NS2DDV documentation

Contents

List of Tables

1 About NS2DDV

1.1 License terms

The toolbox NS2DDV is released under the terms of GNU-GPL version 3 license terms. These terms are recalled here:

1.2 The authors

1.2.1 Current development team

Alexandre Mouton (lead developer) is a research engineer in scientific computing and works in Laboratoire Paul Painlevé (CNRS & Université de Lille - France).
He is in charge of the code maintenance, its development, validation and diffusion.

Caterina Calgaro (lead developer) is an assistant professor in applied mathematics in Laboratoire Paul Painlevé (CNRS & Université de Lille - France) and a member of project-team Inria-RAPSODI (Inria Lille Nord-Europe, France).
She is in charge of the development of new finite volume methods, finite element methods, and applications to physics.

Emmanuel Creusé (lead developer) is a full professor in applied mathematics in Laboratoire de Mathématiques et leurs Applications de Valenciennes (LAMAV) (Université Polytechnique Hauts-de-France - France) and an associate member of project-team Inria-RAPSODI (Inria Lille Nord-Europe, France).
He is in charge of the development of new finite element methods, finite element methods, and applications to physics.

1.2.2 Past developers

- Thierry Goudon
- Roberta Titarelli

- Guillaume Lepoutère
- Francesca Miscioscia
- Alessandro Mozzato
- Maxime Gobert
- Lydéric Debusschère

1.3 Fundings

Inria Lille Nord-Europe is the French National Institute for computer science and applied mathematics based in several french towns. Research at Inria is organised in project teams which bring together researchers with complementary skills to focus on specific scientific projects.
The development of NS2DDV code is partially funded by the project team Inria-RAPSODI located in Inria-Lille Nord-Europe (Villeneuve d'Ascq - France).

Laboratoire Paul Painlevé (CNRS & Université de Lille) is a french laboratory for pure and applied mathematics located in Villeneuve d'Ascq (France). This laboratory is a Mixed Research Unit of CNRS (Centre National de la Recherche Scientifique) and Université de Lille, and is structured in 5 research teams with full-time researchers, research professors and engineers.
The development of NS2DDV code is partially funded by the AN-EDP research team of Paul Painlevé Laboratory.

Laboratoire de Mathématiques et leurs Applications de Valenciennes (Université Polytechnique Hauts-de-France is a french laboratory for pure and applied mathematics located in Valenciennes (France). This laboratory is a research unit of Université Polytechnique Hauts-de-France, and is structured in 4 research tems with full-time researchers, research professors and engineers.

1.4 History

The NS2DDV code has been initially developed by the team SIMPAF (SImulation et Modèles pour les PArticules et les Fluides) of Inria Lille Nord-Europe in cooperation with the Paul Painlevé mathematics laboratory by following the ideas from a paper of E. Creusé, C. Calgaro and T. Goudon published in 2008. The initial purpose of NS2DDV was to apply Finite Element-Finite Volume hybrid methods for 2D incompressible Navier-Stokes equations with variable density.

With this first success, several physical tests and Matlab features have been added to the computational kernel, transforming the initial academic code into a free Matlab toolbox dedicated to viscous fluids under GNU-GPL License Terms, and the release of

version 1.0 has been distributed in November 2011. In addition, the NS2DDV toolbox has been the main subject of the french popular science movie Avis de Recherche.

Thereafter, an important work has been led for restructuring the NS2DDV toolbox in order to embed new models, test cases, numerical methods, and parallelization features as easily as possible. This work led to the version 2.0 of NS2DDV.

2 Recent changes - release notes

2.1 NS2DDV 2.0

The version 2.0 of NS2DDV has been released in November 2018 and brings many new features compared to version 1.0:

- Some tasks can be parallelized with Matlab Parallel Computing Toolbox such as finite element matrices assembling.
- Outputs files can be generated under HDF5 file format if a recent version of Matlab is used. Such format are more convenient for post-processing treatments.
- Some Python 3 routines have been added for post-processing such as movie encoding or signal studies.
- The setup generator has been simplified in order to be run in no-display mode.
- Each numerical method that is implemented has been validated with convergence analysis.
- NS2DDV can be run in batch mode by disabling all graphical features.
- The use of external mesh files has been simplified. It is now possible to import some meshes that have been generated with Gmsh.
- The Navier-Stokes model with constant density (`NS`) has been added.

In addition, the structure of NS2DDV has been slightly modified to be easily scalable by including new numerical methods, models, or test cases. Any user can do it under the license terms.

3 Getting started

3.1 Installation prequisities

As NS2DDV is a simulation code written in Matlab, a Matlab distribution is mandatory to run it. Since it has been developed with Matlab R2007a and newer versions, using these versions may be sufficient to run NS2DDV core code.

In addition of a Matlab distribution, some Matlab toolboxes can be used to improve the NS2DDV performances:

- Matlab Parallel Computing Toolbox (optional): this toolbox allows to speed up the execution of many for loops within the code. To check if you are provided with this toolbox, type the following Matlab command:

```
>> ver distcomp
```

- Matlab Partial Differential Equations Toolbox (optional): this toolbox provides some unstructured mesh generation solution for NS2DDV. To check if you are provided with this toolbox, type the following Matlab command:

```
>> ver pde
```

If Matlab R2011a or newer is used, NS2DDV can write the numerical results under HDF5 format supplemented with Xdmf description files. Such files can be read by visualization softwares like VisIt.

3.2 Installation of NS2DDV

To install NS2DDV, you just need to download the archive containing the code and unzip it in the directory you want.

3.3 Main features

NS2DDV is a Matlab toolbox dedicated to the resolution of 2D Navier-Stokes equations for simulating incompressible fluids. The code embeds the following features:

- Preparation of a test case by modifying a pre-built parameter file,
- Parallelization of the computations (up to the presence of the Matlab Parallel Computing Toolbox),
- Simulations on 2D structured meshes, unstructured meshes and external mesh files,

- Use of second order MUSCL finite volume techniques for solving the density equation,
- Use of parallelized $\mathbb{P}_2-\mathbb{P}_1$ or $\mathbb{P}_{1,b}-\mathbb{P}_1$ finite element methods for solving the velocity-pressure equations,
- Several first order and second order time semi-discretizations,
- Convergence and/or stability analysis on manufactured analytical test cases,
- In-situ visualization or data reading visualization,
- Batch mode run (except the test case preparation step),
- Animated results encoding.

The code is structured in order to make it easily enriched by any user. In such case, the user shall read carefully the License Terms and the NS2DDV Developer's Guide for understanding what (s)he can or must do in order to contribute.

3.4 Running an example

3.4.1 Generating a setup file

The first step consists in starting Matlab in interactive mode. Once it is done, the user must go in the root path of NS2DDV and run the routine `generate_setup_file` as follows:

```
>> generate_setup_file('my_setup_file.m')
```

The unique string argument of this routine is the output setup file to be produced. Running this command will start a short Questions & Answers session where the user has to specify the outlines of the simulation to be run. Note that each answer must be written as a string with simple quotes (example: `'blabla'`).

1. **Cluster profile:** choose the pre-built Matlab cluster profile (see Section 9.1 for the details).
2. **Model:** choose between `'NS'` or `'NSDV'` model (see Section 5 for the details). This choice will give the access to a specific set of test cases.
3. **Test case:** choose a test case (see Section 5 for the details). This chose will give the access to specific domain geometries and/or mesh designs.
4. **Convergence analysis (optional):** if the chosen test case involves an analytic solution, the user may be asked if (s)he wants to perform a convergence analysis.
5. **Domain geometry (optional):** according to the chosen test case, the user may have to choose a domain geometry (see the description of the test cases in Section 5).

6. **Finite volume method (optional):** if the `NSDV` model has been chosen at step 2, the user is asked for a finite volume method family.

7. **Parallelization solution:** choose a parallelization solution (serial or parallelization with Matlab Parallel Computing Toolbox).

3.4.2 Modifying the setup file

Once the execution of `generate_setup_file` is finished, a setup file is generated and can be opened in a text editor for modification by the user. However, some rules should be respected for avoiding a premature crash of the code:

- Since the whole list of parameters has been entirely defined according to the choices that have been done during the execution of `generate_setup_file`, the user must not erase any line in the setup file but can only modify the parameters values.

- The first part of the setup file should not be modified since it somehow defines the structure of the second part of the file. The parameter values can be modified in this second part. Do not hesitate to read the comments and this documentation for ensuring the compatibility of all parameters.

- Every string-type variable within the setup file must be written with simple quotes (example: `'blabla'`). Using double quotes (example: `"blabla"`) should be avoided as much as possible.

The modifiable part of any setup file is organized in several paragraphes:

- Domain parameters: see paragraph 4.1 for the details.

- Physical parameters: some parameters related to the test case and physical constants are gathered in this paragraph. See paragraph 5 for the details.

- External initialization parameters: if the user wants to initialize the simulation with data that are already discretized, a set of specific parameters are dedicated to this optional feature. See paragraphs 5.2.2 and 5.1.2 for more details.

- Mesh generation and renumbering parameters: see paragraph 4.2 for the details.

- Boundary conditions parameters for the velocity: see the test case descriptions in Section 5 for the details.

- Finite element parameters: see the test case descriptions in Section 5 for the details about the pressure constrain and Section 7.2 for general informations about the available finite element methods.

- Finite volume parameters (only when NSDV model has been selected): see Section 7.3 for the details about the available finite volume methods.

- Parallelization parameters: see Section 9 for the details.
- In-situ visualization parameters: see paragraph 8.1 and Tables 4-5 for the details.
- Output results parameters: see paragraph 8.2 and Tables 2-3 for the details.
- Backup and resuming parameters: see Section 9.2 for the details.

3.4.3 Running NS2DDV

After having finished the modification of the setup file, the user can run the associated simulation with the following command:

```
>> start_ns2ddv('my_setup_file.m')
```

If the parameters within the setup file are carefully chosen, NS2DDV will perform the following tasks:

- In any case, NS2DDV creates a log file and completes it at each time step. This log file should be almost identical as the informations printed in the Matlab user interface. It can be used for studying the time evolution of many parameters such as the kinetic energy, the total mass... See Section 8.3 for additional information about log file analysis.
- If the user asks for them, NS2DDV produces output files that store 2D results (velocity, pressure, mass and their derivatives). These files are produced either under `.mat` or `.h5` format according to the Matlab version that is used. One can use Matlab or Python-Matplotlib solutions for visualizing the contents of these files and their time dynamics. See paragraphs 8.2 and 8.3 for the details.
- If the user asks for them, NS2DDV produces some backup files that can be used for resuming the simulation. Additional information about it can be found in paragraph 9.2.

If the output and backup directories contain some files that can be in conflict with the files that should be produced by the simulation, NS2DDV stops and warns the user about this conflict. To solve this problem, the user has two solutions:

- Delete the old files that are characterized by `PARAMETERS.OUTPUT.DIRECTORY_NAME`, `PARAMETERS.OUTPUT.FILE_NAME` and `PARAMETERS.BACKUP.DIRECTORY_NAME`.
- Run the routine `clean_outputs` as follows:

```
>> clean_outputs('my_setup_file.m')
```

3.4.4 Visualization of results

As it has been mentioned in the paragraph above, NS2DDV produces several "raw" files whose contents can be plotted. Four solutions are proposed for plotting these data and saving the graphical outputs. They can answer almost the same requests which are:

- Plot and save some 2D results at a fixed time step with isolines and/or pseudocolors, by loading a single file,
- Plot and save the meshes that are used for spatial discretization of velocity, pressure and mass,
- Plot and save the time evolution of a 2D diagnostic as a movie,
- Plot and save the time evolution of scalar diagnostics such as the kinetic energy, the total mass, etc...

The list of 2D diagnostics that can be plotted are referenced in Tables 4-5 (pp. 96-97).

The first one is an embedded Matlab solution that is specifically dedicated to NS2DDV and should reproduce the same graphs (as far as we tested them) as if they were plotted in a *in-situ* way. It is based on a set of Matlab routines that are parts of NS2DDV sources. To use them, the user must open Matlab in graphical mode, move to the NS2DDV root path, then run the following routine:

```
>> load_paths()
```

Once this routine has been used, it is possible to use the routine `plot_from_file` for selecting some diagnostics in a single `.mat`/`.h5` file, visualizing them and saving the output figure(s) in PNG format. It is also possible to visualize the dynamics within a file set with the routine `movie_from_file`. See paragraphes 8.3.1 and 8.4.1 for the details.

The last visualization solution is based on Python-Matplotlib and is a "no-display" solution, meaning that the visualization can be run without any display server and/or screen contrary to the Matlab solution above. This solution is recommended if the visualization of results has to be scripted and done in batch mode. The main drawback is that this visualization does not embed any graphical user interface, so Python skills are required.

As with classical Python programs, the Python-Matplotlib solution is based on a set of Python modules. It is possible to plot 2D diagnostics such as the velocity components or the pressure, but also to plot the time evolution of the quantities that are stored in each log file such as the kinetic energy, the total mass etc...

For example, in order to plot the time evolution of the kinetic energy stored in the file `diags_log`, the user can write the following Python script:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from plot_overtime_log import *

namelogfile = './diags_log'
[headers, data] = read_logfile(namelogfile)
time = extract_data(headers, data, 'TIME')
ekin = extract_data(headers, data, 'KINETIC ENERGY')
ylim = 'default'
plot_data([ekin], [time], ['Kinetic energy'], 'Kinetic energy', \
   ylim, './ekin.png')
```

Once this script is implemented, the user has to execute it. With Linux or MacOS distribution, and assuming that the script file has been named `myscript.py`, it can be done by opening a terminal and typing

```
$ python myscript.py
```

3.4.5 "Ready to use" examples

Some setup files examples have been prepared for discovering the features of NS2DDV[1]. These files are gathered in the subdirectory `EXAMPLES` in the root path of NS2DDV. For running the first example of the list below, the user should start Matlab with the graphical interface and move to the root path of NS2DDV, then type:

```
>> start_ns2ddv('./EXAMPLES/test_nona.m')
```

For running a Matlab visualization script (ex: `movie_nona.m`), the user should move to the root path of NS2DDV, then type:

```
>> run('./EXAMPLES/movie_nona.m')
```

For running a Python-Matplotlib visualization script (ex: `movie_nona.py`, the user should open a terminal, move to the root path of NS2DDV, then type

```
$ cd EXAMPLES
$ python movie_nona.py
```

All setup files and visualization scripts have been written and commented such that the user should be able to edit them and modify some parameters.

The available "ready-to-use" examples are the following:

[1] These setup files are not specifically adapted for producing highly accurate results.

- `test_nona.m`: this example is focused on the serial simulation of the `NONA` test case (see paragraph 5.2.7) with a structured space mesh and the time domain $[0, 1]$. Several visualization scripts are proposed for this example:
  - `plot_single_nona.m` and `plot_single_scalar_nona.py` are respectively Matlab and Python-Matplotlib solutions for generating 2D plots from a single output file. Note that these outputs files (`.mat` or `.h5` format) are stored in the directory `RESULTS/EXAMPLES/test_nona`.
  - `movie_nona.m` and `movie_nona.py` are Matlab and Python-Matplotlib solutions for plotting the time dynamics within the simulation. Note that the script `movie_nona.m` does not save the produced animation, contrary to the script `movie_nona.py` that saves the result in `RESULTS/EXAMPLES/test_nona`.
- `test_pois.m`: this example is focused on the serial simulation of a Poiseuille flow in a rectangular space domain (see paragraph 5.2.5), with a structured space mesh and the time domain $[0, 0.01]$. In this example, a Minimum Degree Algorithm renumbering procedure is considered for improving the linear system resolution performances. Some visualization scripts in Matlab and Python are proposed as for the `NONA` example above:
  - `plot_single_pois.m` and `plot_single_scalar_pois.py` are respectively Matlab and Python-Matplotlib solutions for generating 2D plots from a single output file. Note that these outputs files (`.mat` or `.h5` format) are stored in the directory `RESULTS/EXAMPLES/test_pois`.
  - `movie_pois.m` and `movie_pois.py` are Matlab and Python-Matplotlib solutions for plotting the time dynamics within the simulation.
- `test_rtin.m`: this example is focused on the serial simulation of a Rayleigh-Taylor instability (see paragraph 5.1.5), with a structured space mesh and the time domain $[0, 0.2]$.
  Some visualization scripts in Matlab and Python are proposed as for the `NONA` and `POIS` examples above.
  - `plot_single_rtin.m` and `plot_single_scalar_rtin.py` are respectively Matlab and Python-Matplotlib solutions for generating 2D plots from a single output file. Note that these outputs files (`.mat` or `.h5` format) are stored in the directory `RESULTS/EXAMPLES/test_rtin`.
  - `movie_rtin.m` and `movie_rtin.py` are Matlab and Python-Matplotlib solutions for plotting the time dynamics within the simulation.

  Two variant of this test are also available:
  - `test_rtin_pct.m`: the same test as `test_rtin.m` but with the use of Matlab Parallel Computing Toolbox for speeding up the computations.

  - `test_rtin_pdet.m`: the same test as `test_rtin.m` but with the use of Matlab Partial Differential Equations Toolbox for generating unstructured meshes.

- `step_neumann.m` and `step_natural.m`: two tests dedicated to the simulation of a Poiseuille input flow in a backward-facing step shaped domain (see Figure 1, paragraphes 4.1.2 and 5.2.5). The main purpose of this couple of test cases is to compare the impact of pseudo-traction (or Neumann) and absorbing (or natural) boundary conditions on the outlet boundary.
  Indeed, both setup files `step_neumann.m` and `step_natural.m` are almost identical and differ on the value of the parameters listed in table 1.

Parameter name	Value in `step_neumann.m`
PARAMETERS.FE.BC_RIGHT_UX	'NEUMANN'
PARAMETERS.FE.BC_RIGHT_UY	'NEUMANN'
PARAMETERS.OUTPUT.DIRECTORY_NAME	'./RESULTS/EXAMPLES/step_neumann'
PARAMETERS.BACKUP.DIRECTORY_NAME	'./BACKUP/EXAMPLES/step_neumann'

Parameter name	Value in `step_natural.m`
PARAMETERS.FE.BC_RIGHT_UX	'NATURAL'
PARAMETERS.FE.BC_RIGHT_UY	'NATURAL'
PARAMETERS.OUTPUT.DIRECTORY_NAME	'./RESULTS/EXAMPLES/step_natural'
PARAMETERS.BACKUP.DIRECTORY_NAME	'./BACKUP/EXAMPLES/step_natural'

Table 1: Parameters for "ready-to-use" setup files `step_neumann.m` and `step_natural.m`

  This couple of test cases is also an example of external mesh file import.
  In order to highlight this comparison, the following visualization scripts are provided in the subdirectory `EXAMPLES`:

  - `plot_single_scalar_step_neumann.py`: a Python-Matplotlib script for plotting and saving a scalar 2D result (here, it is the x-component of velocity) by reading a single `.h5` file from the simulation that has been previously run by using `step_neumann.m`.
  - `plot_single_scalar_step_natural.py`: a Python-Matplotlib script that does the same job as `plot_single_scalar_step_neumann.py` but for the simulation run by using the setup file `step_natural.m`.
  - `plot_single_vector_step_neumann.py`: a Python-Matplotlib script that does the same job as `plot_single_scalar_step_neumann.py` but for plotting a vector field such as the 2D velocity or the pressure gradient.
  - `plot_single_vector_step_natural.py`: a Python-Matplotlib script that does the same job as `plot_single_vector_step_neumann.py` but for the simulation run by using the setup file `step_natural.m`.

– `plot_two_scalar_step.py`: a Python-Matplotlib script for comparing two simulations in the same terms. More precisely, the aim of this script is to plot on the same graph the isovalues of a 2D scalar result (here, we chose the x-component of the velocity) at the same time step in order to compare two simulations that are in competition.

4 Space domain geometries and meshes

From now, we consider the following notations:

- $\mathbf{x}$ stands for the space variable,
- Ω is a bounded connected 2D domain with boundary denoted with $\partial\Omega$,
- $\partial\Omega$ can be decomposed into 3 distinct subparts $\partial\Omega_0$, $\partial\Omega_D$ and $\partial\Omega_N$,
- For any bounded subset $A \subset \mathbb{R}^2$, $\mathbf{n}_A$ is the outward normal unit vector to ∂A, and $\boldsymbol{\tau}_A$ is the unit tangent vector to ∂A such that $\det(\mathbf{n}_A, \boldsymbol{\tau}_A) = 1$.

4.1 Available space domain geometries

We describe in this section the available geometries in NS2DDV. According to the chosen test case, some of them can be used and the user must choose one of them during the execution of `generate_manual_setup`. The value of `PARAMETERS.DOMAIN.GEOMETRY` will be accordingly fixed and the user will be authorized to modify some parameters for resizing the space domain.

4.1.1 `RECTANGLE`

Considering $x_{\min} < x_{\max}$ and $y_{\min} < y_{\max}$, we define the space domain Ω as the 2D rectangle $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$.

```
PARAMETERS.DOMAIN.XMIN = xmin;
PARAMETERS.DOMAIN.XMAX = xmax;
PARAMETERS.DOMAIN.YMIN = ymin;
PARAMETERS.DOMAIN.YMAX = ymax;
```

4.1.2 `STEP`

We consider $x_{\min} < x_{\text{step}} < x_{\max}$ and $y_{\min} < y_{\text{step}} < y_{\max}$. Ω is defined a channel with a facing step.

