent source term. Some of the usual options that are commonly used are

```
DISTURBANCE_CENTER = ( x, y )
DISTURBANCE_RATIO = a0
DISTURBANCE_WIDTH = w
FREQUENCY         = f
PERIODIC_PULSE    = true/false
```

where (x, y) are the location of the center of the source term or disturbance, a0 is the amplitude of the source term with respect to the background flow condition, w is the width of the of the disturbance and f is the frequency in Hertz of the time-dependent source term in case PERIODIC_PULSE is set to true.

6 Boundary Condition Set Up

This solver mainly served as a test code for non-reflecting boundary conditions (NRBCs). Thus, many such methods and combinations of which exist in its current implementation. As a reminder, to understand the theoretical background of these methods see Shehadi and van der Weide [2022]. Before delving into them, a few remarks are brought forth first.

In the case of a single zone simulation, obviously all four boundaries (south, north, west, east) are the boundaries of the domain. In the case of a multi-zone simulation, the boundary conditions are only imposed on the external or terminating boundaries of all the zones. As an example, the boundary conditions are imposed on the green line in Figure 1, which are always composed of four faces (south, north, west, east). The internal boundaries connecting in between adjacent zones are handled via an interface condition which couples between the different zones, see dashed-lines in Figure 1. The second remark is that the boundary conditions (BCs) are specified in

```
MARKER_BOUNDARY = ( BC_SOUTH, BC_NORTH, BC_WEST, BC_EAST )
```

where the convention is that the BCs correspond to that on the south, north, west and then east boundary.

As for the type of boundary conditions themselves, ten currently exist in ASC2D. These are

- PERIODIC
- SYMMETRY
- CBC_OUTLET
- CBC_STATIC_INLET
- CBC_TOTAL_INLET
- STATIC_INLET
- TOTAL_INLET
- SUPERSONIC_INLET
- STATIC_OUTLET
- SUPERSONIC_OUTLET

Note, in case of PERIODIC boundaries, the opposite boundary need to also be specified as periodic as well. For example, the west and east must be periodic, if either of them is specified so. Similarly for the north and south boundaries. The SYMMETRY boundary condition is essentially a slip or no-penetration condition.

The boundaries in CBC_OUTLET, CBC_STATIC_INLET and CBC_TOTAL_INLET are characteristic-based conditions for an outlet, a static-based inlet and a total (stagnation)-condition inlet, respectively. The conditions in STATIC_INLET and STATIC_OUTLET are based on a Riemann extrapolation for the static conditions on the inlet and outlet, while TOTAL_INLET uses the same extrapolation but for an inlet based on the total or stagnation conditions. Finally, SUPERSONIC_INLET and SUPERSONIC_OUTLET are the supersonic boundaries, which imply the outlet is entirely based on the internal solution while the inlet is entirely based on the external imposed solution.

Note, in all these conditions, the actual data is imposed from the initial condition, based on its background flow properties. Therefore, there is no need to explicitly specify the values at those boundaries. This was selected, because the considered benchmark problems allow for this. In case of a generic implementation, this can be easily modified to accommodate different and perhaps even time-dependent BCs.

In the case of characteristic boundaries, additional options are required for the tuning coefficients in both normal and transverse directions. These can be found in

```
OUTLET_NSCBC_TUNING = ( sigma, beta_l, beta_t, len, factor )

INLET_NSCBC_TUNING  = ( sigma, len )

INLET_NONREFLECTING = true/false
```

where sigma is the normal relaxation coefficient, beta_l and beta_t are transverse relaxation coefficients based on the coupled and uncoupled waves, respectively, len is a characteristic length scale and factor is an extra "safety" tuning parameter that is multiplied by the transverse coefficients beta_l and beta_t. Note, if either of the transverse coefficients are negative, then this implies that coefficient uses the local averaged Mach number over the characteristic boundary. For example, if beta_t < 0, then in the formulation beta_t = factor * Mavg, where Mavg is the time-dependant (spatially) averaged Mach number over the respective boundary.

After specifying the type of boundary condition, in the case of a multi-zone procedure with zones whose TYPE_BUFFER_LAYER is specified as either PML or SPONGE, then additional zonal information are required. In both the PML and sponge layer, a damping coefficient and an exponential coefficient are required for each zone via

```
SPONGE_DAMPING_CONSTANT = ( ZONE_MAIN, ... )
SPONGE_DAMPING_EXPONENT = ( ZONE_MAIN, ... )
```

where the coefficients for the physical domain (i.e. ZONE_MAIN) should be kept 0.0, since this is a physical domain, unless intended otherwise.

Furthermore, in all additional zones (besides the physical domain), grid stretching is applicable. Grid stretching introduces additional dissipation, which tend to assist in damping additional perturbations without resorting to long, expensive computational domains. The grid stretching options are controlled via

```
GRID_STRETCHING_CONSTANT = ( ZONE_MAIN, ... )
GRID_STRETCHING_EXPONENT = ( ZONE_MAIN, ... )
```

Also, similar to the damping and grid stretching tuning parameters, there exists an option to specify the profile of the artificial convective velocity term. This is used in the supersonic-type buffer zones. Its respective options are found in

```
ARTIFICIAL_CONVECTION_CONSTANT = ( ZONE_MAIN, ... )
ARTIFICIAL_CONVECTION_EXPONENT = ( ZONE_MAIN, ... )
```

Last but not least, analogous to the previous additional layer parameters, a characteristic matching layer (CML) option is available. This allows for the *gradual* "matching" of the incoming characteristics until they are completely eliminated at the external boundaries. The matching function profile options are found in

```
CHARACTERISTIC_MATCHING_CONSTANT = ( ZONE_MAIN, ... )
CHARACTERISTIC_MATCHING_EXPONENT = ( ZONE_MAIN, ... )
```

where it is advisable to always fix the constants in a CML to one.

7 Input/Output and (post-)Processing Options

Three main output directories are required for ASC2D. These in turn store the solution files, the visualization files and the processed files. The solution files are stored in the data/ directory and are used as restart files. The visualization files are stored in anim/ and are used to visualize data in ParaView or the VTK library. As for the processed files, these are located in the proc/ and are case-specific. The required file names are input using

```
OUTPUT_SOL_FILENAME
OUTPUT_VTK_FILENAME
RESTART_FILENAME
PROCESSED_DATA_FILENAME
```

All the above can be written in both **binary** and **ASCII** format. The options to do so are found in

```
VTK_FILE_FORMAT              = BINARY/ASCII
RESTART_SOLUTION_FILE_FORMAT = BINARY/ASCII
```

Note, given how memory demanding a discontinuous Galerkin discretization can be, it is advised to utilize binary format always.

In order to control the writing frequency of both the solution files and the visualization files, the user can specify after how many iterations to write these files via WRITE_VTKDATA_FREQ and WRITE_RESTART_FREQ.

When writing the VTK data, the user has the flexibility to consider different parameters, such as velocity, vorticity, specific entropy, temperature, pressure and Mach number. Note, the default option is to write the conservative variables only. To include any of these as well, the option needed is in

```
VTK_VARIABLE_WRITE = ( PRESSURE, VELOCITY, VORTICITY, MACH, TEMPERATURE, ENTROPY )
```

However, do keep in mind that this will require some additional computational time and memory.

Additionally, the user is capable of specifying after how many iterations does the solver output the progress on screen via OUTPUT_FREQ. Moreover, if Gnuplot is available, the user is able to visually monitor the progress in runtime on a figure. This is optional and can be specified through the input

```
WRITE_GNUPLOT = true/false
```

and its corresponding file can be saved in the location specified through

```
GNUPLOT_FILENAME = gnuplot/log
```

where it is advised to save the file in the gnuplot/ directory.