```
PARAMETERS.DOMAIN.XMIN = xmin;
PARAMETERS.DOMAIN.XMAX = xmax;
PARAMETERS.DOMAIN.YMIN = ymin;
PARAMETERS.DOMAIN.YMAX = ymax;
PARAMETERS.DOMAIN.XSTEP = xstep;
PARAMETERS.DOMAIN.YSTEP = ystep;
```

At present time, there is a unique configuration that is available:

- **Backward-facing step:** the step is located on the left part of the domain (see Figure 1), *i.e.* Ω is defined as

$$\Omega = ([x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]) \setminus ([x_{\min}, x_{\text{step}}[\times [y_{\min}, y_{\text{step}}[) \ .$$

This configuration can be chosen in the setup file with the following parameter:

```
PARAMETERS.DOMAIN.STEP_POSITION = 'LEFT';
```

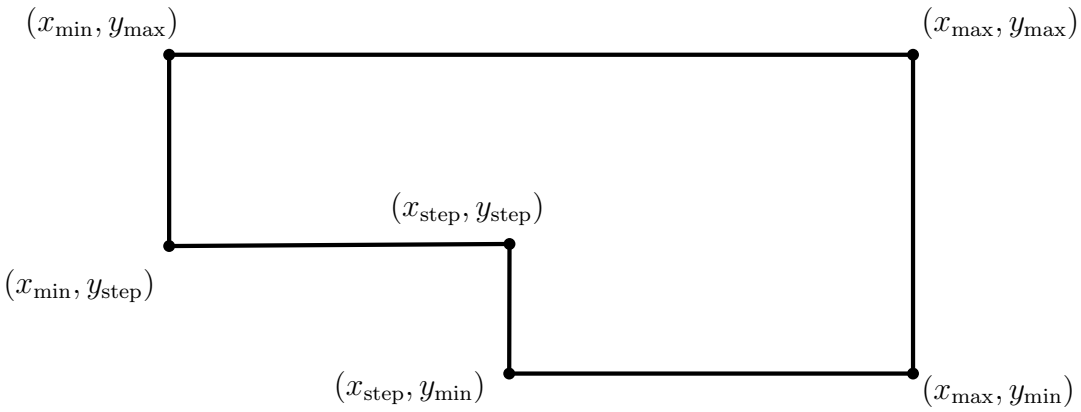

Figure 1: Channel with backward-facing step

4.1.3 DIHEDRON

We consider $x_{\min}, x_{\max}, y_{\min}, y_{\max}, x_{A_1}, y_{A_1}, x_{A_2} \in \mathbb{R}$ such that

$$x_{\min} < x_{A_1} < x_{A_2} < x_{\max} \ , \qquad y_{\min} < y_{A_1} < y_{\max} \ .$$

Ω is set as $\overline{([x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]) \setminus C_D}$ where C_D is the convex hull of the points $(x_{\min}, y_{\min})$, $(x_{A_2}, y_{\min})$, (x_{A_1}, y_{A_1}) and $(x_{\min}, y_{A_1})$ (see Figure 2).
This domain configuration can be modified with the following setup parameters:

```
PARAMETERS.DOMAIN.XMIN = x_min;
PARAMETERS.DOMAIN.XMAX = x_max;
PARAMETERS.DOMAIN.YMIN = y_min;
PARAMETERS.DOMAIN.YMAX = y_max;
PARAMETERS.DOMAIN.XAXIS_ANGLE1 = x_A1;
PARAMETERS.DOMAIN.YAXIS_ANGLE1 = y_A1;
PARAMETERS.DOMAIN.XAXIS_ANGLE2 = x_A2;
```

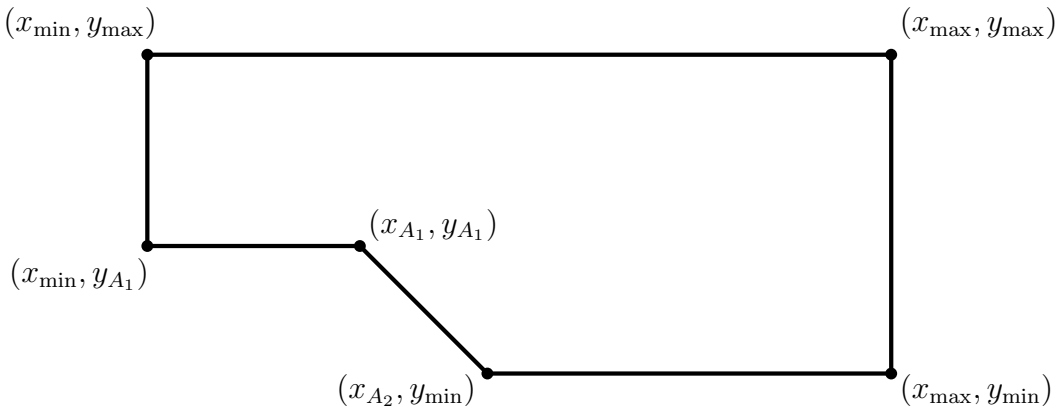

Figure 2: Dihedron-like domain

4.2 Mesh generation solutions

According to the chosen domain geometry (see paragraph 4.1), NS2DDV proposes up to 3 ways to generate a mesh that is constituted of triangular cells.

- `PARAMETERS.MESH.GENERATION = 'NS2DDV'` (only for `RECTANGLE` and `DIHEDRON` geometries): a method for generating "almost regular" meshes that is included in NS2DDV source code. If such method is chosen in the setup file, the user shall precise the value of some additional parameters to complete the characterization of the mesh that will be generated. These parameters depend on the chosen domain geometry kind and are discussed in the following lines:
  - `RECTANGLE` geometry: the mesh is completely described by 3 parameters that are
    - `PARAMETERS.MESH.NBSEG_X`: the number $n_{e,x}$ of edges in x-direction. It defines the x-component of the nodes as

      $$x_i = x_{\min} + \frac{i}{n_{e,x}}(x_{\max} - x_{\min}),$$

      with $i = 0, \ldots, n_{e,x}$.
    - `PARAMETERS.MESH.NBSEG_Y`: the number $n_{e,y}$ of edges in y-direction. It defines the y-component of the nodes as

      $$y_j = y_{\min} + \frac{j}{n_{e,y}}(y_{\max} - y_{\min}),$$

      with $j = 0, \ldots, n_{e,y}$.
    - `PARAMETERS.MESH.TRIANGLES_ORIENTATION`: the main pattern of the mesh. This parameter must be set to `'DIAGONAL'` or `'CROSS'` (see Figure 3).
  - `DIHEDRON` geometry: the mesh is described thanks to 6 additional parameters:

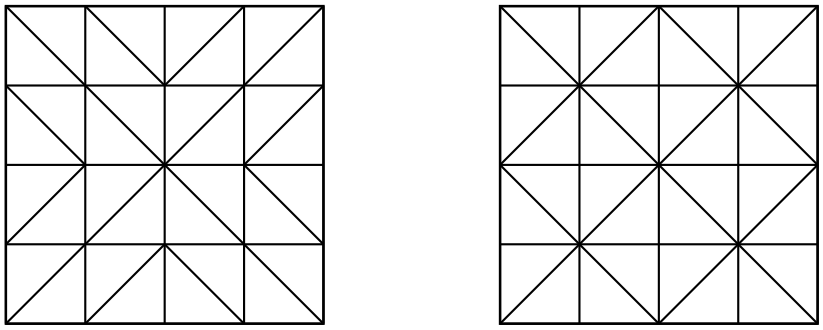

Figure 3: Mesh pattern on a rectangular domain with `PARAMETERS.MESH.TRIANGLES_ORIENTATION = 'DIAGONAL'` (left) and with `PARAMETERS.MESH.TRIANGLES_ORIENTATION = 'CROSS'` (right)

* `PARAMETERS.MESH.HEIGHT_BL`: the height of boundary layer h_{BL} in the lower part of the domain. The default value is set to 0.2 and should remain stricly positive.
* `PARAMETERS.MESH.NBNODES_INBL_Y`: the number of nodes $n_{n,y,BL}$ in y-direction in the boundary layer.
* `PARAMETERS.MESH.PROG_GEOM_BL_Y`: the geometric reason α that describe the y-component of the nodes in the boundary layer. More precisely, these nodes have coordinates of the form $(x_i, y_{j,BL})$ with

$$y_{j,BL} = \begin{cases} y_{A_1} + \dfrac{j^\alpha\, h_{BL}}{(n_{n,y,BL}-1)^\alpha}, & \text{if } x_i \in [x_{\min}, x_{A_1}], \\ y_{A_1} + \dfrac{x_i - x_{A_1}}{x_{A_2} - x_{A_1}}(y_{\min} - y_{A_1}) + \dfrac{j^\alpha\, h_{BL}}{(n_{n,y,BL}-1)^\alpha}, & \text{if } x_i \in [x_{A_1}, x_{A_2}], \\ y_{\min} + \dfrac{j^\alpha\, h_{BL}}{(n_{n,y,BL}-1)^\alpha}, & \text{if } x_i \in [x_{A_2}, x_{\max}], \end{cases}$$

  with $j = 0, \ldots, n_{n,y,BL} - 1$.
* `PARAMETERS.MESH.NBSEG_OUTBL_Y`: the number $n_{e,y,OBL}$ of edges in y-direction outside of the boundary layer. This defines the node coordinates $(x_i, y_{j,OBL})$ outside of the boundary layer as follows

$$y_{j,OBL} = \begin{cases} y_{A_1} + h_{BL} + \dfrac{j}{n_{e,y,OBL}}, & \text{if } x_i \in [x_{\min}, x_{A_1}], \\ y_{A_1} + \dfrac{x_i - x_{A_1}}{x_{A_2} - x_{A_1}}(y_{\min} - y_{A_1}) + h_{BL} + \dfrac{j}{n_{e,y,OBL}}, & \text{if } x_i \in [x_{A_1}, x_{A_2}], \\ y_{\min} + h_{BL} + \dfrac{j}{n_{e,y,OBL}}, & \text{if } x_i \in [x_{A_2}, x_{\max}], \end{cases}$$

  with $j = 0, \ldots, n_{e,y,OBL}$.

* `PARAMETERS.MESH.NBSEG_PERUNIT_X`: the number $n_{e,x,BU}$ of edges by unit in x-direction. This defines the x-coordinates of the nodes as

$$x_i = x_{\min} + \frac{i}{(x_{\max} - x_{\min})\, n_{e,x,BU}},$$

with $i = 0, \ldots, (x_{\max} - x_{\min})\, n_{e,x,BU}$.

* `PARAMETERS.MESH.TRIANGLES_ORIENTATION`: the main pattern of the mesh. This parameter must be set to `'DIAGONAL'` or `'CROSS'` (see Figure 3).

- `PARAMETERS.MESH.GENERATION = 'PDET'`: this mesh generation method uses Matlab Partial Differential Equations Toolbox to provide an unstructured triangulation of the space domain. The user must be sure that the Matlab distribution (s)he is using embeds this toolbox and can verify it with the following Matlab command:

```
>> ver pde
```

If this toolbox can be used, the user just need to precise the characteristic edge length h_0 in the setup file as follows:

```
% Characteristic edge length in the mesh
PARAMETERS.MESH.H0 = h0;
```

- `PARAMETERS.MESH.GENERATION = 'FROM_FILE'`: with this method, NS2DDV loads the mesh nodes and triangles from an external file. Two mesh formats are allowed:
  - A unique file in `.msh` format. Such file can be generated with the mesh generation software Gmsh.
  - A couple of files with `_t1`/`_p1` format. Such files are already provided with NS2DDV source file: see the contents of directory `MESH/MESH_FILES`. The `_t1` file contains the characterization of the triangles and the `_p1` file contains the node coordinates.

To include such mesh files, the user must modify the setup file as in the following examples:

```
% Mesh source file(s) : .msh or _p1/_t1 format allowed
% {'xxxx.msh'}           cell array of size 1 for .msh format
% {'xxxx_p1', 'xxxx_t1'} cell array of size 2 for _p1/_t1 format
PARAMETERS.MESH.SRC_FILES = {'./MESH/MESH_GMSH/step.msh'};
```

or

```
% Mesh source file(s) : .msh or _p1/_t1 format allowed
% {'xxxx.msh'}           cell array of size 1 for .msh format
% {'xxxx.p1', 'xxxx_t1'} cell array of size 2 for _p1/_t1 format
PARAMETERS.MESH.SRC_FILES = ...
    {'./MESH/MESH_FILES/mesh_rect_rtin_p1', ...
     './MESH/MESH_FILES/mesh_rect_rtin_t1'};
```

Independently of the chosen mesh generation method, NS2DDV embeds some node renumbering features in order to speed up the linear system inversion. More precisely, it is possible to use the Minimum Degree Algorithm or the Cuthill-McKee Algorithm[2] for reducing the bandwidth of the matrices that are involved in the finite element methods (see paragraph 7.2). The user can activate such feature by modifying the following lines in the setup file:

```
% Node renumbering method
% 'NONE'  No renumbering
% 'AMD'   Renumbering with Minimum Degree Algorithm Matlab built-in
%         routines
% 'CMK'   Renumbering with Cuthill-McKee Matlab built-in routines
PARAMETERS.MESH.RENUMBERING = 'NONE';
```

Note that no renumbering procedure is chosen by default.

[2]Matlab R2006a or newer is required for using Cuthill-McKee Algorithm.

5 Models and test cases

In addition of the notations introduced in the previous section, we consider the following ones:

- t stands for the time variable,
- ∂_t is the time derivative,
- ∇, $\nabla\cdot$ and Δ stand for the gradient, divergence and laplacian differential operators in $\mathbf{x}$,
- $[0,T]$ is a time interval with fixed $T > 0$.

5.1 NSDV: Incompressible 2D Navier-Stokes equations with variable density

The Incompressible 2D Navier-Stokes model with variable density, or NSDV model, that is solved by NS2DDV is constituted of the following dimensionless equations:

$$\begin{cases} \partial_t \rho + \nabla \cdot (\rho\, \mathbf{u}) = 0\,, & \text{(5.1a)} \\ \rho\,[\partial_t \mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u}] + \nabla p = \dfrac{1}{Re}\Delta\mathbf{u} + \mathbf{f}\,, & \text{(5.1b)} \\ \nabla\cdot\mathbf{u} = 0\,. & \text{(5.1c)} \end{cases}$$

In these equations, the unknowns are the density $\rho : [0,T]\times\Omega \to \mathbb{R}$, the velocity $\mathbf{u} : [0,T]\times\Omega \to \mathbb{R}^2$ and the scalar pressure $p : [0,T]\times\Omega \to \mathbb{R}$. The characteristic Reynolds number $Re > 0$ and the source term $f : [0,T]\times\Omega \to \mathbb{R}^2$ are given.

5.1.1 Boundary conditions

In the following lines, we consider the following partition of the boundary $\partial\Omega$

$$\partial\Omega = \partial\Omega_D \cup \partial\Omega_N\,, \qquad \partial\Omega_D \cap \partial\Omega_N = \emptyset\,. \tag{5.2}$$

In all test cases, $\partial\Omega_D$ is assumed to be non-empty. In some of them, $\partial\Omega_N$ can be an empty set.

Dirichlet boundary conditions are applied on $\partial\Omega_D$. Such conditions are used to model an inlet velocity applied on a part of the boundary and usually write as

$$\mathbf{u} = \mathbf{u}_D\,, \qquad \text{on } [0,T]\times\partial\Omega_D. \tag{5.3}$$

Concerning the boundary conditions on the free boundary $\partial\Omega_N$, several solutions are currently available in NS2DDV code for the model NSDV: assuming that $\partial\Omega_N \neq \emptyset$ and considering $\Gamma \subset \partial\Omega_N$ (we can have $\Gamma = \partial\Omega_N$ or not), one can consider

- Pseudo-traction (`NEUMANN`) boundary conditions:

$$\frac{1}{Re}\left(\nabla \mathbf{u}\right)\mathbf{n}_\Omega - p\mathbf{n}_\Omega = \mathbf{g}\,, \qquad \text{on } [0,T]\times\Gamma, \tag{5.4}$$

  where $\mathbf{g} : [0,T]\times\Gamma \to \mathbb{R}^2$ is given and such that $\mathbf{g}(t,\cdot) \in \left(H^{1/2}(\Gamma)\right)^2$ for any $t \in [0,T]$.

- Symmetry conditions (`SYMMETRY`):

$$\begin{cases} \mathbf{u}\cdot\mathbf{n}_\Omega = 0\,, \\ \left(\dfrac{1}{Re}\left(\nabla \mathbf{u}\right)\mathbf{n}_\Omega\right)\times\mathbf{n}_\Omega = 0\,, \end{cases} \qquad \text{on } [0,T]\times\Gamma. \tag{5.5}$$

Remark 5.1. If the boundary conditions are Dirichlet and/or symmetry boundary conditions (no pseudo-traction conditions), the inlet velocity $\mathbf{u}_D$ must satisfy the compatibility condition

$$\int_{\partial\Omega} \mathbf{u}_D(t,\cdot)\cdot\mathbf{n}_\Omega\, d\sigma = 0\,, \qquad \forall\, t\in[0,T]\,, \tag{5.6}$$

and the system (5.1)-(5.3) must be supplemented with an additional constrain on the pressure to ensure the existence and the uniqueness of its solution. 2 solutions can be used in NS2DDV:

- Null average pressure: $\displaystyle\int_\Omega p(t,\mathbf{x})\, d\mathbf{x} = 0$ is imposed for any $t\in[0,T]$,
- Null pressure on a specific point: provided with $\mathbf{x}_* \in \Omega$, $p(t,\mathbf{x}_*) = 0$ is imposed for any $t\in[0,T]$.

This choice can be done in a setup file as follows:

```
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_AVERAGE';
```

or

```
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_FIXED_POINT';
```

In addition, it is necessary to fix a value for the point $\mathbf{x}_* = (x_*, y_*)$. This can be done by modifying the following parameters:

```
PARAMETERS.FE.NFX_X = x_*;
PARAMETERS.FE.NFX_Y = y_*;
```

Remark 5.2. If $\partial\Omega_N \neq \emptyset$ and if pseudo-traction conditions (5.4) are considered, the system (5.1)-(5.3)-(5.4) does not require any additional constrain on $\mathbf{u}_D$ or p like in Remark 5.1 to be well-posed.

5.1.2 Initial state

In addition to the boundary conditions, the model (5.1) is completed with an initial density $\rho^0 : \Omega \to \mathbb{R}_+$, an initial velocity $\mathbf{u}^0 : \Omega \to \mathbb{R}^2$ and an initial pressure $p^0 : \Omega \to \mathbb{R}$. NS2DDV embeds two ways to choose the definition of the initial state $(\rho^0, \mathbf{u}^0, p^0)$:

- The initial state is analytically defined according to the chosen test case: in such case, the user can refer to the next paragraphs for the complete definition of $(\rho^0, \mathbf{u}^0, p^0)$.
  At the beginning of the simulation, NS2DDV discretizes $(\rho^0, \mathbf{u}^0, p^0)$ according to the chosen finite element scheme.
  To set this initialization choice, the user must modify the setup file as follows:

```
% Use a third-party file to initialize the simulation
% Allowed formats: '', 'xxx.h5', 'xxx.mat'
% If '' (default value) is selected, the simulation will be
% initialized with the test case parameters
% If a valid file is selected, some domain and physical parameters
% above will be ignored.
PARAMETERS.INIT_FILE = '';
```

- The initial state is already discretized and is obtained from an external `.h5` / `.mat` file: in such case, the analytical definition of $(\rho^0, \mathbf{u}^0, p^0)$ associated to the chosen test case is not taken into account, and is replaced by some data that are already discretized that will be denoted here $(\rho_h^0, \mathbf{u}_h^0, p_h^0)$.
  At the beginning of the simulation, NS2DDV reads the external file that is provided by the user (see below), extracts the fields corresponding the density, the velocity and the pressure, and uses them as initial data.
  To set this initialization choice, the user must modify the setup file as follows (`the_external_file.h5` is replaced by the file the user wants to use):

```
% Use a third-party file to initialize the simulation
% Allowed formats: '', 'xxx.h5', 'xxx.mat'
% If '' (default value) is selected, the simulation will be
% initialized with the test case parameters
% If a valid file is selected, some domain and physical parameters
% above will be ignored.
PARAMETERS.INIT_FILE = 'the_external_file.h5';
```

  We remind that such an external file is associated to a mesh file, some domain geometry parameters, and a time value that will be denoted here t^*. By default, NS2DDV will use these data for the initialization step, meaning that the initial time t^0 within the simulation will be set to t^*, and that the domain and mesh parameters within the setup file such as `PARAMETERS.DOMAIN.XMAX` or `PARAMETERS.MESH.NBSEG_X` will be ignored. However, it is possible to take into account these setup parameters as follows:

– Replace the default mesh: in such case, a new mesh based on the parameters within the setup file is generated and the discrete $(\rho_h^0, \mathbf{u}_h^0, p_h^0)$ is interpolated on this new mesh to get $(\rho_{\tilde{h}}^0, \mathbf{u}_{\tilde{h}}^0, p_{\tilde{h}}^0)$ (Warning: this work can be time consuming). To activate this feature, the user must modify the following lines in the setup file:

```
% Build a new mesh at initialization
% (ignored if PARAMETERS.INIT_FILE = '')
% Allowed values: 'YES' or 'NO' (default)
PARAMETERS.REMESH = 'NO';
```

– Modify the domain geometry: in the case where the domain is remeshed, it is possible to to get rid of the domain geometry that is inherited from the "old" mesh, and to replace it by the domain geometry that is described in the setup file. To activate this feature, the user must activate the remeshing option as above and modify the following lines in the setup file:

```
% Use domain parameters from the third-party initialization
% file or PARAMETERS.DOMAIN.XXX
% (ignored if PARAMETERS.INIT_FILE = '' or if
%  PARAMETERS.REMESH = 'NO')
% 'EXT_FILE'      The domain geometry parameters are imposed by
%                 the contents of PARAMETERS.INIT_FILE
% 'CURRENT_FILE'  Use the variables PARAMETERS.DOMAIN.XXX above
PARAMETERS.WHICH_GEOMETRY = 'EXT_FILE';
```

Remark 5.3. If this feature is activated, the domain geometry kind (`RECTANGLE`, ...) that is chosen in the setup file must be the same as in the external file used for initializing NS2DDV.

– Force the initial time value: if a external initial data is used, it is possible to impose the initial time value t^0 to t^* instead of 0 (default value). To do this, the user must modify the following lines in the setup file:

```
% Reset time variable if a third-party file is used to
% initialize the simulation (ignored if
% PARAMETERS.INIT_FILE = '')
% Allowed values: 'YES' (default) or 'NO'
PARAMETERS.RESET_TIME = 'YES';
```

5.1.3 EXAC: analytical test case with full Dirichlet boundary conditions

This test case is dedicated to the validation of any resolution of NSDV model with full-Dirichlet boundary conditions on velocity. It is associated to an analytical solution and *ad hoc* manufactured source terms and a default Reynolds number $\mathrm{Re} = 1$ (see [4]).

This test case can be run on a rectangular space domain $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$

(see paragraph 4.1.1) with default definition $\Omega = [0,1]^2$. In the case `EXAC`, an analytical solution is set as $(\mathbf{u}_{ex}, p_{ex})$ defined as

$$\begin{aligned} \rho_{ex}(t,x,y) &= 2 + x\,\cos(\sin t) + y\sin(\sin t)\,, \\ \mathbf{u}_{ex}(t,x,y) &= \begin{pmatrix} -y\,\cos t \\ x\,\cos t \end{pmatrix}, \\ p_{ex}(t,x,y) &= \sin t\,\sin\left(\frac{2\pi\,(x-x_*)}{x_{\max}-x_{\min}}\right)\,\sin\left(\frac{2\pi\,(y-y_*)}{y_{\max}-y_{\min}}\right), \end{aligned} \tag{5.7}$$

with a fixed point $\mathbf{x}_* = (x_*, y_*) \in \Omega$ (default values $x_* = \frac{x_{\min}+x_{\max}}{2}$, $y_* = \frac{y_{\min}+y_{\max}}{2}$).

The Dirichlet boundary conditions are set as

$$\mathbf{u}_D = \mathbf{u}_{ex|_{\partial\Omega}}\,, \qquad \text{on } [0,T]\times\partial\Omega\ , \tag{5.8}$$

and the source term $\mathbf{f}$ is manufactured to fit with the exact solution. Note that these boundary conditions need Remark 5.1 to be satisfied with the point $\mathbf{x}_* = (x_*, y_*)$ linked to the exact solution above. Hence, the user should have a look at the following lines of the setup file in order to manage the pressure constrain:

```
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_FIXED_POINT';
PARAMETERS.FE.NFX_X = x*;
PARAMETERS.FE.NFX_Y = y*;
```

5.1.4 EXACNEU: analytical test case with Dirichlet-pseudotraction boundary conditions

- Allowed values for `PARAMETERS.DOMAIN.GEOMETRY`: `'RECTANGLE'`
- Exact solution:

$$\begin{aligned} \rho_{ex}(t,x,y) &= 2 + x\,\cos(\sin t) + y\sin(\sin t)\,, \\ \mathbf{u}_{ex}(t,x,y) &= \begin{pmatrix} \sin\left(\dfrac{x-x_{\min}}{x_{\max}-x_{\min}}\right)\,\sin\left(\dfrac{y-y_{\min}}{y_{\max}-y_{\min}}+t\right) \\ \cos\left(\dfrac{x-x_{\min}}{x_{\max}-x_{\min}}\right)\,\cos\left(\dfrac{y-y_{\min}}{y_{\max}-y_{\min}}+t\right) \end{pmatrix}, \\ p_{ex}(t,x,y) &= \frac{1}{Re(x_{\max}-x_{\min})}\cos\left(\frac{x-x_{\min}}{x_{\max}-x_{\min}}\right)\sin\left(\frac{y-y_{\min}}{y_{\max}-y_{\min}}+t\right), \end{aligned} \tag{5.9}$$

- Boundary conditions:

$$\begin{aligned} &\frac{1}{Re}\mathbf{n}_\Omega\cdot\nabla\mathbf{u} - p\,\mathbf{n}_\Omega = 0\,, && \text{on } [0,T]\times\partial\Omega_N, \\ &\mathbf{u} = \mathbf{u}_{ex}\,, && \text{on } [0,T]\times\partial\Omega_D, \end{aligned} \tag{5.10}$$

with $\partial\Omega_N = \{x_{\min}\}\times]y_{\min}, y_{\max}[$ and $\partial\Omega_D = \partial\Omega\backslash\partial\Omega_N$.

- Source term $\mathbf{f}$ is manufactured to fit with the exact solution above.

5.1.5 RTIN: Rayleigh-Taylor instability

This test case is dedicated to the study of the Rayleigh-Taylor instability that appears when two fluids with different densities are separated by a common interface. The proposed configuration is quite academic and is inspired by [7, 4]: the space domain is rectangular and defined as $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ with default definition $[-0.5, 0.5] \times [-2, 2]$, and both fluids are only submitted to the gravitational force, so the external source term in (5.1) is defined as

$$\mathbf{f} = \rho\,\mathbf{g}\,, \qquad \mathbf{g} = \begin{pmatrix} 0 \\ -g \end{pmatrix}\,, \tag{5.11}$$

where $g > 0$ is the dimensionless gravity constant. This constant is set by default to 9.81 but it can be modified in the setup file with the following lines:

```
% Gravity amplitude (Default = 9.81)
PARAMETERS.PHYSICAL.GRAVITY = g;
```

The Reynolds number Re is set by default to 1000 but can be modified with the following line in the setup file:

```
% Reynolds number
PARAMETERS.PHYSICAL.RE = Re;
```

In order to model two fluids with different densities, we consider two characteristic densities ρ_m and ρ_M such that $\rho_M > \rho_m > 0$ (default values: $\rho_m = 1$, $\rho_M = 7$) and we set the initial state as follows:

$$\left\{\begin{array}{rcl} \mathbf{u}^0(x,y) & = & 0\,, \\ p^0(x,y) & = & 0\,, \\ \rho^0(x,y) & = & \dfrac{\rho_M + \rho_m}{2} + \dfrac{\rho_M - \rho_m}{2} \tanh\left(\dfrac{y - \eta\cos(\frac{2\pi\,x}{x_{\max}-x_{\min}})}{0.01\,(x_{\max}-x_{\min})}\right)\,, \end{array}\right. \tag{5.12}$$

for any $(x, y) \in \Omega$, with η defined as

$$\eta = \left\{\begin{array}{ll} 0.1\,, & \text{if At} = \frac{\rho_M-\rho_m}{\rho_M+\rho_m} > 0.5, \\ 0.01\,, & \text{else.} \end{array}\right. \tag{5.13}$$

One can replace the default values of ρ_m and ρ_M in the setup file by modifying the following lines:

```
% Characteristic density of the light fluid
PARAMETERS.PHYSICAL.RHOMIN = ρ_m;
% Characteristic density of the heavy fluid
PARAMETERS.PHYSICAL.RHOMAX = ρ_M;
```

Finally, the boundary conditions are set by splitting $\partial\Omega$ in two parts $\partial\Omega_D$ and $\partial\Omega_N$ such that

$$\partial\Omega_D = [x_{\min}, x_{\max}] \times \{y_{\min}, y_{\max}\}\,, \qquad \partial\Omega_N = \{x_{\min}, x_{\max}\} \times]y_{\min}, y_{\max}[\,, \tag{5.14}$$

and

$$\begin{cases} \mathbf{u} = 0\,, & \text{on } [0,T] \times \partial\Omega_D, \\ \mathbf{u} \cdot \mathbf{n}_\Omega = 0\,, & \text{on } [0,T] \times \partial\Omega_N, \\ \left(\dfrac{1}{Re}(\nabla \mathbf{u})\,\mathbf{n}_\Omega\right) \times \mathbf{n}_\Omega = 0\,, & \text{on } [0,T] \times \partial\Omega_N. \end{cases} \tag{5.15}$$

Note that these boundary conditions need Remark 5.1 to be satisfied. Hence, the user should have a look at the following lines of the setup file in order to manage the pressure constrain:

```
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_FIXED_POINT';
PARAMETERS.FE.NFX_X = x_*;
PARAMETERS.FE.NFX_Y = y_*;
```

For the test case `RTIN`, the default position of the pressure point $\mathbf{x}_*$ is $(\frac{x_{\min}+x_{\max}}{2}, \frac{y_{\min}+3y_{\max}}{4})$.

5.1.6 DROP: falling droplet

This test case is dedicated to the study of the fall of a "heavy" droplet through a "light" into a "heavy" liquid in a cavity. We mean by "heavy" and "light" fluids some specific subregions of the space domain where the initial density is provided with values that slightly different.

As for the `RTIN` test case (see paragraph 5.1.5), the proposed configuration is inspired by [4]: the considered space domain Ω is rectangular and is set as $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ with default value $\Omega = [0,1] \times [0,2]$ and the involved fluids are only submitted to the gravitational force, so the external source term $\mathbf{f}$ is defined as

$$\mathbf{f} = \rho\,\mathbf{g}\,, \qquad \mathbf{g} = \begin{pmatrix} 0 \\ -g \end{pmatrix}, \tag{5.16}$$

where $g > 0$ is the dimensionless gravity constant. This constant is set by default to 1 but it can be modified in the setup file with the following lines:

```
% Gravity amplitude (Default = 1)
PARAMETERS.PHYSICAL.GRAVITY = g;
```

The Reynolds number Re is set by default to 3132 but can be modified with the following line in the setup file:

```
% Reynolds number
PARAMETERS.PHYSICAL.RE = Re;
```

In order to model two fluids with different densities, we consider two characteristic densities ρ_m and ρ_M such that $\rho_M > \rho_m > 0$ (default values: $\rho_m = 1$, $\rho_M = 100$) and we set the initial state as follows:

$$\begin{cases} \mathbf{u}^0(x,y) &= 0\,, \\ p^0(x,y) &= 0\,, \\ \rho^0(x,y) &= \begin{cases} \rho_M\,, & \forall\,(x,y) \in B_d \cup [x_{\min}, x_{\max}] \times [y_{\min}, y_I]\,, \\ \rho_m\,, & \text{else,} \end{cases} \end{cases} \tag{5.17}$$

for any $(x,y) \in \Omega$, where $B_d \subset \Omega$ defines the initial position of the droplet and $y_I \in]y_{\min}, y_{\max}[$ is the y-component of the interface between heavy and light fluids. B_d is defined as the 2D closed disc $B_d = \{(x-x_d)^2 + (y-y_d)^2 \leq r_d^2\}$ with (x_d, y_d) and $r_d > 0$ such that B_d is strictly included in Ω.

One can replace the default values of ρ_m, ρ_M, (x_d, y_d), r_d and y_I in the setup file by modifying the following lines:

```
% Characteristic density of the light fluid
PARAMETERS.PHYSICAL.RHOMIN = ρm;
% Characteristic density of the heavy fluid
PARAMETERS.PHYSICAL.RHOMAX = ρM;
% Coordinates of the droplet center
PARAMETERS.PHYSICAL.DROP_CENTER_X = xd;
PARAMETERS.PHYSICAL.DROP_CENTER_Y = yd;
% Radius of the droplet
PARAMETERS.PHYSICAL.DROP_RADIUS = rd;
% y-component of the interface between light and heavy fluids
PARAMETERS.PHYSICAL.DROP_INTERFACE_Y = yI;
```

Finally, the boundary conditions are set by splitting $\partial\Omega$ in two parts $\partial\Omega_D$ and $\partial\Omega_N$ such that

$$\partial\Omega_D = [x_{\min}, x_{\max}] \times \{y_{\min}, y_{\max}\}\,, \qquad \partial\Omega_N = \{x_{\min}, x_{\max}\} \times]y_{\min}, y_{\max}[\,, \tag{5.18}$$

and

$$\begin{cases} \mathbf{u} = 0\,, & \text{on } [0,T] \times \partial\Omega_D, \\ \mathbf{u} \cdot \mathbf{n}_\Omega = 0\,, & \text{on } [0,T] \times \partial\Omega_N, \\ \left(\dfrac{1}{Re}(\nabla \mathbf{u})\,\mathbf{n}_\Omega\right) \times \mathbf{n}_\Omega = 0\,, & \text{on } [0,T] \times \partial\Omega_N. \end{cases} \tag{5.19}$$

Note that these boundary conditions need Remark 5.1 to be satisfied. Hence, the user should have a look at the following lines of the setup file in order to manage the pressure constrain:

```
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_FIXED_POINT';
PARAMETERS.FE.NFX_X = x_*;
PARAMETERS.FE.NFX_Y = y_*;
```

For the test case `DROP`, the default position of the pressure point $\mathbf{x}_*$ is $(\frac{x_{\min}+x_{\max}}{2}, \frac{y_{\min}+y_I}{2})$.

5.1.7 CAEN: lid-driven cavity

This test case is dedicated to the simulation of the 2D lid-driven cavity problem. This problem is rather studied in the framework of incompressible Navier-Stokes equation but it can also be run in the framework of NSDV model. Indeed, in this last context, the density ρ is initialized with a uniform value across the space domain and should remain constant in time.

The NSDV model (5.1) does not admit an exact solution with such test case, but is modelized in a very simple way:

- The space domain is rectangular and set by default as the unit square $\Omega = [0,1]^2$,
- Full-Dirichlet boundary conditions are considered for the velocity with

$$\mathbf{u}_D(t,x,y) = \begin{pmatrix} u_{D,x}(x,y) \\ 0 \end{pmatrix}, \qquad u_{D,x}(x,y) = \begin{cases} 1\,, & \text{if } y = y_{\max}, \\ 0\,, & \text{else,} \end{cases} \tag{5.20}$$

- The initial state is set as $(\rho^0, \mathbf{u}^0, p^0) = (0,0,0)$,
- There is no source terms: $\mathbf{f} = 0$.

5.2 NS: Incompressible 2D Navier-Stokes equations

The Incompressible 2D Navier-Stokes model, or NS model, that is solved by NS2DDV is constituted by the following dimensionless equations:

$$\begin{cases} \partial_t \mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u} + \nabla p = \dfrac{1}{Re}\Delta\mathbf{u} + \mathbf{f}\,, & \text{(5.21a)} \\ \nabla\cdot\mathbf{u} = 0\,. & \text{(5.21b)} \end{cases}$$

In these equations, the unknowns are the velocity $\mathbf{u} : [0,T]\times\Omega \to \mathbb{R}^2$ and the scalar pressure $p : [0,T]\times\Omega \to \mathbb{R}$, and the characteristic Reynolds number $Re > 0$ and the source term $f : [0,T]\times\Omega \to \mathbb{R}^2$ are given.

5.2.1 Boundary conditions

In the following lines, we consider the following partition of the boundary $\partial\Omega$

$$\partial\Omega = \partial\Omega_D \cup \partial\Omega_N\,, \qquad \partial\Omega_D \cap \partial\Omega_N = \emptyset\,. \tag{5.22}$$

In all test cases, $\partial\Omega_D$ is assumed to be non-empty. In some of them, $\partial\Omega_N$ can be an empty set.

Dirichlet boundary conditions are applied on $\partial\Omega_D$. Such conditions are used to model an inlet velocity applied on a part of the boundary and usually write as

$$\mathbf{u} = \mathbf{u}_D\,, \qquad \text{on } [0,T]\times\partial\Omega_D. \tag{5.23}$$

Concerning the boundary conditions on the free boundary $\partial\Omega_N$, several solutions are currently available in NS2DDV code for the model NS: assuming that $\partial\Omega_N \neq \emptyset$ and considering $\Gamma \subset \partial\Omega_N$ (we can have $\Gamma = \partial\Omega_N$ or not), one can consider

- Pseudo-traction (`NEUMANN`) boundary conditions:

$$\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\mathbf{n}_\Omega = \mathbf{g}\,, \qquad \text{on } [0,T]\times\Gamma, \tag{5.24}$$

  where $\mathbf{g} : [0,T]\times\Gamma \to \mathbb{R}^2$ is given and such that $\mathbf{g}(t,\cdot) \in \left(H^{1/2}(\Gamma)\right)^2$ for any $t \in [0,T]$.

- Absorbing (`NATURAL`) boundary conditions: considering $\beta \geq 0$, $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$ and $\Theta(a) = a - \beta\, a^+$, the boundary conditions are

$$\begin{aligned}\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\,\mathbf{n}_\Omega - \frac{1}{2}\Theta\left(\mathbf{u}\cdot\mathbf{n}_\Omega + (1-2\alpha_1)\,\mathbf{w}\cdot\mathbf{n}_\Omega\right)(\mathbf{u}-\mathbf{w})\\ - \Theta\left((1-\alpha_2)\,\mathbf{u}\cdot\mathbf{n}_\Omega + (\alpha_2-\alpha_3)\,\mathbf{w}\cdot\mathbf{n}_\Omega\right)\mathbf{w} = \frac{1}{Re}(\nabla\mathbf{w})\,\mathbf{n}_\Omega - r\,\mathbf{n}_\Omega + \mathbf{g}\,,\end{aligned} \tag{5.25}$$

  on $[0,T]\times\Gamma$. In the equation above, $\mathbf{g} : [0,T]\times\Gamma \to \mathbb{R}^2$ is given and such that $\mathbf{g}(t,\cdot) \in \left(H^{1/2}(\Gamma)\right)^2$ for any t, and $(\mathbf{w}, r)$ is the solution of

$$\begin{cases} \dfrac{1}{Re}\Delta\mathbf{w}(t,\cdot) - \nabla r(t,\cdot) = 0\,, & \text{in } \Omega, \quad (5.26a)\\ \nabla\cdot\mathbf{w}(t,\cdot) = 0\,, & \text{in } \Omega, \quad (5.26b)\\ \mathbf{w}(t,\cdot) = \overline{\mathbf{w}}(t,\cdot)\,, & \text{on } \partial\Omega, \quad (5.26c)\end{cases}$$

  and where $\overline{\mathbf{w}}(t,\cdot)$ is an extension of $\mathbf{u}_D(t,\cdot)$ to $\partial\Omega$ such that

$$\overline{\mathbf{w}}(t,\cdot) \in \left(H^{1/2}(\partial\Omega)\right)^2\,, \qquad \int_{\partial\Omega}\overline{\mathbf{w}}(t,\cdot)\cdot\mathbf{n}_\Omega\, d\sigma = 0\,,, \tag{5.27}$$

  for any t fixed in $[0,T]$. Hence, if $\mathbf{u}_D$ does not depend on the time variable, the auxiliary problem (5.26) is solved only once at the beginning of the simulation. More details about these boundary conditions for the NS model are available in [1, 2].
  If such boundary conditions are considered, it is necessary to fill the following parameters in the setup file:

```
PARAMETERS.FE.ALPHA1 = α1;
PARAMETERS.FE.ALPHA2 = α2;
PARAMETERS.FE.ALPHA3 = α3;
PARAMETERS.FE.BETA = β;
```

- Stress-balance (`STRESS_BALANCE`) boundary conditions:

$$\frac{1}{Re}(\nabla \mathbf{u})\,\mathbf{n}_\Omega - p\mathbf{n}_\Omega - \frac{|\mathbf{u}|^2}{2} S_0(\mathbf{u}\cdot\mathbf{n}_\Omega)\,\mathbf{n}_\Omega = \mathbf{g}\,, \qquad \text{on } [0,T]\times\Gamma, \tag{5.28}$$

where $\mathbf{g} : [0,T]\times\Gamma \to \mathbb{R}^2$ is given and such that $\mathbf{g}(t,\cdot) \in \left(H^{1/2}(\Gamma)\right)^2$ for any $t \in [0,T]$, and S_0 is defined as

$$S_0(\mathbf{u}\cdot\mathbf{n}_\Omega) = \frac{1}{2}\left(1 - \tanh\left(\frac{\mathbf{u}\cdot\mathbf{n}_\Omega}{U_0\,\delta}\right)\right)\,, \tag{5.29}$$

where $U_0 > 0$ is the characteristic velocity scale (generally linked to the chosen test case), $\delta > 0$ is a fixed dimensionless parameter set such that $\delta \ll 1$. This last parameter can be set in the setup file as follows:

```
PARAMETERS.FE.BC_DELTA = δ;
```

More details about these boundary conditions for the NS model are available in [5].

Remark 5.4. If $\partial\Omega_N = \emptyset$, the inlet velocity $\mathbf{u}_D$ must satisfy the compatibility condition

$$\int_{\partial\Omega} \mathbf{u}_D(t,\cdot)\cdot\mathbf{n}_\Omega\, d\sigma = 0\,, \qquad \forall\, t \in [0,T]\,, \tag{5.30}$$

and the system (5.21)-(5.23) must be supplemented with an additional constrain on the pressure to ensure the existence and the uniqueness of its solution. 2 solutions can be used in NS2DDV:

- Null average pressure: $\displaystyle\int_\Omega p(t,\mathbf{x})\,d\mathbf{x} = 0$ is imposed for any $t \in [0,T]$,
- Null pressure on a specific point: provided with $\mathbf{x}_* \in \Omega$, $p(t,\mathbf{x}_*) = 0$ is imposed for any $t \in [0,T]$.

This choice can be done in a setup file as follows:

```
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_AVERAGE';
```

or

```
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_FIXED_POINT';
```

In addition, it is necessary to fix a value for the point $\mathbf{x}_* = (x_*, y_*)$. This can be done by modifying the following parameters:

```
PARAMETERS.FE.NFX_X = x_*;
PARAMETERS.FE.NFX_Y = y_*;
```

Remark 5.5. If $\partial\Omega_N \neq \emptyset$ and if pseudo-traction conditions (5.24) are considered, the system (5.21)-(5.23)-(5.24) does not require any additional constrain on $\mathbf{u}_D$ or p like in Remark 5.4 to be well-posed.

Remark 5.6. If $\partial\Omega_N \neq \emptyset$ and if absorbing conditions (5.25)-(5.26) are considered, the system (5.21)-(5.23)-(5.25)-(5.26) does not require any additional constrain on $\mathbf{u}_D$ or p like in Remark 5.4 to be well-posed. However, the auxiliary Stokes problem (5.26) requires one of the following constrains to be well-posed:

- $\int_\Omega r(t, \mathbf{x})\, d\mathbf{x} = 0$ is imposed for any $t \in [0, T]$,
- Provided with $\mathbf{x}_* \in \Omega$, $r(t, \mathbf{x}_*) = 0$ is imposed for any $t \in [0, T]$.

To choose one of these constrain and set the coordinates of $\mathbf{x}_*$, the user shall proceed just as in Remark 5.4.

5.2.2 Initial state

In addition of boundary conditions, the model (5.21) is completed with an initial velocity $\mathbf{u}^0 : \Omega \to \mathbb{R}^2$ and an initial pressure $p^0 : \Omega \to \mathbb{R}$. NS2DDV embeds two ways to choose the definition of the initial state $(\mathbf{u}^0, p^0)$:

- The initial state is analytically defined according to the chosen test case: in such case, the user can refer to the next paragraphs for the complete definition of $(\mathbf{u}^0, p^0)$. At the beginning of the simulation, NS2DDV discretizes $(\mathbf{u}^0, p^0)$ according to the chosen finite element scheme.
  To set this initialization choice, the user must modify the setup file as follows:

```
% Use a third-party file to initialize the simulation
% Allowed formats: '', 'xxx.h5', 'xxx.mat'
% If '' (default value) is selected, the simulation will be
% initialized with the test case parameters
% If a valid file is selected, some domain and physical parameters
% above will be ignored.
PARAMETERS.INIT_FILE = '';
```

- The initial state is already discretized and is obtained from an external `.h5` / `.mat` file: in such case, the analytical definition of $(\mathbf{u}^0, p^0)$ associated to the chosen test

case is not taken into account, and is replaced by some data that are already discretized that will be denoted here $(\mathbf{u}_h^0, p_h^0)$.
At the beginning of the simulation, NS2DDV reads the external file that is provided by the user (see below), extracts the fields corresponding to the velocity and the pressure, and uses them as initial data.
To set this initialization choice, the user must modify the setup file as follows (`the_external_file.h5` is replaced by the file the user wants to use):

```
% Use a third-party file to initialize the simulation
% Allowed formats: '', 'xxx.h5', 'xxx.mat'
% If '' (default value) is selected, the simulation will be
% initialized with the test case parameters
% If a valid file is selected, some domain and physical parameters
% above will be ignored.
PARAMETERS.INIT_FILE = 'the_external_file.h5';
```

We remind that such external file is associated to a mesh file, some domain geometry parameters, and a time value that will be denoted here t^*. By default, NS2DDV will use these data for the initialization step, meaning that the initial time t^0 within the simulation will be set to t^*, and that the domain and mesh parameters within the setup file such as `PARAMETERS.DOMAIN.XMAX` or `PARAMETERS.MESH.NBSEG_X` will be ignored. However, it is possible to take into account these setup parameters as follows:

- Replace the default mesh: in such case, a new mesh based on the parameters within the setup file is generated and the discrete $(\mathbf{u}_h^0, p_h^0)$ is interpolated on this new mesh to get $(\mathbf{u}_{\tilde{h}}^0, p_{\tilde{h}}^0)$ (Warning: this work can be time consuming). To activate this feature, the user must modify the following lines in the setup file:

  ```
  % Build a new mesh at initialization
  % (ignored if PARAMETERS.INIT_FILE = '')
  % Allowed values: 'YES' or 'NO' (default)
  PARAMETERS.REMESH = 'NO';
  ```

- Modify the domain geometry: in the case where the domain is remeshed, it is possible to to get rid of the domain geometry that is inherited from the "old" mesh, and to replace it by the domain geometry that is described in the setup file. To activate this feature, the user must activate the remeshing option as above and modify the following lines in the setup file:

```
% Use domain parameters from the third-party initialization
% file or PARAMETERS.DOMAIN.XXX
% (ignored if PARAMETERS.INIT_FILE = '' or
% PARAMETERS.REMESH = 'NO')
% 'EXT_FILE'      The domain geometry parameters are imposed by
%                 the contents of PARAMETERS.INIT_FILE
% 'CURRENT_FILE'  Use the variables PARAMETERS.DOMAIN.XXX above
PARAMETERS.WHICH_GEOMETRY = 'EXT_FILE';
```

Remark 5.7. If this feature is activated, the domain geometry kind (`RECTANGLE`, `DIHEDRON`, ...) that is chosen in the setup file must be the same as in the external file used for initializing NS2DDV.

– Force the initial time value: if a external initial data is used, it is possible to impose the initial time value t^0 to t^* instead of 0 (default value). To do this, the user must modify the following lines in the setup file:

```
% Reset time variable if a third-party file is used to
% initialize the simulation (ignored if
%  PARAMETERS.INIT_FILE = '')
% Allowed values: 'YES' (default) or 'NO'
PARAMETERS.RESET_TIME = 'YES';
```

5.2.3 EXAC: analytical test case with full Dirichlet boundary conditions

This test case is dedicated to the validation of any resolution of NS model with full-Dirichlet boundary conditions. It is associated to an analytical solution and *ad hoc* manufactured source terms and a default Reynolds number $\mathrm{Re} = 1$.

This test case can be run on a rectangular space domain $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ (see paragraph 4.1.1) with default definition $\Omega = [0,1]^2$. In the case `EXAC`, an analytical solution is set as $(\mathbf{u}_{ex}, p_{ex})$ defined as

$$\begin{aligned} \mathbf{u}_{ex}(t,x,y) &= \begin{pmatrix} -y\cos t \\ x\cos t \end{pmatrix}, \\ p_{ex}(t,x,y) &= \sin t \sin\left(\frac{2\pi\,(x-x_*)}{x_{\max}-x_{\min}}\right) \sin\left(\frac{2\pi\,(y-y_*)}{y_{\max}-y_{\min}}\right), \end{aligned} \tag{5.31}$$

with a fixed point $\mathbf{x}_* = (x_*, y_*) \in \Omega$ (default values $x_* = \frac{x_{\min}+x_{\max}}{2}$, $y_* = \frac{y_{\min}+y_{\max}}{2}$). The boundary conditions are set as

$$\mathbf{u}_D = \mathbf{u}_{ex|_{\partial\Omega}}, \qquad \text{on } [0,T]\times\partial\Omega\ , \tag{5.32}$$

and the source term $\mathbf{f}$ is manufactured to fit with the exact solution.

5.2.4 EXACNEU: analytical test case with Dirichlet-pseudotraction boundary conditions

This test case is dedicated to the validation of any resolution of NS model with mixed Dirichlet-Neumann boundary conditions. It is associated to an analytical solution and *ad hoc* manufactured source terms and a default Reynolds number $\mathrm{Re} = 1$.

This test case can be run on a rectangular space domain $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ (see paragraph 4.1.1) with default definition $\Omega = [0,1]^2$. In the case `EXACNEU`, the analytical solution is set as $(\mathbf{u}_{ex}, p_{ex})$ defined as

$$\begin{aligned} \mathbf{u}_{ex}(t,x,y) &= \begin{pmatrix} \sin\left(\dfrac{x-x_{\min}}{x_{\max}-x_{\min}}\right) \sin\left(\dfrac{y-y_{\min}}{y_{\max}-y_{\min}}+t\right) \\ \cos\left(\dfrac{x-x_{\min}}{x_{\max}-x_{\min}}\right) \cos\left(\dfrac{y-y_{\min}}{y_{\max}-y_{\min}}+t\right) \end{pmatrix}, \\ p_{ex}(t,x,y) &= \dfrac{1}{Re(x_{\max}-x_{\min})} \cos\left(\dfrac{x-x_{\min}}{x_{\max}-x_{\min}}\right) \sin\left(\dfrac{y-y_{\min}}{y_{\max}-y_{\min}}+t\right). \end{aligned} \tag{5.33}$$

The boundary conditions are set as

$$\begin{aligned} &\frac{1}{Re}(\nabla \mathbf{u})\,\mathbf{n}_\Omega - p\,\mathbf{n}_\Omega = 0\,, && \text{on } [0,T]\times\partial\Omega_N, \\ &\mathbf{u}_D = \mathbf{u}_{ex|_{\partial\Omega_D}}\,, && \text{on } [0,T]\times\partial\Omega_D, \end{aligned} \tag{5.34}$$

with $\partial\Omega_N = \{x_{\min}\}\times]y_{\min}, y_{\max}[$ and $\partial\Omega_D = \partial\Omega \backslash \partial\Omega_N$.

Concerning the source term $\mathbf{f}$, it is analytically manufactured to fit with the exact solution $(\mathbf{u}_{ex}, p_{ex})$.

5.2.5 POIS: 2D Poiseuille flow

This test case is dedicated to the evolution of a 2D Poiseuille flow between two plates. It can be run on a rectangular space domain $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ (see paragraph 4.1.1) or a backward-facing step space domain (see Fig. 1 and paragraph 4.1.2). In both cases, we have the following common assumptions:

- The source term $\mathbf{f}$ is set to 0,
- The domain boundary is splitted in two main parts $\partial\Omega_N$ and $\partial\Omega_D$ with

$$\partial\Omega_N = \{x_{\max}\}\times]y_{\min}, y_{\max}[\,, \qquad \partial\Omega_D = \partial\Omega\backslash\partial\Omega_N\,, \tag{5.35}$$

  and the Dirichlet boundary conditions writes as (5.23) where the inlet velocity $\mathbf{u}_D : [0,T]\times\partial\Omega_D \to \mathbb{R}^2$ is defined in the following lines,
- The characteristic inlet velocity $\overline{U}$ is set by default to 1.

According to the chosen domain geometry, various definitions of $\mathbf{u}_D$ are proposed:

- Rectangular space domain: The considered space domain is $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ with default value $\Omega = [0, 0.04] \times [-0.005, 0.005]$ (see paragraph 4.1.1). 3 ways are proposed to define $\mathbf{u}_D(t,x,y) = \begin{pmatrix} u_{D,x}(x,y) \\ 0 \end{pmatrix}$

  – Constant inlet velocity:

$$u_{D,x}(x,y) = \begin{cases} \overline{U}\,, & \text{if } x = x_{\min},\ y \in]y_{\min}, y_{\max}[, \\ 0\,, & \text{else.} \end{cases} \tag{5.36}$$

  In such case, there is no exact solution and the initial state is defined by

$$\mathbf{u}^0 = 0\,, \qquad p^0 = 0\,. \tag{5.37}$$

  This choice can be done in a setup file by setting

```
PARAMETERS.PHYSICAL.INLET_VELOCITY = U;
PARAMETERS.PHYSICAL.INLET_VELOCITY_KIND = 'CONSTANT';
```

  – Polynomial inlet velocity (default):

$$u_{D,x}(x,y) = \begin{cases} -\frac{6\overline{U}\,(y^2 - y_{\min} - (y_{\max}+y_{\min})(y-y_{\min}))}{(y_{\max}-y_{\min})^2}\,, & \text{if } x = x_{\min},\ y \in]y_{\min}, y_{\max}[, \\ 0\,, & \text{else.} \end{cases} \tag{5.38}$$

  In such case, there is an exact solution defined as

$$\begin{aligned} \mathbf{u}_{ex}(t,x,y) &= \begin{pmatrix} -\frac{6\overline{U}\,(y^2 - y_{\min} - (y_{\max}+y_{\min})(y-y_{\min}))}{(y_{\max}-y_{\min})^2} \\ 0 \end{pmatrix}, \\ p_{ex}(t,x,y) &= -\frac{12\overline{U}}{Re\,(y_{\max}-y_{\min})^2}(x - x_{\min})\,, \end{aligned} \tag{5.39}$$

  and the initial state is set as

$$\mathbf{u}^0(\mathbf{x}) = \mathbf{u}_{ex}(0,\mathbf{x})\,, \qquad p^0(\mathbf{x}) = p_{ex}(0,\mathbf{x})\,. \tag{5.40}$$

  This choice can be done in a setup file by setting

```
PARAMETERS.PHYSICAL.INLET_VELOCITY = U;
PARAMETERS.PHYSICAL.INLET_VELOCITY_KIND = 'POLYNOMIAL';
```

  – Piecewise polynomial inlet velocity (see also [10]): fixing $\overline{y} \in]y_{\min}, y_{\max}[$, the inlet velocity is defined as

$$u_{D,x}(x,y) = \begin{cases} -\frac{6\overline{U}\,(y^2 - \overline{y} - (y_{\max}+\overline{y})(y-\overline{y}))}{(y_{\max}-\overline{y})^2}\,, & \text{if } x = x_{\min},\ y \in]\overline{y}, y_{\max}[, \\ 0\,, & \text{else.} \end{cases} \tag{5.41}$$

In such case, there is no exact solution and the initial state is defined by

$$\mathbf{u}^0 = 0\,, \qquad p^0 = 0\,. \tag{5.42}$$

This choice can be done in a setup file by setting

```
PARAMETERS.PHYSICAL.INLET_VELOCITY = U;
PARAMETERS.PHYSICAL.INLET_VELOCITY_KIND = 'MIXED';
PARAMETERS.PHYSICAL.INLET_VELOCITY_Y = y;
```

By default, $\overline{y}$ is set to $\frac{y_{\min}+y_{\max}}{2}$.

- Backward-facing step space domain: a step as it is represented in Fig. 1 is chosen (see paragraph 4.1.2), with default domain parameters $x_{\min} = 0$, $x_{\max} = 27$, $y_{\min} = 0$, $y_{\max} = 3$, $x_{\text{step}} = 12$, $y_{\text{step}} = 1$.

  2 ways are proposed to define $\mathbf{u}_D(t,x,y) = \begin{pmatrix} u_{D,x}(x,y) \\ 0 \end{pmatrix}$

  – Constant inlet velocity:

$$u_{D,x}(x,y) = \begin{cases} \overline{U}\,, & \text{if } x = x_{\min},\ y \in]y_{\text{step}}, y_{\max}[, \\ 0\,, & \text{else.} \end{cases} \tag{5.43}$$

  In such case, there is no exact solution and the initial state is defined by

$$\mathbf{u}^0 = 0\,, \qquad p^0 = 0\,. \tag{5.44}$$

  – Polynomial inlet velocity:

$$u_{D,x}(x,y) = \begin{cases} -\frac{6\overline{U}\,(y^2 - y_{\text{step}} - (y_{\max}+y_{\text{step}})(y-y_{\text{step}}))}{(y_{\max}-y_{\text{step}})^2}\,, & \text{if } x = x_{\min},\ y \in]y_{\text{step}}, y_{\max}[, \\ 0\,, & \text{else.} \end{cases} \tag{5.45}$$

  In such case, there is no exact solution and the initial state is defined by

$$\mathbf{u}^0 = 0\,, \qquad p^0 = 0\,. \tag{5.46}$$

In the `POIS` test case, 3 kinds of boundary conditions can be applied on $\partial\Omega_N$. It can be done by modifying the following parameters in the setup file:

```
PARAMETERS.FE.BC_RIGHT_UX = BC;
PARAMETERS.FE.BC_RIGHT_UY = BC;
```

where BC stands for one of the following boundary conditions for $\mathbf{u}$:

- Homogeneous pseudo-traction conditions (BC = `'NEUMANN'`) (default):

$$\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\mathbf{n}_\Omega = 0\,, \qquad \text{on } [0,T]\times\partial\Omega_N, \tag{5.47}$$

- Homogeneous absorbing conditions ($BC =$ `'NATURAL'`):

$$\begin{aligned}\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\,\mathbf{n}_\Omega - \frac{1}{2}\Theta\left(\mathbf{u}\cdot\mathbf{n}_\Omega + (1-2\alpha_1)\,\mathbf{w}\cdot\mathbf{n}_\Omega\right)(\mathbf{u}-\mathbf{w}) \\ - \Theta\left((1-\alpha_2)\,\mathbf{u}\cdot\mathbf{n}_\Omega + (\alpha_2-\alpha_3)\,\mathbf{w}\cdot\mathbf{n}_\Omega\right)\mathbf{w} = \frac{1}{Re}(\nabla\mathbf{w})\,\mathbf{n}_\Omega - r\,\mathbf{n}_\Omega\,,\end{aligned} \tag{5.48}$$

where $(\mathbf{w}, r)$ does not depend on t and is the solution of

$$\begin{cases} \dfrac{1}{Re}\Delta\mathbf{w} - \nabla r = 0\,, & \text{in } \Omega, \quad (5.49a) \\ \nabla\cdot\mathbf{w} = 0\,, & \text{in } \Omega, \quad (5.49b) \\ \mathbf{w} = \overline{\mathbf{w}}\,, & \text{on } \partial\Omega, \quad (5.49c) \end{cases}$$

and $\overline{\mathbf{w}}$ is defined as

$$\overline{\mathbf{w}}_{|\partial\Omega_D} = \mathbf{u}_D\,, \qquad \overline{\mathbf{w}}_{|\partial\Omega_N} = \begin{pmatrix} w_{N,x} \\ 0 \end{pmatrix}, \tag{5.50}$$

with

$$w_{N,x}(x,y) = -\frac{6\overline{W}\,(y^2 - y_{\min} - (y_{\max}+y_{\min})(y-y_{\min}))}{(y_{\max}-y_{\min})^2}\,, \tag{5.51}$$

for any $(x,y)\in\partial\Omega_N$, and with $\overline{W}$ defined as

$$\overline{W} = \begin{cases} \overline{U}\,, & \text{if } \Omega \text{ is rectangular and } \mathbf{u}_D \text{ is polynomial,} \\ \overline{U}\,, & \text{if } \Omega \text{ is rectangular and } \mathbf{u}_D \text{ is constant,} \\ \frac{\overline{U}\,(y_{\max}-\overline{y})}{y_{\max}-y_{\min}}\,, & \text{if } \Omega \text{ is rectangular and } \mathbf{u}_D \text{ is piecewise polynomial,} \\ \frac{\overline{U}\,(y_{\max}-y_{\text{step}})}{y_{\max}-y_{\min}}\,, & \text{if } \Omega \text{ is backward-facing step-shaped.} \end{cases} \tag{5.52}$$

Finally, the function Θ is defined by $\Theta(a) = a - \beta a^+$ with default $\beta = 1$ and the parameters $\alpha_1, \alpha_2, \alpha_3$ are set by default to 0.5, 1 and 1 respectively.

- Homogeneous stress-balance boundary conditions ($BC =$ `'STRESS_BALANCE'`):

$$\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\mathbf{n}_\Omega - \frac{|\mathbf{u}|^2}{2}S_0(\mathbf{u}\cdot\mathbf{n}_\Omega)\,\mathbf{n}_\Omega = 0\,, \qquad \text{on } [0,T]\times\partial\Omega_N, \tag{5.53}$$

with

$$S_0(\mathbf{u}\cdot\mathbf{n}_\Omega) = \frac{1}{2}\left(1 - \tanh\left(\frac{\mathbf{u}\cdot\mathbf{n}_\Omega}{\overline{U}\,\delta}\right)\right), \tag{5.54}$$

where δ is set to 0.05 by default.

5.2.6 CAEN: lid-driven cavity

This test case is dedicated to the simulation of the 2D lid-driven cavity problem. This problem does not admit an exact solution but is modelized in a very simple way:

- The space domain is rectangular and set by default as the unit square $\Omega = [0,1]^2$,
- Full-Dirichlet boundary conditions are considered for the velocity, *i.e.* $\partial\Omega = \partial\Omega_D$, with

$$\mathbf{u}_D(t,x,y) = \begin{pmatrix} u_{D,x}(x,y) \\ 0 \end{pmatrix}, \qquad u_{D,x}(x,y) = \begin{cases} 1\,, & \text{if } y = y_{\max}, \\ 0\,, & \text{else,} \end{cases} \tag{5.55}$$

- The initial state is set as $(\mathbf{u}^0, p^0) = (0,0)$,
- There is no source terms: $\mathbf{f} = 0$.

5.2.7 NONA: non-analytical test case

This test case is dedicated to the simulation of turbulences arising near an airplane wing. At present time, 3 different domain geometries can be used to run this test case (see paragraph 4.1):

- Rectangular space domain $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$: if such geometry is chosen, the default definition of Ω is $[-1,5] \times [0,2]$.
- Backward-facing step-shaped domain (see Figure 1): if such geometry is chosen, the default domain parameters are the following:

$$x_{\min} = -1\,, \quad x_{\max} = 5\,, \quad y_{\min} = 0\,, \quad y_{\max} = 2\,, \quad x_{\text{step}} = 0\,, \quad y_{\text{step}} = 1\,.$$

- Dihedron-shaped domain (see Figure 2): if such geometry is chosen, the default domain parameters are the following:

$$x_{\min} = -1\,, \quad x_{\max} = 5\,, \quad y_{\min} = 0\,, \quad y_{\max} = 2\,, \quad x_{A_1} = 0\,, \quad y_{A_1} = 1\,, \quad x_{A_2} = 3\,.$$

According to the chosen geometry, we define $y_{m,l}$ as follows:

$$y_{m,l} = \begin{cases} y_{\min}\,, & \text{if } \Omega \text{ is a rectangular domain,} \\ y_{\text{step}}\,, & \text{if } \Omega \text{ is a backward-facing step domain,} \\ y_{A_1}\,, & \text{if } \Omega \text{ is a dihedron domain.} \end{cases} \tag{5.56}$$

In any case, the initial state is characterized by

$$\mathbf{u}^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad p^0 = 0\,, \tag{5.57}$$

and there is no source term in the Navier-Stokes equation, *i.e.* $\mathbf{f} = 0$.

For all geometries, we consider the following free boundary subpart:

$$\partial\Omega_N = \{x_{\max}\}\times]y_{\min}, y_{\max}[\,, \tag{5.58}$$

and $\partial\Omega_D = \partial\Omega - \partial\Omega_N$. In the case of a dihedron-shaped space domain, we consider the following partition of $\partial\Omega_D$:

$$\partial\Omega_D = \partial\Omega_{D,B} \cup \partial\Omega_{D,NS} \cup \partial\Omega_{D,FSV}\,, \tag{5.59}$$

where

- $\partial\Omega_{D,B} = \{x_{\min}\}\times]y_{m,l}, y_{\max}[$ is the inlet velocity boundary,
- $\partial\Omega_{D,FSV} = [x_{\min}, x_{\max}] \times \{y_{\max}\}$ is a Dirichlet-like extension of the inlet velocity boundary,
- $\partial\Omega_{D,NS} = \partial\Omega_D \setminus (\partial\Omega_{D,B} \cup \partial\Omega_{D,FSV})$ is the no-slip boundary.

To precise the definition of $\mathbf{u}_D$, we define the Blasius function (see [11]) as the solution $f : [0, +\infty[\to \mathbb{R}$ of

$$\left\{\begin{array}{ll} 2f''' + ff'' = 0\,, & \text{on }]0, +\infty[, \\ f'(0) = f(0) = 0\,, & \\ \lim_{\eta\to+\infty} f'(\eta) = 1\,, & \end{array}\right. \tag{5.60}$$

we fix $x_s < x_{\min}$, and we introduce $\mathbf{u}_B$ as

$$\mathbf{u}_B(\mathbf{x}) = \left(\begin{array}{c} u_{B,x}(\mathbf{x}) \\ u_{B,y}(\mathbf{x}) \end{array}\right), \tag{5.61}$$

$$\begin{aligned} u_{B,x}(x,y) &= (\partial_\eta f)\,(\eta(x,y))\,, \\ u_{B,y}(x,y) &= -\frac{1}{2}\frac{1}{\sqrt{(x-x_s)\,Re}}\big[f\,(\eta(x,y)) - \eta(x,y)(\partial_\eta f)\,(\eta(x,y))\,\big] \end{aligned} \tag{5.62}$$

where $\eta : \Omega \to \mathbb{R}$ is defined as

$$\eta(x,y) = (y - y_{m,l})\sqrt{\frac{Re}{x - x_s}}. \tag{5.63}$$

Then the Dirichlet boundary conditions are the following:

$$\mathbf{u}_D(x,y) = \left\{\begin{array}{ll} \mathbf{u}_B(x,y)\,, & \text{if } (x,y) \in \partial\Omega_{D,B}, \\ \mathbf{u}_B(x,y_{\max})\,, & \text{if } (x,y) \in \partial\Omega_{D,FSV}, \\ 0\,, & \text{if } (x,y) \in \partial\Omega_{D,NS}. \end{array}\right. \tag{5.64}$$

For managing the value of x_s, the user can modify the following parameter in the setup file:

```
PARAMETERS.PHYSICAL.XSTART = x_s;
```

In the `NONA` test case, 3 kinds of boundary conditions can be applied on $\partial\Omega_N$. It can be done by modifying the following parameters in the setup file:

```
PARAMETERS.FE.BC_RIGHT_UX = BC;
PARAMETERS.FE.BC_RIGHT_UY = BC;
```

where BC stands for one of the following boundary conditions for $\mathbf{u}$:

- Homogeneous pseudo-traction conditions ($BC =$ `'NEUMANN'`) (default):

$$\frac{1}{Re}(\nabla \mathbf{u})\,\mathbf{n}_\Omega - p\mathbf{n}_\Omega = 0\,, \qquad \text{on } [0,T]\times\partial\Omega_N, \tag{5.65}$$

- Homogeneous absorbing conditions ($BC =$ `'NATURAL'`):

$$\begin{aligned}\frac{1}{Re}(\nabla \mathbf{u})\,\mathbf{n}_\Omega - p\,\mathbf{n}_\Omega - \frac{1}{2}\Theta\left(\mathbf{u}\cdot\mathbf{n}_\Omega + (1-2\alpha_1)\,\mathbf{w}\cdot\mathbf{n}_\Omega\right)(\mathbf{u}-\mathbf{w})\\ -\Theta\left((1-\alpha_2)\,\mathbf{u}\cdot\mathbf{n}_\Omega + (\alpha_2-\alpha_3)\,\mathbf{w}\cdot\mathbf{n}_\Omega\right)\mathbf{w} = \frac{1}{Re}(\nabla\mathbf{w})\,\mathbf{n}_\Omega - r\,\mathbf{n}_\Omega\,,\end{aligned} \tag{5.66}$$

on $[0,T]\times\partial\Omega_N$, where $(\mathbf{w}, r)$ does not depend on t and is the solution of

$$\begin{cases} \dfrac{1}{Re}\Delta\mathbf{w} - \nabla r = 0\,, & \text{in } \Omega, \quad (5.67a)\\ \nabla\cdot\mathbf{w} = 0\,, & \text{in } \Omega, \quad (5.67b)\\ \mathbf{w} = \overline{\mathbf{w}}\,, & \text{on } \partial\Omega, \quad (5.67c)\end{cases}$$

and $\overline{\mathbf{w}}$ is defined as

$$\overline{\mathbf{w}}_{|\partial\Omega_D} = \mathbf{u}_D\,, \qquad \overline{\mathbf{w}}_{|\partial\Omega_N} = \mathbf{w}_N\,, \tag{5.68}$$

where $\mathbf{w}_N$ can be defined in two different ways:

  - Blasius completion: define $\mathbf{w}_N$ on $\partial\Omega_N$ as

$$\mathbf{w}_N(x,y) = \begin{pmatrix} (\partial_{\tilde\eta} f)\left(\tilde\eta(x,y)\right) \\ -\frac{1}{2}\frac{1}{\sqrt{(x-\tilde{x_s})\,Re}}\left[f\left(\tilde\eta(x,y)\right) - \tilde\eta(x,y)(\partial_{\tilde\eta} f)\left(\tilde\eta(x,y)\right)\right]\end{pmatrix}, \tag{5.69}$$

    with $\tilde\eta$ defined as

$$\tilde\eta(x,y) = (y-y_{\min})\sqrt{\frac{Re}{x-\tilde{x_s}}}, \tag{5.70}$$

and $\tilde{x_s}$ has to be identified as the unique solution in $\mathbb{R}$ of

$$\sqrt{\frac{x_{\max}-\tilde{x_s}}{x_{\max}-x_s}}\, f\left((y_{\max}-y_{\min})\sqrt{\frac{Re}{x_{\max}-\tilde{x_s}}}\right) = f\left((y_{\max}-y_{m,l})\sqrt{\frac{Re}{x_{\max}-x_s}}\right), \tag{5.71}$$

This completion can be chosen by setting

```
PARAMETERS.FE.BC_EXTENSIONREF = 'BLASIUS';
```

in the setup file.

– Couette completion: define $\mathbf{w}_N$ on $\partial\Omega_N$ as

$$\mathbf{w}_N(x,y) = \begin{pmatrix} \overline{W}\,\frac{y-y_{\min}}{y_{\max}-y_{\min}} \\ 0 \end{pmatrix}, \tag{5.72}$$

with

$$\overline{W} = \frac{2}{y_{\max}-y_{\min}}\sqrt{\frac{\overline{U_x}\,(x_{\max}-x_s)}{Re}}\, f\left(\eta(x_{\max},y_{\max})\right). \tag{5.73}$$

This completion can be chosen by setting

```
PARAMETERS.FE.BC_EXTENSIONREF = 'COUETTE';
```

in the setup file.
Finally, the function Θ is defined by $\Theta(a) = a - \beta a^+$ with default $\beta = 1$ and the parameters $\alpha_1, \alpha_2, \alpha_3$ are set by default to 0.5, 1 and 1 respectively.

- Homogeneous stress-balance boundary conditions (BC = `'STRESS_BALANCE'`):

$$\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\mathbf{n}_\Omega - \frac{|\mathbf{u}|^2}{2}S_0(\mathbf{u}\cdot\mathbf{n}_\Omega)\,\mathbf{n}_\Omega = 0\,, \qquad \text{on } [0,T]\times\partial\Omega_N, \tag{5.74}$$

with

$$S_0(\mathbf{u}\cdot\mathbf{n}_\Omega) = \frac{1}{2}\left(1-\tanh\left(\frac{\mathbf{u}\cdot\mathbf{n}_\Omega}{\delta}\right)\right), \tag{5.75}$$

where δ is set to 0.05 by default.

5.2.8 GTPSI: translation of a vortex

This test case is dedicated to the validation of free boundary condition implementation like Absorbing boundary conditions or Stress-Balancing conditions (see paragraph 5.2.1).

The purpose of this test case is to put a vortex in a rectangular domain, apply an inlet velocity on the left part of the boundary and to observe the drift of the vortex until it goes out of the domain.

The test case `GTPSI` does not involve any analytical solution and is described by the following assumptions:

- There is no external forces, so $\mathbf{f} = 0$,
- The space domain is rectangular and of the form $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ (default domain $\Omega = [0,1]^2$),
- The initial state is defined as follows:

$$\begin{aligned} \mathbf{u}^0(x,y) &= \begin{cases} \mathbf{u}_{BG}(x,y) + \mathbf{u}_v(x,y)\,, & \text{if } |x-x_v|^2 + |y-y_v|^2 < R_{v,trunc}^2, \\ \mathbf{u}_{BG}(x,y)\,, & \text{else,} \end{cases} \\ p^0(x,y) &= 0\,, \end{aligned} \tag{5.76}$$

  with the following definitions:

  - $\mathbf{u}_v$ writes as

$$\mathbf{u}_v(x,y) = \frac{2a_v}{r_v^2} \exp\left(-\frac{|x-x_v|^2 + |y-y_v|^2}{r_v^2} \right) \begin{pmatrix} -y+y_v \\ x-x_v \end{pmatrix}, \tag{5.77}$$

  - $\mathbf{x}_v = (x_v, y_v)$ is the vortex center and $R_{v,trunc} > 0$ is the truncation radius such that the closed disc $D(\mathbf{x}_v, R_{v,trunc})$ is strictly embedded in Ω.
    These parameters take the default values $\mathbf{x}_v = (\frac{x_{\min}+x_{\max}}{2}, \frac{y_{\min}+y_{\max}}{2})$ and $R_v = 0.2$ and can be modified in a setup file as follows:

```
% Coordinates of the vortex center
PARAMETERS.PHYSICAL.VORTEX_XC = x_v;
PARAMETERS.PHYSICAL.VORTEX_YC = y_v;
% Radius of the vortex support
PARAMETERS.PHYSICAL.VORTEX_RADIUS = R_v;
```

  - $a_v > 0$ and $r_v > 0$ are respectively the vortex amplitude and core size. Their respective default values are $a_v = 1$ and $r_v = 0.2$ and the user can modify them in a setup file as follows:

```
% Vortex core size
PARAMETERS.PHYSICAL.VORTEX_CORESIZE = r_v;
% Vortex intensity
PARAMETERS.PHYSICAL.VORTEX_INTENSITY = a_v;
```

  - $\mathbf{u}_{BG}$ is the background velocity and can be defined in two different ways:

* Constant background velocity (default):

$$\mathbf{u}_{BG}(x,y) = \begin{pmatrix} \overline{U} \\ 0 \end{pmatrix}, \tag{5.78}$$

with the inlet averaged velocity $\overline{U}$ is set to 1 by default. This configuration can be chosen by setting