In order to use the gnuplot feature, a simple script is also included in each benchmark problem, under the name of peek.gnuplot. To monitor the solution in runtime, simple execute gnuplot peek.gnuplot. This will plot the max Mach number versus time, as the simulation progresses.

The type of processing or post-processing of a simulation can be specified as well. For now, the only options include certain ratios of pressure, velocity, Mach number and different variations of wave amplitudes. This is a fairly technical process, which the user can take a thorough look at in the source files. However, in its current form, these types of processing are specified in

```
PROCESS_DATA = NOTHING/RATIO_P_U/RATIO_P_M/RATIO_WM_WP/RATIO_LM_LP/WAVE_AMPLITUDE_ENTROPY
```

where they also require the location of where this processing takes place. Currently, five options are available: xmin, xmax, ymin or ymax boundaries or entire domain. The option for the specification of the processing location is done via

```
PROCESS_LOCATION = XMIN/XMAX/YMIN/YMAX/DOMAIN
```

The user may also opt for a specific probe to sample certain variables from. The option to do so is input via

```
PROBE_SPECIFIED = true/false
```

If a probe is selected, then additional information are required. These correspond to the probe file format, its location in the domain as well as the variables to collect. These options can be specified via the inputs below

```
PROCESS_PROBE_FILE_FORMAT = BINARY/ASCII
PROBE_LOCATION = ( x1,y1,  x2,y2, ..., etc. )
PROBE_VARIABLE = ( DENSITY,XMOMENTUM,YMOMENTUM,TOTALENERGY,XVELOCITY,YVELOCITY,PRESSURE )
```

where (x_1, y_1) corresponds to the first probe point, (x_2, y_2) for the second, and so on and so forth.

Furthermore, the user has the option to sample the conservative variable at a given boundary surface explicitly. This can be achieve via

```
SAMPLE_SURFACE_DATA = true/false
```

In case a surface sampling option is selected, then the boundary surface needs to be specified along with the surface sampling directory. These are done via

```
MARKER_SAMPLE_BOUNDARY   = ( XMIN,XMAX,YMIN,YMAX )
SAMPLE_SURFACE_DIRECTORY = surf/
```

where it is advised to keep the directory at `surf/`.

Finally, the last feature that may be useful for computing the norms between different simulations on the same grid is to sample and save the solution of an entire zone. This feature is specified in `SAMPLE_ZONE_DATA` by either `true` or `false`. Consequently, the location of the sampled zone file is stored via the option

```
ZONE_DATA_FILENAME = zone/sample
```

where it is advised to keep the directory at `zone/`.

The sampled zone is usually taken as the physical domain, although it is up to the user to decide which via the option `MARKER_ZONE_DATA`. The frequency of the sampling is given in `ZONE_DATA_WRITE_FREQ`, which determines after how many temporal iterations does the writing commence. Keep in mind that this can be a very memory intensive procedure. That said, the user has the flexibility to determine the output file format written. Similar to the previous options, this also can be written in binary or ASCII format, through the option `ZONE_FILE_FORMAT`. Note, due to the enormous memory restrictions with the discontinuous Galerkin method, it is advisable to use binary format.

Also, as a final remark, keep in mind that the binary format retains the actual accuracy of the solution and are much more memory and computation efficient that ASCII forms.

References

Kazuki Maeda and Tim Colonius. A source term approach for generation of one-way acoustic waves in the euler and navier–stokes equations. *Wave Motion*, 75:36–49, 2017.

Jan Nordström and Andrew R Winters. Stable filtering procedures for nodal discontinuous galerkin methods. *Journal of Scientific Computing*, 87(1):1–9, 2021.

Edmond K Shehadi and Edwin TA van der Weide. Non-Reflecting Boundary Conditions in the Context of the Discontinuous Galerkin Method. *Available at SSRN 4155198*, 2022. DOI: https://dx.doi.org/10.2139/ssrn.4155198. URL `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4155198`.