```
% Averaged inlet velocity
PARAMETERS.PHYSICAL.INLET_VELOCITY = U;
% Type of inlet velocity
% (allowed values: 'CONSTANT' or 'POLYNOMIAL')
PARAMETERS.PHYSICAL.INLET_VELOCITY_KIND = 'CONSTANT';
```

* Polynomial background velocity:

$$\mathbf{u}_{BG}(x,y) = \begin{pmatrix} -\frac{6\overline{U}\,(y^2 - y_{\min}^2 - (y_{\min}+y_{\max})(y-y_{\min}))}{(y_{\max}-y_{\min})^2} \\ 0 \end{pmatrix}, \tag{5.79}$$

with the inlet averaged velocity $\overline{U}$ is set to 1 by default. This configuration can be chosen by setting

```
% Averaged inlet velocity
PARAMETERS.PHYSICAL.INLET_VELOCITY = U;
% Type of inlet velocity
% (allowed values: 'CONSTANT' or 'POLYNOMIAL')
PARAMETERS.PHYSICAL.INLET_VELOCITY_KIND = 'POLYNOMIAL';
```

- The bound domain $\partial\Omega$ is splitted into two parts $\partial\Omega_D$ and $\partial\Omega_N$ with

$$\partial\Omega_N = \{x_{\max}\}\times]y_{\min}, y_{\max}[\,, \qquad \partial\Omega_D = \partial\Omega \backslash \partial\Omega_N\,, \tag{5.80}$$

- The trace of velocity on the Dirichlet boundary is set as

$$\mathbf{u}_D(x,y) = \mathbf{u}_{BG}(x,y)\,, \qquad \forall\,(x,y) \in \partial\Omega_D\,. \tag{5.81}$$

- 3 kinds of boundary conditions can be applied on $\partial\Omega_N$. It can be done by modifying the following parameters in the setup file:

```
PARAMETERS.FE.BC_RIGHT_UX = BC;
PARAMETERS.FE.BC_RIGHT_UY = BC;
```

where BC stands for one of the following boundary conditions for $\mathbf{u}$:

– Homogeneous pseudo-traction conditions (BC = `'NEUMANN'`) (default):

$$\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\mathbf{n}_\Omega = 0\,, \qquad \text{on } [0,T]\times\partial\Omega_N, \tag{5.82}$$

– Homogeneous absorbing conditions ($BC =$ `'NATURAL'`):

$$\begin{aligned}&\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\,\mathbf{n}_\Omega - \frac{1}{2}\Theta\left(\mathbf{u}\cdot\mathbf{n}_\Omega + (1-2\alpha_1)\,\mathbf{w}\cdot\mathbf{n}_\Omega\right)(\mathbf{u}-\mathbf{w})\\&-\Theta\left((1-\alpha_2)\,\mathbf{u}\cdot\mathbf{n}_\Omega + (\alpha_2-\alpha_3)\,\mathbf{w}\cdot\mathbf{n}_\Omega\right)\mathbf{w} = \frac{1}{Re}(\nabla\mathbf{w})\,\mathbf{n}_\Omega - r\,\mathbf{n}_\Omega\,,\end{aligned} \tag{5.83}$$

on $[0,T]\times\partial\Omega_N$, where $(\mathbf{w}, r)$ does not depend on t and is the solution of

$$\begin{cases}\dfrac{1}{Re}\Delta\mathbf{w} - \nabla r = 0\,, & \text{in } \Omega, \quad (5.84a)\\ \nabla\cdot\mathbf{w} = 0\,, & \text{in } \Omega, \quad (5.84b)\\ \mathbf{w} = \mathbf{u}_{BG}\,, & \text{on } \partial\Omega, \quad (5.84c)\end{cases}$$

and the function Θ is defined by $\Theta(a) = a - \beta a^+$ with default $\beta = 1$ and the parameters $\alpha_1, \alpha_2, \alpha_3$ are set by default to 0.5, 1 and 1 respectively.

– Homogeneous stress-balance boundary conditions ($BC =$ `'STRESS_BALANCE'`):

$$\frac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\mathbf{n}_\Omega - \frac{|\mathbf{u}|^2}{2}S_0(\mathbf{u}\cdot\mathbf{n}_\Omega)\,\mathbf{n}_\Omega = 0\,, \qquad \text{on } [0,T]\times\partial\Omega_N, \tag{5.85}$$

with

$$S_0(\mathbf{u}\cdot\mathbf{n}_\Omega) = \frac{1}{2}\left(1 - \tanh\left(\frac{\mathbf{u}\cdot\mathbf{n}_\Omega}{\overline{U}\,\delta}\right)\right)\,, \tag{5.86}$$

where δ is set to 0.05 by default.

6 Time semi-discretizations

We assume that a time grid $(t^0, \dots, t^N)$ is provided, and satisfying

$$0 = t^0 < t^1 < \cdots < t^N = T\,, \tag{6.1}$$

with $\Delta t^n = t^{n+1} - t^n$ defined for any $n = 0, \dots, N-1$. In addition, we assume that N is as small as possible and that all Δt^n are smaller than a given Δt that can be chosen by the user. Indeed, for a single run (no convergence or stability analysis), this bound is defined as

$$\Delta t = C_m h_{\min}^{\alpha_m} + C_M h_{\max}^{\alpha_M}\,, \tag{6.2}$$

where $h_{\min}$ and $h_{\max}$ are respectively the minimal and maximal edge length in the provided space mesh, and $C_m, C_M, \alpha_m, \alpha_M$ can be specified by the user in the setup file by modifying the following lines:

```
% Value of alphamax
PARAMETERS.FE.ALPHAMAX_STEP_TIME = αM;
% Value of alphamin
PARAMETERS.FE.ALPHAMIN_STEP_TIME = αm;
% Value of Cmax
PARAMETERS.FE.CMAX_STEP_TIME = CM;
% Value of Cmin
PARAMETERS.FE.CMIN_STEP_TIME = Cm;
```

Hence, the approximation of $\mathbf{u}$ (respectively p and ρ) associated with the space mesh $\mathfrak{M}$ and the time step bound Δt is denoted with $\mathbf{u}_{\Delta t,\mathfrak{M}}$ (respectively $p_{\Delta t,\mathfrak{M}}$ and $\rho_{\Delta t,\mathfrak{M}}$) and, for each t^n in the time mesh associated to Δt, $\mathbf{u}_{\Delta t,\mathfrak{M}}(t^n,\cdot)$ (respectively $p_{\Delta t,\mathfrak{M}}(t^n,\cdot)$ and $\rho_{\Delta t,\mathfrak{M}}(t^n,\cdot)$) is simply denoted with $\mathbf{u}^n$ (respectively p^n and ρ^n).

6.1 Time splittings (specific to `NSDV` model)

In this paragraph, we describe the time splittings that can be applied to the NSDV model (5.1). In the following lines, we denote with $\rho^k : \Omega \to \mathbb{R}$, $\mathbf{u}^k : \Omega \to \mathbb{R}^2$ and $p^k : \Omega \to \mathbb{R}$ some approximations of $\rho(t^k,\cdot)$, $\mathbf{u}(t^k,\cdot)$ and $p(t^k,\cdot)$ respectively.

At present time, Lax (order 1 in time) and Strang (order 2 in time) splittings are available. We should recommend the use of Strang splitting since only BDF2-type and MUSCL-type time semi-discretizations are implemented (these methods are second order accurate in time).

6.1.1 Lax splitting

This choice can be made by modifying the setup file in the following way:

```
% Time splitting
% 1  1st order Lie splitting
% 2  2nd order symmetric Strang splitting
PARAMETERS.TIME_SPLITTING = 1;
```

We assume that $(\rho^k, \mathbf{u}^k, p^k)$ are known for any $k \leq n$:

1. ρ^{n+1} is computed by the resolution of the following equation on $[t^n, t^{n+1}]$:

$$\begin{cases} \partial_t \rho + \nabla \cdot (\rho\, \mathbf{u}^n) = 0\,, & \text{(6.3a)} \\ \rho(t^n, \cdot) = \rho^n\,, & \text{(6.3b)} \end{cases}$$

   and we denote $\rho^{n+1} := \rho(t^{n+1}, \cdot)$.

2. $(\mathbf{u}^{n+1}, p^{n+1})$ is computed by the resolution of the following linearized equation on $[t^n, t^{n+1}]$:

$$\begin{cases} \rho^{n+1}\left[\partial_t \mathbf{u} + (\mathbf{u}^{\sharp,n+1} \cdot \nabla)\mathbf{u}\right] + \nabla p = \dfrac{1}{Re}\mu\, \Delta \mathbf{u} + \rho^{n+1}\, \mathbf{f}\,, & \text{(6.4a)} \\ \nabla \cdot \mathbf{u} = 0\,, & \text{(6.4b)} \\ \mathbf{u}(t^n, \cdot) = \mathbf{u}^n\,, & \text{(6.4c)} \\ p(t^n, \cdot) = p^n\,, & \text{(6.4d)} \end{cases}$$

   and we denote $\mathbf{u}^{n+1} := \mathbf{u}(t^{n+1}, \cdot)$, $p^{n+1} := p(t^{n+1}, \cdot)$.
   In the equation above, $\mathbf{u}^{\sharp,n+1}$ is computed as an extrapolation of $(\mathbf{u}^k)_{k=0,\dots,n}$ (see Section 6.2 for the details).

6.1.2 Strang splitting

This choice can be made by modifying the setup file in the following way:

```
% Time splitting
% 1  1st order Lie splitting
% 2  2nd order symmetric Strang splitting
PARAMETERS.TIME_SPLITTING = 2;
```

We assume that $(\rho^k, \mathbf{u}^k, p^k)$ are known for any $k \leq n$:

1. ρ^{n+1} is computed by the resolution of the following equation on $[t^n, t^{n+1}]$:

$$\begin{cases} \partial_t \rho + \nabla \cdot (\rho\, \mathbf{u}^n) = 0\,, & \text{(6.5a)} \\ \rho(t^n, \cdot) = \rho^n\,, & \text{(6.5b)} \\ & \text{(6.5c)} \end{cases}$$

   and we denote $\rho^{n+1} := \rho(t^{n+1}, \cdot)$.

2. $(\mathbf{u}^{n+1}, p^{n+1})$ is computed by the resolution of the following linearized equation on $[t^n, t^{n+1}]$:

$$\begin{cases} \rho^{n+1}\left[\partial_t \mathbf{u} + (\mathbf{u}^{\sharp,n+1} \cdot \nabla)\mathbf{u}\right] + \nabla p = \dfrac{1}{Re}\mu\, \Delta \mathbf{u} + \rho^{n+1}\, \mathbf{f}\,, & \text{(6.6a)} \\ \nabla \cdot \mathbf{u} = 0\,, & \text{(6.6b)} \\ \mathbf{u}(t^n, \cdot) = \mathbf{u}^n\,, & \text{(6.6c)} \\ p(t^n, \cdot) = p^n\,, & \text{(6.6d)} \end{cases}$$

and we denote $\mathbf{u}^{n+1} := \mathbf{u}(t^{n+1}, \cdot)$, $p^{n+1} := p(t^{n+1}, \cdot)$.
In the equation above, $\mathbf{u}^{\sharp,n+1}$ is computed as an extrapolation of $(\mathbf{u}^k)_{k=0,\dots,n}$ (see Section 6.2 for the details).

3. $(\mathbf{u}^{n+2}, p^{n+2})$ is computed by the resolution of the following linearized equation on $[t^{n+1}, t^{n+2}]$:

$$\begin{cases} \rho^{n+1}\left[\partial_t \mathbf{u} + (\mathbf{u}^{\sharp,n+2}\cdot\nabla)\mathbf{u}\right] + \nabla p = \dfrac{1}{Re}\mu\,\Delta\mathbf{u} + \rho^{n+1}\,\mathbf{f}\,, & \text{(6.7a)} \\ \nabla\cdot\mathbf{u} = 0\,, & \text{(6.7b)} \\ \mathbf{u}(t^{n+1},\cdot) = \mathbf{u}^{n+1}\,, & \text{(6.7c)} \\ p(t^{n+1},\cdot) = p^{n+1}\,, & \text{(6.7d)} \end{cases}$$

and we denote $\mathbf{u}^{n+2} := \mathbf{u}(t^{n+2}, \cdot)$, $p^{n+2} := p(t^{n+2}, \cdot)$.
In the equation above, $\mathbf{u}^{\sharp,n+2}$ is computed as an extrapolation of $(\mathbf{u}^k)_{k=0,\dots,n+1}$ (see Section 6.2 for the details).

4. ρ^{n+1} is computed by the resolution of the following equation on $[t^{n+1}, t^{n+2}]$:

$$\begin{cases} \partial_t\rho + \nabla\cdot(\rho\,\mathbf{u}^{n+2}) = 0\,, & \text{(6.8a)} \\ \rho(t^{n+1},\cdot) = \rho^{n+1}\,, & \text{(6.8b)} \end{cases}$$

and we denote $\rho^{n+2} := \rho(t^{n+2}, \cdot)$.

6.2 Time semi-discretizations for Navier-Stokes equation

In the following lines, we focus on the time semi-discretization of the following Navier-Stokes equation:

$$\begin{cases} \rho^*\,\left[\partial_t\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right] + \nabla p - \dfrac{1}{Re}\Delta\mathbf{u} = \rho\,\mathbf{f}\,, \\ \nabla\cdot\mathbf{u} = 0\,, \end{cases} \tag{6.9}$$

in Ω. In these equation, ρ^* and $\mathbf{f}$ are given, and we consider the following decomposition of $\partial\Omega$:

$$\partial\Omega = \partial\Omega_D \cup \underbrace{\partial\Omega_{Na} \cup \partial\Omega_{Ne}}_{=\partial\Omega_N}, \tag{6.10}$$

and we couple (6.9) the following boundary conditions:

$$\begin{cases} \mathbf{u} = \mathbf{u}_D\,, & \text{on } [0,T]\times\partial\Omega_D, \\ \mathbf{u}\cdot\mathbf{n}_\Omega = 0\,, & \text{on } [0,T]\times\partial\Omega_{Na}, \\ \left(\dfrac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega\right)\times\mathbf{n}_\Omega = 0\,, & \text{on } [0,T]\times\partial\Omega_{Na}, \\ \dfrac{1}{Re}(\nabla\mathbf{u})\,\mathbf{n}_\Omega - p\,\mathbf{n}_\Omega = \mathbf{g}_{Ne}\,, & \text{on } [0,T]\times\partial\Omega_{Ne}, \end{cases} \tag{6.11}$$

6.2.1 BDF2 "direct" method

The more convenient way to discretize in time the system (6.9) is to consider the second order Implicit Backward Differential Formula. Assuming that, for some $n \in \{1, \dots, N-1\}$, we already know ρ^{n+1}, $\mathbf{u}^n$ and $\mathbf{u}^{n-1}$, the couple $(\mathbf{u}^{n+1}, p^{n+1})$ is computed by solving the following system:

$$\begin{cases} \rho^{n+1}\left[\alpha^n\,\mathbf{u}^{n+1} + (\mathbf{u}^{\sharp,n+1}\cdot\nabla)\,\mathbf{u}^{n+1}\right] + \nabla p^{n+1} - \dfrac{1}{Re}\Delta\mathbf{u}^{n+1} \\ \qquad\qquad = \rho^{n+1}\,(\mathbf{f}^{n+1} - \beta^n\,\mathbf{u}^n - \gamma^n\,\mathbf{u}^{n-1})\,, & \text{in } \Omega, \\ \nabla\cdot\mathbf{u}^{n+1} = 0\,, & \text{in } \Omega, \\ \mathbf{u}^{n+1} = \mathbf{u}_D(t^{n+1},\cdot)\,, & \text{on } \partial\Omega_D, \\ \mathbf{u}^{n+1}\cdot\mathbf{n}_\Omega = 0\,, & \text{on } \partial\Omega_{Na}, \\ \left(\dfrac{1}{Re}(\nabla\mathbf{u}^{n+1})\,\mathbf{n}_\Omega\right)\times\mathbf{n}_\Omega = 0\,, & \text{on } \partial\Omega_{Na}, \\ \dfrac{1}{Re}(\nabla\mathbf{u}^{n+1})\,\mathbf{n}_\Omega - p^{n+1}\,\mathbf{n}_\Omega = \mathbf{g}_{Ne}(t^{n+1},\cdot)\,, & \text{on } \partial\Omega_{Ne}, \end{cases} \tag{6.12}$$

where $\alpha^n, \beta^n, \gamma^n$ are defined as

$$\alpha^n = \frac{\Delta t^{n-1} + 2\Delta t^n}{\Delta t^n\,(\Delta t^{n-1} + \Delta t^n)}, \quad \beta^n = -\frac{\Delta t^{n-1} + \Delta t^n}{\Delta t^{n-1}\,\Delta t^n}, \quad \gamma^n = \frac{\Delta t^n}{\Delta t^{n-1}\,(\Delta t^{n-1} + \Delta t^n)}. \tag{6.13}$$

Finally, $\mathbf{u}^{\sharp,n+1}$ is an approximation of $\mathbf{u}(t^{n+1},\cdot)$ obtained with an extrapolation of $\mathbf{u}^n$ and $\mathbf{u}^{n-1}$. More precisely, it is defined as

$$\mathbf{u}^{\sharp,n+1} = \left(1 + \frac{\Delta t^n}{\Delta t^{n-1}}\right)\mathbf{u}^n - \frac{\Delta t^n}{\Delta t^{n-1}}\mathbf{u}^{n-1}\,. \tag{6.14}$$

To choose this time semi-discretization for Navier-Stokes equation, the user must specify the following parameters in the setup file:

```
% Time semi-discretization of the Stokes equation
PARAMETERS.FE.SCHEME = 'BDF2';
```

6.2.2 BDF2 projection methods

A projection method derived from the BDF2 method (6.12)-(6.13)-(6.14) is also proposed in NS2DDV code. This method is based on the use of an additional function sequence $(\phi^n)_{n=0,\dots,N}$ and the successive resolution of the following systems:

1. Knowing $(\rho^{n-1}, \mathbf{u}^{n-1}, p^{n-1}, \phi^{n-1})$, $(\rho^n, \mathbf{u}^n, p^n, \phi^n)$ and ρ^{n+1}, we first compute $\mathbf{u}^{\sharp,n+1}$ and $p^{\sharp,n+1}$ where $p^{\sharp,n+1}$ writes as

$$p^{\sharp,n+1} = p^{n-1} - \frac{\beta^n}{\alpha^{n-1}}\phi^n - \frac{\gamma^n}{\alpha^{n-2}}\phi^{n-1}\,, \tag{6.15}$$

then we solve

$$\begin{cases} \rho^{n+1}\,\left[\alpha^n\,\mathbf{u}^{n+1} + (\mathbf{u}^{\sharp,n+1}\cdot\nabla)\,\mathbf{u}^{n+1}\right] - \dfrac{1}{Re}\Delta\mathbf{u}^{n+1} \\ \qquad = \rho^{n+1}\,(\mathbf{f}^{n+1} - \beta^n\,\mathbf{u}^n - \gamma^n\,\mathbf{u}^{n-1}) - \nabla p^{\sharp,n+1}\,, & \text{in } \Omega, \\ \mathbf{u}^{n+1} = \mathbf{u}_D(t^{n+1},\cdot)\,, & \text{on } \partial\Omega_D, \\ \\ \mathbf{u}^{n+1}\cdot\mathbf{n}_\Omega = 0\,, & \text{on } \partial\Omega_{Na}, \\ \left(\dfrac{1}{Re}(\nabla\mathbf{u}^{n+1})\,\mathbf{n}_\Omega\right)\times\mathbf{n}_\Omega = 0\,, & \text{on } \partial\Omega_{Na}, \\ \\ \dfrac{1}{Re}(\nabla\mathbf{u}^{n+1})\,\mathbf{n}_\Omega = \mathbf{g}_{Ne}(t^{n+1},\cdot) + p^{\sharp,n+1}\,\mathbf{n}_\Omega\,, & \text{on } \partial\Omega_{Ne}, \end{cases} \tag{6.16}$$

2. We compute ϕ^{n+1} by solving an elliptic problem. According to the chosen model and the chosen test case, up to 3 ways for doing this job can be proposed (see [6, 7, 8, 9] for futher details):

   - `PARAMETERS.FE.SCHEME = 'BDF2_PROJ'` (works for all models and all test cases):

$$\begin{cases} \nabla\cdot\left(\dfrac{1}{\alpha^n\,\overline{\rho}}\nabla\phi^{n+1}\right) = \nabla\cdot\mathbf{u}^{n+1}\,, & \text{in } \Omega, \\ \\ \nabla\phi^{n+1}\cdot\mathbf{n}_\Omega = 0\,, & \text{on } \partial\Omega_D\cup\partial\Omega_{Na}, \\ \\ \phi^{n+1} = 0\,, & \text{on } \partial\Omega_{Ne}, \end{cases} \tag{6.17}$$

     where $\overline{\rho} = \min_\Omega(\rho^0)$,

   - `PARAMETERS.FE.SCHEME = 'BDF2_PROJ_VAR'` (works for all test cases associated to the `NSDV` model):

$$\begin{cases} \nabla\cdot\left(\dfrac{1}{\alpha^n\,\rho^{n+1}}\nabla\phi^{n+1}\right) = \nabla\cdot\mathbf{u}^{n+1}\,, & \text{in } \Omega, \\ \\ \nabla\phi^{n+1}\cdot\mathbf{n}_\Omega = 0\,, & \text{on } \partial\Omega_D\cup\partial\Omega_{Na}, \\ \\ \phi^{n+1} = 0\,, & \text{on } \partial\Omega_{Ne}, \end{cases} \tag{6.18}$$

- `PARAMETERS.FE.SCHEME = 'BDF2_PROJ_VAR_A'` (only works for test cases associated to the `NSDV` model and that involve an analytic solution):

$$\begin{cases} \nabla \cdot \left(\dfrac{1}{\alpha^n \, \rho_{ex}(t^{n+1}, \cdot)} \nabla \phi^{n+1} \right) = \nabla \cdot \mathbf{u}^{n+1} \,, & \text{in } \Omega, \\ \nabla \phi^{n+1} \cdot \mathbf{n}_\Omega = 0 \,, & \text{on } \partial\Omega_D \cup \partial\Omega_{Na}, \\ \phi^{n+1} = 0 \,, & \text{on } \partial\Omega_{Ne}, \end{cases} \tag{6.19}$$

For each approach above, if $\partial\Omega_{Ne} = \emptyset$, the Dirichlet condition on ϕ^{n+1} above is replaced by one of the following constrain for insuring the uniqueness of ϕ^{n+1}:

- Null average of ϕ^{n+1}: $\displaystyle\int_\Omega \phi^{n+1}(\mathbf{x}) \, d\mathbf{x} = 0$ is imposed. This choice can be done in the setup file by choosing

```
% Time semi-discretization of the Stokes equation
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_AVERAGE';
```

- Null value of ϕ^{n+1} on a specific point: provided with $\mathbf{x}_* \in \Omega$, $\phi^{n+1}(\mathbf{x}_*) = 0$ is imposed. This choice can be done in the setup file by choosing

```
% Time semi-discretization of the Stokes equation
PARAMETERS.FE.PRESSURE_CONSTRAIN = 'ZERO_FIXED_POINT';
```

In addition, the user must provide the coordinates of $\mathbf{x}_* = (x_*, y_*)$ by modifying the following parameters in the setup file:

```
% Time semi-discretization of the Stokes equation
PARAMETERS.FE.NFX_X = x*;
PARAMETERS.FE.NFX_Y = y*;
```

3. Independently of the value of `PARAMETERS.FE.SCHEME`, we compute p^{n+1} as follows:

$$p^{n+1} = p^n + \phi^{n+1} - \chi \, \nabla \cdot \mathbf{u}^{n+1} \,, \tag{6.20}$$

where χ is a positive parameter that controls the "rotational" correction of the pressure p^{n+1}. Its default value is set to $\frac{1}{2Re}$. This parameter can be managed in the setup file by modifying the following line:

```
% Amplitude of the rotational term in the pressure correction
% for 'BDF2_PROJ*' methods
PARAMETERS.FE.ROT_CORRECTION = 0.5/PARAMETERS.PHYSICAL.RE;
```

6.3 Time semi-discretizations for mass convervation equation (only for `NSDV` model)

In the following lines, we present the time semi-discretization used for the generic mass conservation law

$$\begin{cases} \partial_t \rho + \nabla \cdot (\rho \, \mathbf{u}) = 0 \,, \\ \rho(t = t^0, \cdot) = \rho^0 \,, \end{cases} \tag{6.21}$$

where $\mathbf{u} : \Omega \to \mathbb{R}$ is assumed to be known. In NS2DDV, we use Heun's method: with ρ^n being known, we compute ρ^{n+1} on Ω thanks to the following procedure:

1. Compute ρ^* as
$$\rho^* := \rho^n - \Delta t^n \, \nabla \cdot (\rho^n \, \mathbf{u}) \,, \tag{6.22}$$
2. Deduce ρ^{n+1} with
$$\rho^{n+1} := \rho^n - \frac{\Delta t^n}{2} \left[\nabla \cdot (\rho^n \, \mathbf{u}) + \nabla \cdot (\rho^* \, \mathbf{u}) \right] . \tag{6.23}$$

7 Spatial discretizations

7.1 Mesh notations

We consider a space domain $\Omega \subset \mathbb{R}^2$ on which a mesh $\mathfrak{M}$ has been generated (see paragraph 4.2 dedicated to the mesh generation methods that are available). What is called *mesh* here is represented by a node set $\mathcal{X}$ and a triangle set $\mathcal{T}$. In the next lines, we describe some additional data that can be used in the context of the Finite Element and/or Finite Volume schemes used by NS2DDV.

For 3 distinct points $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ in Ω (see Figure 4), we denote with

- $[\mathbf{A}, \mathbf{B}]$ as the segment with extremities $\mathbf{A}$ and $\mathbf{B}$,
- $\mathbf{AB}$ as the vector going from $\mathbf{A}$ to $\mathbf{B}$,
- $\|\mathbf{AB}\|$ as the length of the vector $\mathbf{AB}$ (and of the segment $[\mathbf{A}, \mathbf{B}]$),
- $(\mathbf{A}, \mathbf{B})$ as the unique line in $\mathbb{R}^2$ passing through $\mathbf{A}$ and $\mathbf{B}$,
- ABC as the unique triangle with vertices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$.

For each node $\mathbf{A} \in \mathcal{X}$, we add the following definitions (see also Figure 4):

- $nt_{\mathbf{A}}$ as the number of triangles $M \in \mathcal{T}$ such that $\mathbf{A} \in \overline{M}$,
- $\mathcal{N}_{\mathbf{A}} = \{\mathbf{A}_1, \ldots, \mathbf{A}_{nt_{\mathbf{A}}}\}$ as the set of the neighbouring vertices of $\mathbf{A}$ oriented in the counter-clockwise order and M_i as the triangle AA_iA_{i+1},
- $\mathbf{A'_{i,i+1}}$ as the control point of the triangle M_i,
- $\mathbf{A_{i,i+1}}$ as the middle-point of $[\mathbf{A}_i, \mathbf{A}_{i+1}]$,
- $\mathbf{A'_i}$ as the middle-point of the segment $[\mathbf{A}, \mathbf{A_i}]$ for any $i = 1, \ldots, nt_{\mathbf{A}}$,
- $\Gamma_i^+ = [\mathbf{A'_i}, \mathbf{A'_{i,i+1}}]$ and $\Gamma_i^- = [\mathbf{A'_{i-1,i}}, \mathbf{A'_i}]$ for any $i = 1, \ldots, nt_{\mathbf{A}}$, by using the convention

$$\begin{array}{rclcrcl} \mathbf{A_1} & = & \mathbf{A_{nt_A+1}}\,, & & \mathbf{A_0} & = & \mathbf{A_{nt_A}}\,, \\ \mathbf{A'_1} & = & \mathbf{A'_{nt_A+1}}\,, & & \mathbf{A'_0} & = & \mathbf{A'_{nt_A}}\,, \\ \mathbf{A_{nt_A,nt_A+1}} & = & \mathbf{A_{nt_A,1}}\,, & & \mathbf{A_{0,1}} & = & \mathbf{A_{nt_A-1,nt_A}}\,, \\ \mathbf{A'_{nt_A,nt_A+1}} & = & \mathbf{A'_{nt_A,1}}\,, & & \mathbf{A'_{0,1}} & = & \mathbf{A'_{nt_A-1,nt_A}}\,, \end{array}$$

- The dual cell $\mathcal{C}_{\mathbf{A}}$ as the polygon with bound defined as

$$\partial\mathcal{C}_{\mathbf{A}} = \bigcup_{i=1}^{nt_{\mathbf{A}}} \Gamma_i^- \cup \Gamma_i^+\,,$$

and $\mathbf{A} \in \mathcal{C}_{\mathbf{A}}$,

- $\mathbf{n}^{\pm}_{i,\mathbf{A}}$ as the outward normal unit vector to $\mathcal{C}_{\mathbf{A}}$ on the edge $\Gamma^{\pm}_i$,
- $\mathbf{C}^{\pm}_{\mathbf{i}}$ as the middle-point of $\Gamma^{\pm}_i$,
- $\mathbf{G}^{\pm}_{\mathbf{A},\mathbf{i}}$ and $\mathbf{H}^{\pm}_{\mathbf{A},\mathbf{i}}$ as the couple of points that are the intersections of the line $(\mathbf{A}, \mathbf{C}^{\pm}_{\mathbf{i}})$ and $\partial\mathcal{C}_{\mathbf{A}}$ such that $\mathbf{G}^{+}_{\mathbf{A},\mathbf{i}} \in [\mathbf{A_i}, \mathbf{A_{i+1}}]$, $\mathbf{G}^{-}_{\mathbf{A},\mathbf{i}} \in [\mathbf{A_{i-1}}, \mathbf{A_i}]$, $\mathbf{H}^{\pm}_{\mathbf{A},\mathbf{i}} \in [\mathbf{A}_{\mathbf{d}^{\pm}_{\mathbf{i}}}, \mathbf{A}_{\mathbf{d}^{\pm}_{\mathbf{i}}+\mathbf{1}}]$ for some $d^{\pm}_i \in \{1, \dots, nt_{\mathbf{A}}\}$,
- $\mathbf{K}^{\pm}_{\mathbf{A},\mathbf{i}}$ and $\mathbf{L}^{\pm}_{\mathbf{A},\mathbf{i}}$ as the couple of points that are the intersections of the line $(\mathbf{A_i}, \mathbf{C}^{\pm}_{\mathbf{i}})$ and $\partial\mathcal{C}_{\mathbf{A}}$ such that $\mathbf{K}^{+}_{\mathbf{A},\mathbf{i}} \in [\mathbf{A}, \mathbf{A_{i+1}}]$, $\mathbf{K}^{-}_{\mathbf{A},\mathbf{i}} \in [\mathbf{A}, \mathbf{A_{i-1}}]$, $\mathbf{L}^{\pm}_{\mathbf{A},\mathbf{i}} \in [\mathbf{A}_{\mathbf{i}_{\mathbf{u}^{\pm}_{\mathbf{i}}}}, \mathbf{A}_{\mathbf{i}_{\mathbf{u}^{\pm}_{\mathbf{i}}}+\mathbf{1}}]$ for some $u^{\pm}_i \in \{1, \dots, nt_{\mathbf{A}_i}\}$,
- $D^{\pm}_{\mathbf{A},\mathbf{A_i}}$ as the *downstream* triangle $AA_{d^{\pm}_i}A_{d^{\pm}_i+1}$ to edge $[\mathbf{A}, \mathbf{A_i}]$: note that

$$\frac{\mathbf{AC}^{\pm}_{\mathbf{i}}}{\|\mathbf{AC}^{\pm}_{\mathbf{i}}\|} \cdot \frac{\mathbf{AA}_{\mathbf{d}^{\pm}_{\mathbf{i}}} + \mathbf{AA}_{\mathbf{d}^{\pm}_{\mathbf{i}}+\mathbf{1}}}{\|\mathbf{AA}_{\mathbf{d}^{\pm}_{\mathbf{i}}} + \mathbf{AA}_{\mathbf{d}^{\pm}_{\mathbf{i}}+\mathbf{1}}\|} \quad \text{is minimal,} \tag{7.1}$$

- $U^{\pm}_{\mathbf{A},\mathbf{A_i}}$ as the *upstream* triangle $A_iA_{i_{u^{\pm}_i}}A_{i_{u^{\pm}_i}+1}$ to edge $[\mathbf{A}, \mathbf{A_i}]$: note that

$$\frac{\mathbf{A_iC}^{\pm}_{\mathbf{i}}}{\|\mathbf{A_iC}^{\pm}_{\mathbf{i}}\|} \cdot \frac{\mathbf{A_iA}_{\mathbf{i}_{\mathbf{u}^{\pm}_{\mathbf{i}}}} + \mathbf{A_iA}_{\mathbf{i}_{\mathbf{u}^{\pm}_{\mathbf{i}}+\mathbf{1}}}}{\|\mathbf{A_iA}_{\mathbf{i}_{\mathbf{u}^{\pm}_{\mathbf{i}}}} + \mathbf{A_iA}_{\mathbf{i}_{\mathbf{u}^{\pm}_{\mathbf{i}}+\mathbf{1}}}\|} \quad \text{is minimal.} \tag{7.2}$$

Remark 7.1. Note that if, for any triangle $M_i \in \mathcal{N}_{\mathbf{A}}$, its control point is chosen as the barycenter, $\mathbf{A_{i,i+1}}$ coincides with the intersection of $(\mathbf{A_i}, \mathbf{A_{i+1}})$ and $(\mathbf{A}, \mathbf{A'_{i,i+1}})$ and $\mathbf{A'_i}$ coincides with the intersection of $(\mathbf{A}, \mathbf{A_i})$ and $(\mathbf{A_{i+1}}, \mathbf{A'_{i,i+1}})$ (see Figure 4).
Such choice for cell control points produces star-shaped dual cells $\mathcal{C}_{\mathbf{A}}$. It can be set by the user by modifying the following lines in the setup file:

```
% Shape of dual cells (node-centered)
% (a dual cell is centered on a node and its bound is obtained by
% connecting the middle point of edges that start from the node and the
% neighbour cells control points)
% 'STARS'    The cell control points are the barycenters
% 'SQUARES'  The cell control points are the orthocenters (works only
%            with PARAMETERS.MESH.GENERATION = 'NS2DDV')
PARAMETERS.FV.CELLS_DESIGN = 'STARS';
```

Remark 7.2. Assume that the considered domain Ω is a rectangle and is provided with a structured mesh constituted of rectangle triangles with control points set as orthocenters. In such case, for any node $\mathbf{A}$ of the mesh, the control volume $\mathcal{C}_{\mathbf{A}}$ is a square with vertices $\mathbf{A_{B,R}}, \mathbf{A_{T,R}}, \mathbf{A_{T,L}}, \mathbf{A_{B,L}}$, and such that $\mathbf{A_R}, \mathbf{A_L}, \mathbf{A_T}, \mathbf{A_B}$ are the respective middle-points of $[\mathbf{A_{T,R}}, \mathbf{A_{B,R}}]$, $[\mathbf{A_{T,L}}, \mathbf{A_{B,L}}]$, $[\mathbf{A_{T,L}}, \mathbf{A_{T,R}}]$, $[\mathbf{A_{B,L}}, \mathbf{A_{B,R}}]$. These notations are independent of the value of $nt_{\mathbf{A}}$ (see Figure 5).

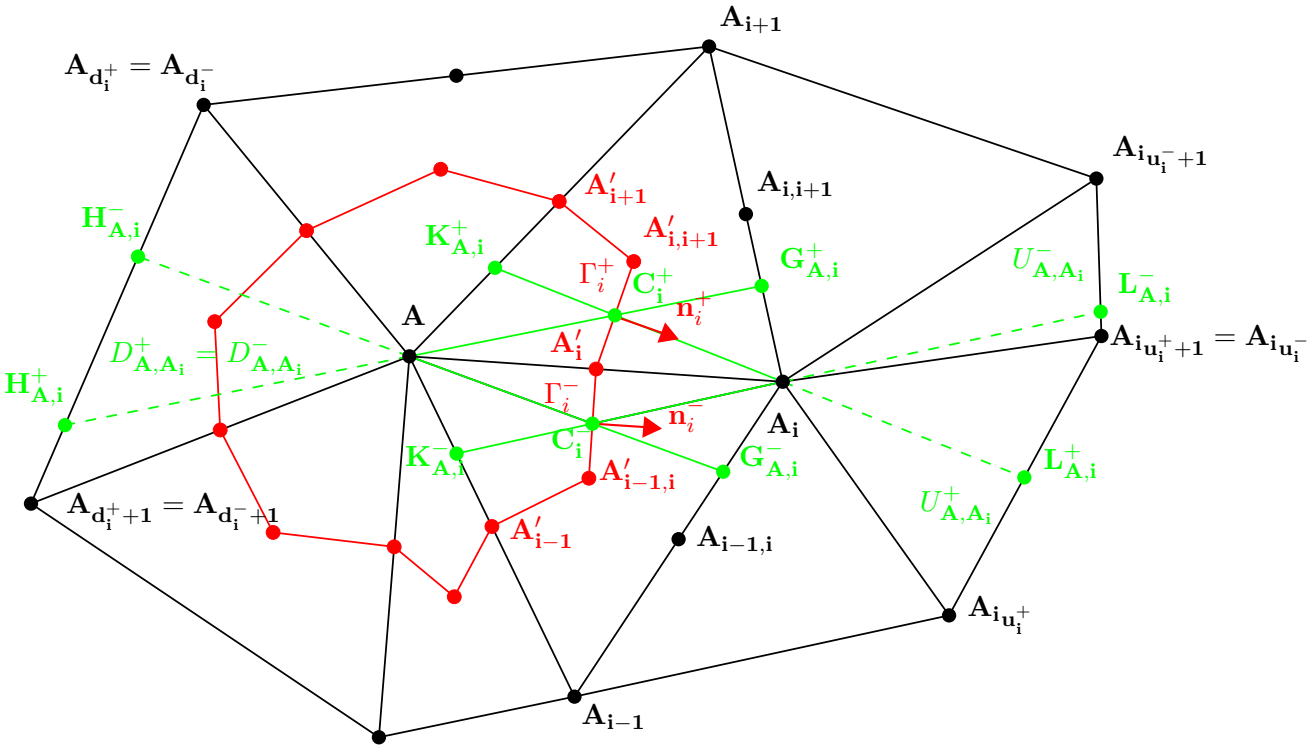

Figure 4: Example of control volume $\mathcal{C}_\mathbf{A}$ with $nt_\mathbf{A} = 6$ and where the control points of triangles are chosen as barycenters.

Such choice can be set by the user by modifying the following lines in the setup file:

```
% Shape of dual cells (node-centered)
% (a dual cell is centered on a node and its bound is obtained by
% connecting the middle point of edges that start from the node and the
% neighbour cells control points)
% 'STARS'    The cell control points are the barycenters
% 'SQUARES'  The cell control points are the orthocenters (works only
%            with PARAMETERS.MESH.GENERATION = 'NS2DDV')
PARAMETERS.FV.CELLS_DESIGN = 'SQUARES';
```

7.2 Finite element methods

Finite element space discretizations are used in NS2DDV for solving the Navier-Stokes equation. More precisely, NS2DDV uses $\mathbb{P}_1$ finite elements for discretizing the pressure p, the auxiliary pressure r (see paragraph 5.2.1) and the ϕ function associated to BDF2 projection methods (see paragraph 6.2.2), and can use either $\mathbb{P}_2$ or $\mathbb{P}_{1,b}$ ($\mathbb{P}_1$-bubble) elements for the velocity $\mathbf{u}$ and the auxiliary velocity $\mathbf{w}$ (see paragraph 5.2.1). It is possible

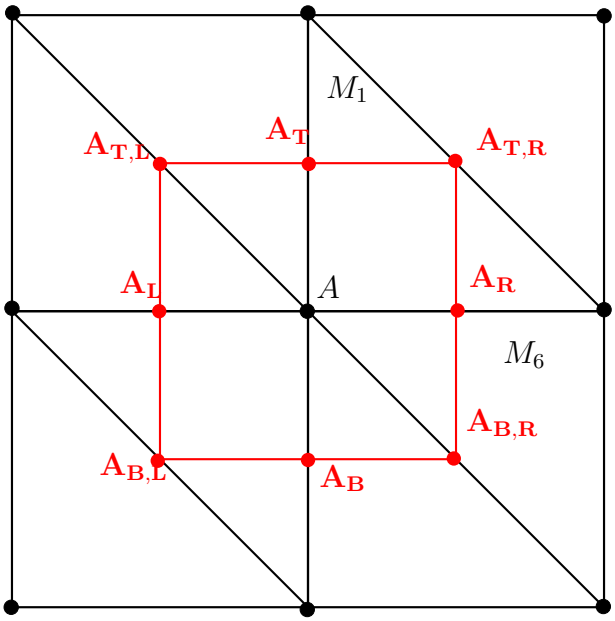

Figure 5: Example of control volume $\mathcal{C}_{\mathbf{A}}$ with $nt_{\mathbf{A}} = 6$ and where the control points of triangles are chosen as orthocenters.

for the user to choose the finite elements for velocity by modifying the setup file as follows:

```
% Space discretization of velocity
% 'P1B' P1-bubble finite elements approximation
% 'P2'  P2 finite elements approximation
PARAMETERS.FE.TYPE = 'P2';
```

We briefly recall what are these finite element discretizations and how they are used in NS2DDV.

We first define a *reference triangle* $\hat{M}$ with vertices $(0,0)$, $(1,0)$, $(0,1)$. Secondly, we define the *reference basis functions* corresponding to a finite element method as a set of polynomial functions $\hat{\mathcal{P}} = (\hat{\varphi}_i)_{i=1,\dots,nd}$ with $\hat{\varphi}_i : \hat{M} \to \mathbb{R}$, and the corresponding set of *reference nodes* $\hat{\Sigma} = (\hat{\mathbf{x}}_\mathbf{i})_{i=1,\dots,nd}$ such that $\hat{\varphi}_i(\hat{\mathbf{x}}_\mathbf{j}) = \delta_{i,j}$ for any $i, j \in [\![1, nd]\!]$. The finite element methods that are considered in NS2DDV are the following:

- $\mathbb{P}_1$ finite elements: $nd = 3$, and the reference basis functions and nodes are defined

as

$$\begin{array}{rclcrcl} \hat{\mathbf{x}}_{\mathbf{1}} & = & (0,0)\,, & & \hat{\varphi}_1(\hat{x},\hat{y}) & = & 1-\hat{x}-\hat{y}\,, \\ \hat{\mathbf{x}}_{\mathbf{2}} & = & (1,0)\,, & & \hat{\varphi}_2(\hat{x},\hat{y}) & = & \hat{x}\,, \\ \hat{\mathbf{x}}_{\mathbf{3}} & = & (1,0)\,, & & \hat{\varphi}_3(\hat{x},\hat{y}) & = & \hat{y}\,. \end{array} \tag{7.3}$$

- $\mathbb{P}_2$ finite elements: $nd = 6$, and the reference basis functions and nodes are defined as

$$\begin{array}{rclcrcl} \hat{\mathbf{x}}_{\mathbf{1}} & = & (0,0)\,, & & \hat{\varphi}_1(\hat{x},\hat{y}) & = & 1-3\hat{x}-3\hat{y}+4\hat{x}\hat{y}+2\hat{x}^2+2\hat{y}^2\,, \\ \hat{\mathbf{x}}_{\mathbf{2}} & = & (1,0)\,, & & \hat{\varphi}_2(\hat{x},\hat{y}) & = & 2\hat{x}^2-\hat{x}\,, \\ \hat{\mathbf{x}}_{\mathbf{3}} & = & (1,0)\,, & & \hat{\varphi}_3(\hat{x},\hat{y}) & = & 2\hat{y}^2-\hat{y}\,, \\ \hat{\mathbf{x}}_{\mathbf{4}} & = & (\frac{1}{2},\frac{1}{2})\,, & & \hat{\varphi}_4(\hat{x},\hat{y}) & = & 2\hat{x}\hat{y}\,, \\ \hat{\mathbf{x}}_{\mathbf{5}} & = & (0,\frac{1}{2})\,, & & \hat{\varphi}_5(\hat{x},\hat{y}) & = & 4\hat{y}-4\hat{x}\hat{y}+4\hat{y}^2\,, \\ \hat{\mathbf{x}}_{\mathbf{6}} & = & (\frac{1}{2},0)\,, & & \hat{\varphi}_6(\hat{x},\hat{y}) & = & 4\hat{x}-4\hat{x}\hat{y}+4\hat{x}^2\,. \end{array} \tag{7.4}$$

- $\mathbb{P}_{1,b}$ finite elements: $nd = 4$, and the reference basis functions and nodes are defined as

$$\begin{array}{rclcrcl} \hat{\mathbf{x}}_{\mathbf{1}} & = & (0,0)\,, & & \hat{\varphi}_1(\hat{x},\hat{y}) & = & (1-\hat{x}-\hat{y})\,(1-9\hat{x}\hat{y})\,, \\ \hat{\mathbf{x}}_{\mathbf{2}} & = & (1,0)\,, & & \hat{\varphi}_2(\hat{x},\hat{y}) & = & \hat{x}\,(1-9(1-\hat{x}-\hat{y})\,\hat{y})\,, \\ \hat{\mathbf{x}}_{\mathbf{3}} & = & (1,0)\,, & & \hat{\varphi}_3(\hat{x},\hat{y}) & = & \hat{y}\,(1-9\,(1-\hat{x}-\hat{y})\,\hat{x})\,, \\ \hat{\mathbf{x}}_{\mathbf{4}} & = & (\frac{1}{3},\frac{1}{3})\,, & & \hat{\varphi}_4(\hat{x},\hat{y}) & = & 27\,(1-\hat{x}-\hat{y})\,\hat{x}\hat{y}\,. \end{array} \tag{7.5}$$

For each triangle $M = ABC \in \mathcal{T}$, we define the affine bijective map $\phi_M : \hat{M} \to M$ as

$$\phi_M(\hat{\mathbf{x}}) = \begin{pmatrix} x_{\mathbf{B}} - x_{\mathbf{A}} & x_{\mathbf{C}} - x_{\mathbf{A}} \\ y_{\mathbf{B}} - y_{\mathbf{A}} & y_{\mathbf{C}} - y_{\mathbf{A}} \end{pmatrix} \hat{\mathbf{x}} + \begin{pmatrix} x_{\mathbf{A}} \\ y_{\mathbf{A}} \end{pmatrix}. \tag{7.6}$$

Then we define $\mathcal{P}_M = (\varphi_{M,i})_{i=1,\dots,nd}$ and $\Sigma_M = (\mathbf{x_{M,i}})_{i=1,\dots,nd}$ with

$$\varphi_{M,i} = \hat{\varphi}_i \circ \phi_M^{-1}\,, \qquad \mathbf{x_{M,i}} = \phi_M(\hat{\mathbf{x}}_{\mathbf{i}})\,, \tag{7.7}$$

for any $i = 1, \dots, nd$ and for any $M \in \mathcal{T}$.

We finally define the node set Σ as

$$\Sigma = (\mathbf{x_k})_{k=1,\dots,\#\Sigma} := \bigcup_{M \in \mathcal{T}} \Sigma_M\,, \tag{7.8}$$

and the basis function set $\mathcal{P} = (\varphi_k)_{k=1,\dots,\#\Sigma}$ with

$$\varphi_{k|_M} := \begin{cases} \varphi_{M,i}\,, & \text{if } \exists\, i \in [\![1, nd]\!] \text{ such that } \mathbf{x_k} = \mathbf{x_{M,i}}, \\ 0\,, & \text{else,} \end{cases} \tag{7.9}$$

for any $k = 1, \dots, \#\Sigma$ and for any $M \in \mathcal{T}$.

In order to apply a finite element spatial discretization to one of the time semi-discrete systems (6.12), (6.16)-(6.17), (6.16)-(6.18) or (6.16)-(6.19), we proceed as follows. In NS2DDV, the density ρ, the pressure p, the auxiliary pressure r and the ϕ function associated to BDF2 projection methods are discretized in space with $\mathbb{P}_1$ finite elements, meaning that we write

$$\rho^n = \sum_{\varphi_k \in \Sigma_\rho} \rho_k^n \, \varphi_k \,, \tag{7.10}$$

and

$$\begin{aligned} r^n &= \sum_{\varphi_k \in \Sigma_p} r_k^n \, \varphi_k \,, \\ \phi^n &= \sum_{\varphi_k \in \Sigma_p} \phi_k^n \, \varphi_k \,, \\ p^n &= \sum_{\varphi_k \in \Sigma_p} p_k^n \, \varphi_k \,, \end{aligned} \tag{7.11}$$

for any time step n, where $\Sigma_p = \Sigma_\rho$ is the function basis set associated to $\mathbb{P}_1$ elements. In the same spirit, the velocity $\mathbf{u}$ and the auxiliary velocity $\mathbf{w}$ are discretized with $\mathbb{P}_2$ or $\mathbb{P}_{1,b}$ finite elements, so we write

$$\mathbf{u}^n = \begin{pmatrix} \displaystyle\sum_{\psi_k \in \Sigma_\mathbf{u}} u_{x,k}^n \, \psi_k \\ \displaystyle\sum_{\psi_k \in \Sigma_\mathbf{u}} u_{y,k}^n \, \psi_k \end{pmatrix}, \qquad \mathbf{w}^n = \begin{pmatrix} \displaystyle\sum_{\psi_k \in \Sigma_\mathbf{u}} w_{x,k}^n \, \psi_k \\ \displaystyle\sum_{\psi_k \in \Sigma_\mathbf{u}} w_{y,k}^n \, \psi_k \end{pmatrix}, \tag{7.12}$$

for any time step n, where $\Sigma_\mathbf{u}$ is the function basis set associated to $\mathbb{P}_2$ or $\mathbb{P}_{1,b}$ elements, according to the value of `PARAMETERS.FE.TYPE` that is provided by the user in the setup file.

A fully-discrete version of (6.12) is obtained as follows: for any node $\mathbf{x_k} \in \Sigma_\mathbf{u}$ such that $\mathbf{x_k} \in \partial\Omega_D$, we set

$$\mathbf{u_k}^{n+1} = \mathbf{u}_D(t^{n+1}, \mathbf{x_k}) \,, \tag{7.13}$$

and for any other node $\mathbf{x_k} \in \Sigma_\mathbf{u}$, we consider the following linear combinations involving $(\rho_k^{n+1})_{k=1,\ldots,\#\Sigma_\rho}$, $(u_{x,k}^m)_{k=1,\ldots,\#\Sigma_\mathbf{u}}$ $(m = n-1, n, n+1)$, $(u_{y,k}^m)_{k=1,\ldots,\#\Sigma_\mathbf{u}}$ $(m = n-1, n, n+1)$, $(p_k^{*,n+1})_{k=1,\ldots,\#\Sigma_p}$:

$$\begin{aligned} &\int_\Omega \left[\rho^{n+1} \left[\alpha^n u_x^n + (\mathbf{u}^{\sharp,n+1} \cdot \nabla) u_x^{n+1} \right] \psi_{\mathbf{u},k} + \frac{1}{Re} \nabla u_x^{n+1} \cdot \nabla \psi_{\mathbf{u},k} - p^{n+1} \, \partial_x \psi_{\mathbf{u},k} \right] d\mathbf{x} \\ &\quad = \int_{\partial\Omega_{Ne}} g_{Ne,x}(t^{n+1}, \cdot) \, \psi_{\mathbf{u},k} \, d\sigma + \int_\Omega \rho^{n+1} \left[f_x^{n+1} - \beta^n \, u_x^n - \gamma^n \, u_x^{n-1} \right] \psi_{\mathbf{u},k} \, d\mathbf{x} \,, \end{aligned} \tag{7.14}$$

$$\begin{aligned} &\int_\Omega \left[\rho^{n+1} \left[\alpha^n u_y^n + (\mathbf{u}^{\sharp,n+1} \cdot \nabla) u_y^{n+1} \right] \psi_{\mathbf{u},k} + \frac{1}{Re} \nabla u_y^{n+1} \cdot \nabla \psi_{\mathbf{u},k} - p^{n+1} \, \partial_y \psi_{\mathbf{u},k} \right] d\mathbf{x} \\ &\quad = \int_{\partial\Omega_{Ne}} g_{Ne,y}(t^{n+1}, \cdot) \, \psi_{\mathbf{u},k} \, d\sigma + \int_\Omega \rho^{n+1} \left[f_y^{n+1} - \beta^n \, u_y^n - \gamma^n \, u_y^{n-1} \right] \psi_{\mathbf{u},k} \, d\mathbf{x} \,, \end{aligned} \tag{7.15}$$

$$\int_\Omega \left(\nabla \cdot \mathbf{u}^{n+1}\right)\, \psi_{p,l}\, d\mathbf{x} = 0\,, \tag{7.16}$$

for any basis function $\psi_{\mathbf{u},k} \in \mathcal{P}_\mathbf{u}$ such that $\mathbf{x_k} \notin \partial\Omega_D$ and for any basis function $\psi_{p,l} \in \mathcal{P}_p$. Coupled with the discrete Dirichlet boundary conditions, we obtain a linear system in which the unknowns are $(u_{x,k}^{n+1})_{k=1,\ldots,\#\Sigma_\mathbf{u}}$, $(u_{y,k}^{n+1})_{k=1,\ldots,\#\Sigma_\mathbf{u}}$, $(p_l^{n+1})_{l=1,\ldots,\#\Sigma_p}$.

A fully-discrete version of (6.16) is obtained by replacing equations (7.14)-(7.15)-(7.16) by

$$\begin{aligned}
&\int_\Omega \left[\rho^{n+1}\left[\alpha^n u_x^n + (\mathbf{u}^{\sharp,n+1}\cdot\nabla)u_x^{n+1}\right]\psi_{\mathbf{u},k} + \frac{1}{Re}\nabla u_x^{n+1}\cdot\nabla\psi_{\mathbf{u},k}\right] d\mathbf{x}\\
&= \int_{\partial\Omega_{Ne}} g_{Ne,x}(t^{n+1},\cdot)\,\psi_{\mathbf{u},k}\, d\sigma\\
&\quad + \int_\Omega \left[\rho^{n+1}\,\left[f_x^{n+1} - \beta^n\, u_x^n - \gamma^n\, u_x^{n-1}\right]\,\psi_{\mathbf{u},k} + p^{\sharp,n+1}\,\partial_x\psi_{\mathbf{u},k}\right] d\mathbf{x}\,,
\end{aligned} \tag{7.17}$$

$$\begin{aligned}
&\int_\Omega \left[\rho^{n+1}\left[\alpha^n u_y^n + (\mathbf{u}^{\sharp,n+1}\cdot\nabla)u_y^{n+1}\right]\psi_{\mathbf{u},k} + \frac{1}{Re}\nabla u_y^{n+1}\cdot\nabla\psi_{\mathbf{u},k}\right] d\mathbf{x}\\
&= \int_{\partial\Omega_{Ne}} g_{Ne,y}(t^{n+1},\cdot)\,\psi_{\mathbf{u},k}\, d\sigma\\
&\quad + \int_\Omega \left[\rho^{n+1}\,\left[f_y^{n+1} - \beta^n\, u_y^n - \gamma^n\, u_y^{n-1}\right]\,\psi_{\mathbf{u},k} + p^{\sharp,n+1}\,\partial_y\psi_{\mathbf{u},k}\right] d\mathbf{x}\,,
\end{aligned} \tag{7.18}$$

for any basis function $\psi_{\mathbf{u},k} \in \mathcal{P}_\mathbf{u}$ such that $\mathbf{x_k} \notin \partial\Omega_D$. Coupled with the discrete Dirichlet boundary conditions, we obtain a linear system in which the unknowns are $(u_{x,k}^{n+1})_{k=1,\ldots,\#\Sigma_\mathbf{u}}$ and $(u_{y,k}^{n+1})_{k=1,\ldots,\#\Sigma_\mathbf{u}}$.

To obtain the pressure $(p_k^{n+1})_{k=1,\ldots,\#\Sigma_p}$, we compute $(\phi_k^{n+1})_{k=1,\ldots,\#\Sigma_p}$ by solving the linear system constituted of

$$-\int_\Omega \frac{1}{\alpha^n\,\hat{\rho}}\nabla\phi^{n+1}\cdot\nabla\psi_{p,l}\, d\mathbf{x} = \int_\Omega (\nabla\cdot\mathbf{u}^{n+1})\,\psi_{p,l}\, d\mathbf{x} \tag{7.19}$$

for any $\psi_{p,l} \in \mathcal{P}_p$ such that $\mathbf{x_l} \notin \partial\Omega_{Ne}$ and of the Dirichlet conditions

$$\phi_l^{n+1} = 0\,, \qquad \forall\, \mathbf{x_l} \in \Sigma_p \text{ such that } \mathbf{x_l} \in \partial\Omega_{Ne}. \tag{7.20}$$

In (7.19), $\hat{\rho}$ is replaced either by $\bar{\rho}$, ρ^{n+1} or $\rho_{ex}(t^{n+1},\cdot)$ according to the value of the setup parameter `PARAMETERS.FE.SCHEME` (see paragraph 6.2.2).

$(p_k^{n+1})_{k=1,\ldots,\#\Sigma_p}$ is finally obtained by inverting the linear system constituted of

$$\int_\Omega p^{n+1}\,\psi_{p,l}\, d\mathbf{x} = \int_\Omega (p^n + \phi^{n+1} - \chi\,\nabla\cdot\mathbf{u}^{n+1})\,\psi_{p,l}\, d\mathbf{x}\,, \tag{7.21}$$

for all $\psi_{p,l} \in \mathcal{P}_p$.

7.3 Finite volume methods

We describe here the discretization in space of the following mass conservation law:

$$\partial_t \rho + \nabla \cdot (\rho\, \mathbf{u}) = 0\,, \tag{7.22}$$

where $\mathbf{u} : \Omega \to \mathbb{R}^2$ is given and $\rho : \Omega \to \mathbb{R}$ has be to be identified. At present time, NS2DDV is only based on the use of MUSCL-type Finite Volume schemes for solving (7.22). Such choice is motivated by the fact that MUSCL methods are node-centered methods and second order accurate in time and space.

7.3.1 MUSCL-type methods

Considering a mesh node $\mathbf{A} \in \mathcal{X}$, we denote with $\rho_{\mathbf{A}} : [0,T] \to \mathbb{R}$ the approximation of $\rho(\cdot, \mathbf{A})$. Hence, assuming that $\mathbf{u}$ is constant on each $\Gamma_i^{\pm}$ and ρ is constant on each control volume $\mathcal{C}_{\mathbf{A}}$, we replace the equation above by

$$\partial_t \rho_{\mathbf{A}} + \frac{1}{|\mathcal{C}_{\mathbf{A}}|} \sum_{i=1}^{nt_{\mathbf{A}}} \left[\mathbf{u}_{\mathbf{i},\mathbf{A}}^{\star,-} \cdot \mathbf{n}_i^- \int_{\Gamma_i^-} \rho\, d\sigma + \mathbf{u}_{\mathbf{i},\mathbf{A}}^{\star,+} \cdot \mathbf{n}_i^+ \int_{\Gamma_i^+} \rho\, d\sigma \right] = 0\,, \tag{7.23}$$

where the $\mathbf{u}_{\mathbf{i},\mathbf{A}}^{\pm}$ are built from the finite element discretization of the velocity $\mathbf{u}$. Such FE-FV coupling method has been introduced in [4] and is insured as follows:

- Assume that the velocity is computed thanks to the BDF2 direct method (Eqs. (6.12)):
  - Considering that the velocity is discretized on $\mathbb{P}_2$ finite elements means that $\mathbf{u}$ is approached on points $\mathbf{A}$, $(\mathbf{A_i})_i$, $(\mathbf{A_{i,i+1}})_i$ and $(\mathbf{A'_i})_i$ by $\mathbf{u_A}$, $(\mathbf{u_i})_i$, $(\mathbf{u_{i,i+1}})_i$ and $(\mathbf{u'_i})_i$ respectively. Consequently, we have

$$\begin{array}{rcl} \mathbf{u}_{\mathbf{i},\mathbf{A}}^{+} & = & \overline{\mathbf{u}_{\mathbf{i},\mathbf{A}}^{+}} \;=\; \dfrac{1}{3}(\mathbf{u'_i} + \mathbf{u'_{i+1}} + \mathbf{u_{i,i+1}})\,, \\ \mathbf{u}_{\mathbf{i},\mathbf{A}}^{-} & = & \overline{\mathbf{u}_{\mathbf{i},\mathbf{A}}^{-}} \;=\; \dfrac{1}{3}(\mathbf{u'_{i-1}} + \mathbf{u'_i} + \mathbf{u_{i-1,i}})\,, \end{array} \tag{7.24}$$

  - Considering that the velocity is discretized on $\mathbb{P}_2$ finite elements means that $\mathbf{u}$ is approached on points $\mathbf{A}$, $(\mathbf{A_i})_i$ and $(\mathbf{A'_{i,i+1}})_i$ by $\mathbf{u_A}$, $(\mathbf{u_i})_i$ and $(\mathbf{u'_{i,i+1}})_i$ respectively. Consequently, we have

$$\begin{array}{rcl} \mathbf{u}_{\mathbf{i},\mathbf{A}}^{+} & = & \dfrac{11}{60}(\mathbf{u_i} + \mathbf{u_{i+1}} + \mathbf{u_A}) + \dfrac{27}{60}\mathbf{u'_{i,i+1}}\,, \\ \mathbf{u}_{\mathbf{i},\mathbf{A}}^{-} & = & \dfrac{11}{60}(\mathbf{u_{i-1}} + \mathbf{u_i} + \mathbf{u_A}) + \dfrac{27}{60}\mathbf{u'_{i-1,i}}\,, \end{array} \tag{7.25}$$

- Assume now that velocity is computed thanks to one of the BDF2 projection methods. In such case, the formulae (7.24) and (7.25) are corrected into

$$\mathbf{u}^+_{\mathbf{i},\mathbf{A}} = \overline{\mathbf{u}^+_{\mathbf{i},\mathbf{A}}} - \frac{1}{\alpha}\frac{1}{\overline{\rho_{M_i}}}(\nabla\phi)_{M_i}\,, \qquad \mathbf{u}^-_{\mathbf{i},\mathbf{A}} = \overline{\mathbf{u}^-_{\mathbf{i},\mathbf{A}}} - \frac{1}{\alpha}\frac{1}{\overline{\rho_{M_{i-1}}}}(\nabla\phi)_{M_{i-1}}\,, \tag{7.26}$$

where $\overline{\mathbf{u}^\pm_{\mathbf{i},\mathbf{A}}}$ are computed with (7.24) or (7.25) according to the velocity space discretization, α is the constant α^n in (6.17) (or (6.18) or (6.19)), and

$$\frac{1}{\overline{\rho_{M_i}}} = \begin{cases} \frac{1}{\min_\Omega(\rho^0)}\,, & \text{with method (6.16)-(6.17),} \\ \\ \frac{1}{3}\left(\frac{1}{\rho_{\mathbf{A}}} + \frac{1}{\rho_{\mathbf{A_i}}} + \frac{1}{\rho_{\mathbf{A_{i+1}}}}\right)\,, & \text{with methods (6.16)-(6.18) or (6.16)-(6.19).} \end{cases} \tag{7.27}$$

The average values of ρ on Γ_i^+ and Γ_i^- are computed as follows:

$$\int_{\Gamma_i^+} \rho\, d\sigma = \begin{cases} |\Gamma_i^+|\,\tilde{\rho}^+_{\mathbf{A_i},\mathbf{A_i}}\,, & \text{if } \mathbf{u}_\mathbf{i}^+\cdot\mathbf{n}_i^+ \le 0, \\ |\Gamma_i^+|\,\tilde{\rho}^+_{\mathbf{A},\mathbf{A_i}}\,, & \text{if } \mathbf{u}_\mathbf{i}^+\cdot\mathbf{n}_i^+ > 0, \end{cases} \tag{7.28}$$

$$\int_{\Gamma_i^-} \rho\, d\sigma = \begin{cases} |\Gamma_i^-|\,\tilde{\rho}^-_{\mathbf{A_i},\mathbf{A_i}}\,, & \text{if } \mathbf{u}_\mathbf{i}^-\cdot\mathbf{n}_i^- \le 0, \\ |\Gamma_i^-|\,\tilde{\rho}^-_{\mathbf{A},\mathbf{A_i}}\,, & \text{if } \mathbf{u}_\mathbf{i}^-\cdot\mathbf{n}_i^- > 0, \end{cases} \tag{7.29}$$

For computing the terms $\tilde{\rho}^\pm_{\mathbf{A_i},\mathbf{A_i}}$ and $\tilde{\rho}^\pm_{\mathbf{A},\mathbf{A_i}}$, we can use a monoslope approach or a multislope procedure (default).

Monoslope procedure without flux limiter

Such approach has been investigated in [4, **?**]. This feature can be activated in NS2DDV by modifying the setup file as follows:

```
% Flux limiter method
% 'NOLIM'    No flux limiter
% 'STDLIM'   Mono-slope flux limiter
% 'TAULIM'   Multi-slope flux limiter (only with
%            PARAMETERS.FV.CELLS_DESIGN = 'STARS' and
%            PARAMETERS.FV.GRADIENT_COMPUTING = 'PRECEEDING')
PARAMETERS.FV.FLUX_LIMITER = 'NOLIM';
```

Recalling the definition of $\mathbf{C_i}^\pm$ as the middle-point of $\Gamma_i^\pm$, we provide $\tilde{\rho}^\pm_{\mathbf{A},\mathbf{A_i}}$ and $\tilde{\rho}^\pm_{\mathbf{A_i},\mathbf{A_i}}$ with the following second order definition:

$$\tilde{\rho}^\pm_{\mathbf{A},\mathbf{A_i}} = \rho_\mathbf{A} + \nabla^\pm\rho_{\mathbf{A},\mathbf{A_i}}\cdot\mathbf{AC_i}^\pm\,, \qquad \tilde{\rho}^\pm_{\mathbf{A_i},\mathbf{A_i}} = \rho_{\mathbf{A_i}} + \nabla^\pm\rho_{\mathbf{A_i},\mathbf{A_i}}\cdot\mathbf{A_iC_i}^\pm\,, \tag{7.30}$$

with $\nabla^\pm\rho_{\mathbf{A},\mathbf{A_i}}$ defined as

$$\nabla^\pm\rho_{\mathbf{A},\mathbf{A_i}} = \beta\,\overline{\nabla^\pm\rho_{\mathbf{A},\mathbf{A_i}}} + (1-\beta)\,\underline{\nabla^\pm\rho_{\mathbf{A},\mathbf{A_i}}}\,, \tag{7.31}$$

$$\underline{\nabla^+\rho_{\mathbf{A},\mathbf{A_i}}} = (\nabla\rho)_{M_i}\,, \qquad \underline{\nabla^-\rho_{\mathbf{A},\mathbf{A_i}}} = (\nabla\rho)_{M_{i-1}}\,, \tag{7.32}$$

with $\beta \in [0,1]$. This parameter is set by default to $1/3$ in NS2DDV but it can be managed fixed in the setup file by the user by modifying the following line:

```
% Beta parameter in gradient reconstruction
% (should be in [0,1]; default value = 1/3)
PARAMETERS.FV.BETA = 1./3.;
```

Concerning $\overline{\nabla^\pm\rho_{\mathbf{A},\mathbf{A_i}}}$, two definitions are proposed:

- A "surrounding" approximation of $\nabla\rho$ on node $\mathbf{A}$:

$$\overline{\nabla^+\rho_{\mathbf{A},\mathbf{A_i}}} = \overline{\nabla^-\rho_{\mathbf{A},\mathbf{A_i}}} = \frac{\displaystyle\sum_{j=1}^{nt_\mathbf{A}} |M_j|\,(\nabla\rho)_{M_j}}{\displaystyle\sum_{j=1}^{nt_\mathbf{A}} |M_j|}, \qquad \forall\, i = 1,\ldots,nt_\mathbf{A}\,. \tag{7.33}$$

  This choice can be done in the setup file as follows:

```
% Gradient reconstruction method
%'PRECEEDING'   Use the gradient on the upstream cell
%'SURROUNDING'  Use an average of the gradient on the neighbour
%               cells
% WARNING: this parameter is ignored if
% PARAMETERS.FV.CELLS_DESIGN = 'SQUARES'
PARAMETERS.FV.GRADIENT_COMPUTING = 'SURROUNDING';
```

- A "downstream/upstream" approximation of $\nabla\rho$ on node $\mathbf{A}$:

$$\overline{\nabla^\pm\rho_{\mathbf{A},\mathbf{A_i}}} = (\nabla\rho)_{D^\pm_{\mathbf{A},\mathbf{A_i}}}\,, \qquad \overline{\nabla^\pm\rho_{\mathbf{A_i},\mathbf{A_i}}} = (\nabla\rho)_{U^\pm_{\mathbf{A},\mathbf{A_i}}}\,. \tag{7.34}$$

  This choice can be done in the setup file as follows:

```
% Gradient reconstruction method
%'PRECEEDING'   Use the gradient on the upstream cell
%'SURROUNDING'  Use an average of the gradient on the neighbour
%               cells
% WARNING: this parameter is ignored if
% PARAMETERS.FV.CELLS_DESIGN = 'SQUARES'
PARAMETERS.FV.GRADIENT_COMPUTING = 'PRECEEDING';
```

Finally, each $(\nabla\rho)_M$ ($M \in \mathcal{T}$) is computed as a uniform vector. Denoting $M = ABC$, $(\nabla\rho)_M$ writes as

$$(\nabla\rho)_M = \frac{1}{2|T|}\begin{pmatrix} y_\mathbf{C} - y_\mathbf{A} & y_\mathbf{A} - y_\mathbf{B} \\ x_\mathbf{A} - x_\mathbf{C} & x_\mathbf{B} - x_\mathbf{A} \end{pmatrix}\begin{pmatrix} \rho_\mathbf{B} - \rho_\mathbf{A} \\ \rho_\mathbf{C} - \rho_\mathbf{A} \end{pmatrix}. \tag{7.35}$$

Monoslope procedure with flux limiter

Such approach can be selected by modifying the setup file as follows:

```
% Flux limiter method
% 'NOLIM'    No flux limiter
% 'STDLIM'   Mono-slope flux limiter
% 'TAULIM'   Multi-slope flux limiter (only with
%            PARAMETERS.FV.CELLS_DESIGN = 'STARS' and
%            PARAMETERS.FV.GRADIENT_COMPUTING = 'PRECEEDING')
PARAMETERS.FV.FLUX_LIMITER = 'STDLIM';
```

In such case, a flux limiter is used for improving the the finite volume method. More precisely, the definition of $\tilde{\rho}^{\pm}_{\mathbf{A},\mathbf{A_i}}$ and $\tilde{\rho}^{\pm}_{\mathbf{A_i},\mathbf{A_i}}$ given in (7.30)-(7.31) is replaced by

$$\tilde{\rho}^{\pm}_{\mathbf{A},\mathbf{A_i}} = \rho_{\mathbf{A}} + \beta\,\psi(r^{\pm}_{\mathbf{A},\mathbf{A_i}}) + (1-\beta)\,\psi\left(\frac{1}{r^{\pm}_{\mathbf{A},\mathbf{A_i}}}\right), \tag{7.36}$$

$$\tilde{\rho}^{\pm}_{\mathbf{A_i},\mathbf{A_i}} = \rho_{\mathbf{A_i}} + \beta\,\psi(r^{\pm}_{\mathbf{A_i},\mathbf{A_i}}) + (1-\beta)\,\psi\left(\frac{1}{r^{\pm}_{\mathbf{A_i},\mathbf{A_i}}}\right), \tag{7.37}$$

with

$$r^{\pm}_{\mathbf{A},\mathbf{A_i}} = \frac{p^{\pm,\mathrm{down}}_{\mathbf{A},\mathbf{A_i}}}{p^{\pm,\mathrm{up}}_{\mathbf{A},\mathbf{A_i}}}, \qquad r^{\pm}_{\mathbf{A_i},\mathbf{A_i}} = \frac{p^{\pm,\mathrm{down}}_{\mathbf{A_i},\mathbf{A_i}}}{p^{\pm,\mathrm{up}}_{\mathbf{A_i},\mathbf{A_i}}}, \tag{7.38}$$

and

$$\begin{aligned} p^{\pm,\mathrm{down}}_{\mathbf{A},\mathbf{A_i}} &= \underline{\nabla^{\pm}\rho_{\mathbf{A},\mathbf{A_i}}}\cdot\mathbf{AC}^{\pm}_{\mathbf{i}}, & p^{\pm,\mathrm{down}}_{\mathbf{A_i},\mathbf{A_i}} &= \underline{\nabla^{\pm}\rho_{\mathbf{A_i},\mathbf{A_i}}}\cdot\mathbf{A_iC}^{\pm}_{\mathbf{i}}, \\ p^{\pm,\mathrm{up}}_{\mathbf{A},\mathbf{A_i}} &= \overline{\nabla^{\pm}\rho_{\mathbf{A},\mathbf{A_i}}}\cdot\mathbf{AC}^{\pm}_{\mathbf{i}}, & p^{\pm,\mathrm{down}}_{\mathbf{A_i},\mathbf{A_i}} &= \overline{\nabla^{\pm}\rho_{\mathbf{A_i},\mathbf{A_i}}}\cdot\mathbf{A_iC}^{\pm}_{\mathbf{i}}, \end{aligned} \tag{7.39}$$

Finally, $\psi : \mathbb{R} \to \mathbb{R}$ is a flux limiter function that can be defined as follows:

- Van Leer: $\psi(r) = \dfrac{r+|r|}{1+|r|}\,\mathbb{1}_{[\epsilon,+\infty[}(r)$,
- Minmod: $\psi(r) = \max(0,\min(1,r))\,\mathbb{1}_{[\epsilon,+\infty[}(r)$,
- Van Albada: $\psi(r) = \max\left(0,\dfrac{r+r^2}{1+r^2}\right)\,\mathbb{1}_{[\epsilon,+\infty[}(r)$,
- Superbee: $\psi(r) = \max(0,\min(1,2r),\min(2,r))\,\mathbb{1}_{[\epsilon,+\infty[}(r)$.

To choose the flux limiter function and the threshold ϵ (default value $\epsilon = 10^{-6}$), the user can modify the setup file as follows:

```
% Flux limiter function (ignored if PARAMETERS.FV.FLUX_LIMITER = 'NONE')
% 'MINMOD'      Min-Mod limiter
% 'VANLEER'     Van Leer limiter
% 'SUPERBEE'    Super-bee limiter (only with
%               PARAMETERS.FV.FLUX_LIMITER = 'STDLIM')
% 'VANALBADA'   Van Albada limiter (only with
%               PARAMETERS.FV.FLUX_LIMITER = 'STDLIM')
PARAMETERS.FV.FLUX_LIMITER_FUNCTION = 'VANLEER';
% Flux limiter usage threshold
% (ignored if PARAMETERS.FV.FLUX_LIMITER = 'NOLIM')
PARAMETERS.FV.EPSILON = ϵ;
```

Multislope procedure with flux limiter

Such approach can be selected by modifying the setup file as follows:

```
% Flux limiter method
% 'NOLIM'    No flux limiter
% 'STDLIM'   Mono-slope flux limiter
% 'TAULIM'   Multi-slope flux limiter (only with
%            PARAMETERS.FV.CELLS_DESIGN = 'STARS' and
%            PARAMETERS.FV.GRADIENT_COMPUTING = 'PRECEEDING')
PARAMETERS.FV.FLUX_LIMITER = 'TAULIM';
```

In such case, the definition of $\tilde{\rho}^{\pm}_{\mathbf{A},\mathbf{A_i}}$ and $\tilde{\rho}^{\pm}_{\mathbf{A_i},\mathbf{A_i}}$ given in (7.30)-(7.31) is replaced by

$$\tilde{\rho}^{\pm}_{\mathbf{A},\mathbf{A_i}} = \rho_{\mathbf{A}} + p^{\pm}_{\mathbf{A},\mathbf{A_i}} \left\| \mathbf{AC_i}^{\pm} \right\|, \qquad \tilde{\rho}^{\pm}_{\mathbf{A_i},\mathbf{A_i}} = \rho_{\mathbf{A_i}} + p^{\pm}_{\mathbf{A_i},\mathbf{A_i}} \left\| \mathbf{A_iC_i}^{\pm} \right\|, \tag{7.40}$$

with $p^{\pm}_{\mathbf{A},\mathbf{A_i}}$ and $p^{\pm}_{\mathbf{A_i},\mathbf{A_i}}$ are defined as

$$p^{\pm}_{\mathbf{A},\mathbf{A_i}} = p^{\pm,\mathrm{up}}_{\mathbf{A},\mathbf{A_i}}\, \psi\left(r^{\pm}_{\mathbf{A},\mathbf{A_i}}\right), \qquad p^{\pm}_{\mathbf{A_i},\mathbf{A_i}} = p^{\pm,\mathrm{up}}_{\mathbf{A_i},\mathbf{A_i}}\, \psi\left(r^{\pm}_{\mathbf{A_i},\mathbf{A_i}}\right), \tag{7.41}$$

with

$$r^{\pm}_{\mathbf{A},\mathbf{A_i}} = \frac{p^{\pm,\mathrm{down}}_{\mathbf{A},\mathbf{A_i}}}{p^{\pm,\mathrm{up}}_{\mathbf{A},\mathbf{A_i}}}, \qquad r^{\pm}_{\mathbf{A_i},\mathbf{A_i}} = \frac{p^{\pm,\mathrm{down}}_{\mathbf{A_i},\mathbf{A_i}}}{p^{\pm,\mathrm{up}}_{\mathbf{A_i},\mathbf{A_i}}}, \tag{7.42}$$

and

$$p^{\pm,\mathrm{up}}_{\mathbf{A},\mathbf{A_i}} = \frac{\mathbf{AC_i}^{\pm}}{\left\|\mathbf{AC_i}^{\pm}\right\|} \cdot (\nabla\rho)_{D^{\pm}_{\mathbf{A},\mathbf{A_i}}}, \qquad p^{\pm,\mathrm{up}}_{\mathbf{A_i},\mathbf{A_i}} = \frac{\mathbf{A_iC_i}^{\pm}}{\left\|\mathbf{A_iC_i}^{\pm}\right\|} \cdot (\nabla\rho)_{U^{\pm}_{\mathbf{A},\mathbf{A_i}}}, \tag{7.43}$$

$$\begin{aligned} p^{+,\mathrm{down}}_{\mathbf{A},\mathbf{A_i}} &= \frac{\mathbf{AC_i}^{+}}{\left\|\mathbf{AC_i}^{+}\right\|} \cdot (\nabla\rho)_{M_i}, & p^{-,\mathrm{down}}_{\mathbf{A},\mathbf{A_i}} &= \frac{\mathbf{AC_i}^{-}}{\left\|\mathbf{AC_i}^{-}\right\|} \cdot (\nabla\rho)_{M_{i-1}}, \\ p^{+,\mathrm{down}}_{\mathbf{A_i},\mathbf{A_i}} &= \frac{\mathbf{A_iC_i}^{+}}{\left\|\mathbf{A_iC_i}^{+}\right\|} \cdot (\nabla\rho)_{M_i}, & p^{-,\mathrm{down}}_{\mathbf{A_i},\mathbf{A_i}} &= \frac{\mathbf{A_iC_i}^{-}}{\left\|\mathbf{A_iC_i}^{-}\right\|} \cdot (\nabla\rho)_{M_{i-1}}. \end{aligned} \tag{7.44}$$

Finally, the flux limiter function can be defined as one of the following ones:

- Van Leer: $\psi(r) = \begin{cases} \dfrac{r + (\tau - 1)\, r}{(\tau - 1)\, r}, & \text{if } r > 1, \\ \dfrac{r + (\tau - 1)\, r}{1 + (\tau - 1)\, r}, & \text{if } r \in [\epsilon, 1], \\ 0\,, & \text{else,} \end{cases}$
- Minmod: $\psi(r) = \max(0, \min(1, r))\, \mathbb{1}_{[\epsilon, +\infty[}(r)$,

with τ taken as the inverse of

$$\max_{\mathbf{A} \in \mathcal{X}} \max_{i = 1, \ldots, nt_{\mathbf{A}}} \left(\frac{\mathbf{AA'_{i,i+1}} \cdot \mathbf{AA_i}}{\mathbf{AG^{+}_{A,i}} \cdot \mathbf{AA_i}}, \frac{\mathbf{AA'_{i-1,i}} \cdot \mathbf{AA_i}}{\mathbf{AG^{-}_{A,i}} \cdot \mathbf{AA_i}} \right).$$

To choose the flux limiter function and the threshold ϵ (default value $\epsilon = 10^{-6}$), the user can modify the setup file as follows:

```
% Flux limiter function (ignored if PARAMETERS.FV.FLUX_LIMITER = 'NONE')
% 'MINMOD'      Min-Mod limiter
% 'VANLEER'     Van Leer limiter
% 'SUPERBEE'    Super-bee limiter (only with
%               PARAMETERS.FV.FLUX_LIMITER = 'STDLIM')
% 'VANALBADA'   Van Albada limiter (only with
%               PARAMETERS.FV.FLUX_LIMITER = 'STDLIM')
PARAMETERS.FV.FLUX_LIMITER_FUNCTION = 'VANLEER';
% Flux limiter usage threshold
% (ignored if PARAMETERS.FV.FLUX_LIMITER = 'NOLIM')
PARAMETERS.FV.EPSILON = ϵ;
```

Some details about the numerical behaviour of such method can be found in [3].

8 Visualization and post-processing

8.1 In-situ visualization

NS2DDV offers some in-situ visualization tools, *i.e.* the ability to visualize the numerical results in Matlab figures as soon as they are calculated. To activate this tool, the user must specify the following parameters in the setup file before running the simulation:

```
PARAMETERS.OUTPUT.XDISPLAY = 'SINGLE_FIGURE';
```

or

```
PARAMETERS.OUTPUT.XDISPLAY = 'MULTIPLE_FIGURE';
```

The `'SINGLE_FIGURE'` option produces a single Matlab figure window in which the results are plotted in subplot regions, and the `'MULTIPLE_FIGURE'` option produces a Matlab figure window per 2D result that is asked for plotting.

Remark 8.1. According to the Matlab release version, the `'SINGLE_FIGURE'` may be unstable.

If one of these two solutions is chosen for in-situ visualization, it is possible to select a list of 2D diagnostics to be plotted. To do this, the user shall modify the list `PARAMETERS.OUTPUT.XDISPLAY_LIST` in the setup file. Such diagnostics are listed in Table 4. When a manufactured test case is run (EXAC or EXACNEU), additional 2D diagnostics linked to the exact solution are available (see Table 5). For example, for visualizing the mesh, the velocity **u** as a vector field, the pressure and its first order derivative, the user should set

```
% List of 2D results to be plotted :
% ...
PARAMETERS.OUTPUT.XDISPLAY_LIST = {'MESH', 'U_VECTOR', 'P', ...
                                   'P_DX', 'P_DY'};
```

In addition, it is possible to set the refreshing frequency of the figure windows. By default, they are refreshed after each iteration of the time solver. To modify this refreshing frequency, the user must modify the following parameter in the setup file:

```
PARAMETERS.OUTPUT.XDISPLAY_FREQUENCY = 1;
```

8.2 Saving numerical results

NS2DDV can save the numerical results of each run under a sequence of files with Matlab binary (`.mat`) or HDF5 (`.h5`) format.

According to the Matlab release that is used, NS2DDV will choose outputs under `.mat`

(Matlab R2010b and older) or `.h5`.

To manage these output files, the user can modify the following parameters in the setup file:

```
% Directory for numerical results
PARAMETERS.OUTPUT.DIRECTORY_NAME = '/path/to/results';
% Prefix of output file(s)
PARAMETERS.OUTPUT.FILE_NAME = 'diags';
% Save method of these results :
% 'LAST_FRAME'  The solution is only saved at final time
% 'ANIMATION'   If time dynamics is considered, an animated result
%               is saved
% 'NONE'        Only the log file is saved
PARAMETERS.OUTPUT.XDISPLAY_SAVE = 'NONE';
```

According to the choice for `PARAMETERS.OUTPUT.XDISPLAY_SAVE`, NS2DDV will produce a specific output file set:

- `'ANIMATION'`: a set files with name `diags_n.h5` (or `.mat`) is generated, where `diags` is the prefix that is chosen for `PARAMETERS.OUTPUT.FILE_NAME` and n is the file number.
  Such choice is relevant for studying and/or visualizing the time dynamics of a simulation after it has been fully computed.

  Each of these files is associated to the results of a unique iteration in the time solver. By default, the results are saved for each time iteration but it is possible to manage this save frequency with the following setup parameter:

  ```
  % Refreshing frequency for save 2D results (default = 1)
  PARAMETERS.OUTPUT.XDISPLAY_SAVE_FREQUENCY = 1;
  ```

  Each of these files is also associated to a mesh file that contains the data of the mesh on which the results can be plotted. Such mesh file is usually named with the syntax `diags_MESH_k.h5`, where k stands for the k-modification of the initial mesh[3]. For insuring visualization and/or postprocessing, it is highly recommended to put the output result file and the associated mesh files in the same directory. In the case of a non-adaptative mesh, there is a unique mesh file named `diags_MESH_0.h5`.

  If HDF5 files are generated, each output result result file is associated with a Xdmf decriptor file (same file name, but `.xdmf` extension). These reader files are mandatory for visualization with external visualization softwares like Visit.

[3]Numerical methods involving adaptative meshes are not implemented yet. Consequently a unique mesh file with be produced.

- 'LAST_FRAME': the output results are structured in the same way as for 'ANIMATION' but only last time iteration is saved.
  Such choice is relevant for developing new branches in the code and validating the computations.

- 'NONE': no .h5/.mat file is produced.
  Such choice is relevant for performing a numerical convergence study where only log files are necessary.

If the solution 'ANIMATION' or 'LAST_FRAME' has been selected, it is possible to enrich the list of 2D diagnostics to be saved. To do this, the user shall fill the list PARAMETERS.OUTPUT.XDISPLAY_SAVE_OPTDIAGS in the setup file with some fields listed in Table 2 (and Table 3 if a manufactured test case is considered). For example, if the user wants to save the first order derivatives of the pressure in addition of the default saves, (s)he must write the following line in the setup file:

```
% List of optional diagnostics
PARAMETERS.OUTPUT.XDISPLAY_SAVE_OPTDIAGS = {'P_DX', 'P_DY'};
```

Note that selecting 'GRAD_P', 'P_DX' or 'P_DY' will automatically save the three fields. A same automatism is applied to the following field sets:

- 'GRAD_P', 'P_DX' and 'P_DY',
- 'GRAD_P_EX', 'P_DX_EX' and 'P_DY_EX',
- 'W_VECTOR', 'WX' and 'WY'.

8.3 Visualization from a result file

NS2DDV includes some solutions for visualizing the contents of .mat/.h5 files.

By default, the user can run the Matlab routines that are embedded in NS2DDV sources. These routines are dedicated to a simple visualization of results without saving.

In addition, it is possible to use the Python-Matplotlib scripts provided in the subdirectory PYTHON in the NS2DDV source files.

8.3.1 Matlab solution

NS2DDV embeds Matlab routines named plot_from_file and movie_from_file that allow to plot some 2D diagnostics from a single .mat or .h5 file or visualize the dynamics from a file set that has been generated by a previous run of the code.

To plot the 2D results from a single file, the user must move to the root path of NS2DDV, then type the following commands:

2D diagnostic	Name in the setup file	Restrictions	Saving in `.h5`/`.mat` files
Mesh	`'MESH'`	-	By default
Density ρ (scalar)	`'RHO'`	Only with NSDV model	By default
Velocity $\mathbf{u}$ (vector)	`'U_VECTOR'`	-	By default
x-part of velocity u_x (scalar)	`'UX'`	-	By default
y-part of velocity u_y (scalar)	`'UY'`	-	By default
$\partial_x u_x$ (scalar)	`'DX_UX'`	-	Optional
$\partial_y u_x$ (scalar)	`'DY_UX'`	-	Optional
$\partial_x u_y$ (scalar)	`'DX_UY'`	-	Optional
$\partial_y u_y$ (scalar)	`'DY_UY'`	-	Optional
Shear rate $\partial_y u_x + \partial_x u_y$ (scalar)	`'SHEAR'`	-	Optional
Velocity magnitude $\lvert\mathbf{u}\rvert^2$ (scalar)	`'U_MODULE'`	-	By default
Vorticity ω (scalar)	`'VORTIC'`	-	By default
Velocity streamlines ψ (scalar)	`'STREAM'`	-	By default
Pressure p (scalar)	`'P'`	-	By default
Pressure gradient ∇p (vector)	`'P_GRAD'`	-	Optional
$\partial_x p$ (scalar)	`'P_DX'`	-	Optional
$\partial_y p$ (scalar)	`'P_DY'`	-	Optional
Auxiliary velocity $\mathbf{w}$ (vector)	`'W_VECTOR'`	Only with Absorbing boundary conditions	Optional
x-part of auxiliary velocity w_x (scalar)	`'WX'`	Only with Absorbing boundary conditions	Optional
y-part of auxiliary velocity w_y (scalar)	`'WY'`	Only with Absorbing boundary conditions	Optional
Auxiliary pressure r (scalar)	`'R'`	Only with Absorbing boundary conditions	Optional

Table 2: List of 2D diagnostics

2D diagnostic	Name in the setup file	Restrictions	Saving in `.h5`/`.mat` files
Exact density ρ_{ex} (scalar)	`'RHO_EX'`	Only with NSDV model	Optional
Exact velocity $\mathbf{u}_{ex}$ (vector)	`'U_VECTOR_EX'`	-	By default
x-part of exact velocity $u_{x,ex}$ (scalar)	`'UX_EX'`	-	By default
y-part of exact velocity $u_{y,ex}$ (scalar)	`'UY_EX'`	-	By default
$\partial_x u_{x,ex}$ (scalar)	`'DX_UX_EX'`	-	Optional
$\partial_y u_{x,ex}$ (scalar)	`'DY_UX_EX'`	-	Optional
$\partial_x u_{y,ex}$ (scalar)	`'DX_UY_EX'`	-	Optional
$\partial_y u_{y,ex}$ (scalar)	`'DY_UY_EX'`	-	Optional
Exact shear rate $\partial_y u_{x,ex} + \partial_x u_{y,ex}$ (scalar)	`'SHEAR_EX'`	-	Optional
Exact velocity magnitude $\lvert\mathbf{u}_{ex}\rvert^2$ (scalar)	`'U_MODULE_EX'`	-	By default
Exact vorticity ω_{ex} (scalar)	`'VORTIC_EX'`	-	By default
Exact velocity streamlines ψ_{ex} (scalar)	`'STREAM_EX'`	-	By default
Exact pressure p_{ex} (scalar)	`'P_EX'`	-	By default
Exact pressure gradient ∇p_{ex} (vector)	`'P_GRAD_EX'`	-	Optional
$\partial_x p_{ex}$	`'P_DX_EX'`	-	Optional
$\partial_y p_{ex}$	`'P_DY_EX'`	-	Optional

Table 3: List of specific 2D diagnostics for manufactured test cases (EXAC, EXACNEU)

```
>> load_paths();

>> plot_from_file(namefile, XDISPLAY_LIST, LEVELS, XDISPLAY_METHOD, ...
typeplot, nbrows, nbcols);
```

The first command is useful for loading all subdirectories of NS2DDV and to access to all routines within the sources. Once it is done, the user can run `plot_from_file` with the following arguments:

- `namefile` (string) is the name of the file to be read (single `.mat` or `.h5` file),
- `XDISPLAY_LIST` is a cell line where each cell is a diagnostic to be plotted (see Tables 4-5). Here is a example of such list:

```
>> XDISPLAY_LIST = {'MESH', 'P', 'UX', 'UY', 'U_VECTOR'};
```

- `LEVELS` is a cell line that contains the isovalues for each scalar diagnostic. This cell line must have the same size as `XDISPLAY_LIST`. If the user wants to fix the isovalues, (s)he must provide an line vector containing the required isovalues. In the other cases, the user must provide an empty vector. Here is a example of such list associated to the example of `XDISPLAY_LIST` above:

```
>> LEVELS = {[], [0., 1., 2.], [], [], []};
```

  Here, we fix the isovalues for the pressure and leave them to the computer for the other diagnostics.
  Note that, if `XDISPLAY_LIST{i}` is a vector field, the value of `LEVELS{i}` is not taken into account (see the example below for further details).

- `XDISPLAY_METHOD` (string) must be either `'SINGLE_FIGURE'` or `'MULTIPLE_FIGURE'`. The `'SINGLE_FIGURE'` option produces a single Matlab figure window in which the results are plotted in subplot regions, and the `'MULTIPLE_FIGURE'` option produces a Matlab figure window per 2D result that is asked for plotting.

- `typeplot` (string) must be either `'contour'` or `'pseudocolor'`. It will specify if the plot mode for scalar diagnostics.

  Remark 8.2. At present time, the diagnostics `'P_DX'`, `'P_DY'`, `'P_DX_EX'` and `'P_DY_EX'` are automatically plotted in `'pseudocolor'` mode. We are currently looking for a `'contour'` solution for these diagnostics.

- `nbrows` and `nbcols` (integers) are respectively the number of rows and columns that define the mosaic of graphs in the case where the `'SINGLE_FIGURE'` display method is chosen. They are mandatory if this display method is chosen, and optional if `'MULTIPLE_FIGURE'` is chosen instead.

Example 8.3. This example is based on the Matlab script `EXAMPLES/plot_single_nona.m`: to run it, the user must start Matlab in the root directory of NS2DDV, then run the following command:

```
>> run('./EXAMPLES/plot_single_nona.m')
```

The idea is to study the contents of the file `RESULTS/EXAMPLES/test_nona/diags_5.h5`.

More precisely, we want to plot the $\mathbb{P}_1$ mesh, the vector field $\mathbf{u}$, the pressure p and its first order derivatives. In addition, we want to focus on the isovalues $p = -1, 0, \ldots, 3$. This is why we can find the following lines within the script `EXAMPLES/plot_single_nona.m`:

```
load_paths()

XDISPLAY_LIST = {'MESH', 'U_VECTOR', 'P', 'P_DX', 'P_DY'};

LEVELS = {[], [1], [-1., 0., 1., 2., 3.], [], []};

namefile = './RESULTS/EXAMPLES/test_nona/diags_5.h5';

plot_from_file(namefile, XDISPLAY_LIST, LEVELS, 'SINGLE_FIGURE', ...
'contour', 3, 2);
```

8.3.2 Python-Matplotlib solution (no-display)

NS2DDV is supplemented with some Python scripts for generating plots from the result files. Since some third-party Python modules are needed, the user can use the Python installer `pip` to install the following packages:

- numpy,
- h5py,
- decimal,
- shutil,
- matplotlib.

Another way to ensure the dependencies above is to install Python-Anaconda distribution. More informations are available at the URL below:

`http://anaconda.org/anaconda/python`

In any case, the user should use Python 3.x instead of Python 2.x since there is no insurance that Python 2.x support will be ensured in the long term.

For using the Python-Matplotlib functions provided along with NS2DDV, the user must implement a Python script that should start with the following lines:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
sys.path.insert(0, 'PATH/TO/NS2DDV/PYTHON')

# If you want to plot 2D results, uncomment the following line
#from plots import *

# If you want to to encode movies, uncomment the following line
#from make_movie import *

# If you want the plot the contents of log files, uncomment the
# following line
#from plot_overtime_log import *

# ... then, type the main script below
```

For running such Python script under Linux-type OS or MacOS, the user must open a terminal, then type the following command:

```
$ python thescript.py
```

Plotting diagnostics over time NS2DDV is provided with some Python functions for exploring log files and plot the evolution in time of some quantities that are listed in such files. In addition, it is possible to compare several simulations in such terms.

Example 8.4. In this example, we aim to plot the evolution of the kinetic energy for a unique simulation.
To do this, we first read the corresponding log file, then extract the time grid and the discrete kinetic energy. Finally, we set the title, the legend and the limits of the bounding box in y direction before plotting the data and saving the figure.
It is possible to compare several simulations in terms of kinetic energy. To do this, we extract the data, set the title, the legend etc. just as above for each log file, and we finally use the routine `plot_data` in a little bit different way. For the details, see the script below:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Import the module for plotting the contents of log files
from plot_overtime_log import *
```

```
# Read log files
namelogfile1 = 'PATH/TO/NS2DDV/RESULTS/diags1_log'
[headers1, data1] = read_logfile(namelogfile1)
namelogfile2 = 'PATH/TO/NS2DDV/RESULTS/diags2_log'
[headers2, data2] = read_logfile(namelogfile2)

# Extract time grid
time1 = extract_data(headers1, data1, 'TIME')
time2 = extract_data(headers2, data2, 'TIME')

# Extract the discrete kinetic energy
ekin1 = extract_data(headers1, data1, 'KINETIC ENERGY')
ekin2 = extract_data(headers2, data2, 'KINETIC ENERGY')

# Define legends
leg1 = 'Kinetic energy (run 1)'
leg2 = 'Kinetic energy (run 2)'

# Title of figure
ti = 'Kinetic energy'

# Leave the limits in y-direction to the computer
ylim = 'default'

# Output PNG file
outputpng1 = './temp1.png'
outputpng12 = './temp12.png'

# Plot the kinetic energy of run 1
plot_data([ekin1], [time1], [leg1], ti, ylim, outputpng1)

# Plot the kinetic energy for both runs
plot_data([ekin1, ekin2], [time1, time2], [leg1, leg2], ti, ylim, \
          outputpng12)
```

Example 8.5. In a second example, we assume that we have a set of log files and we want to compare them in the same terms in various ways. We take the same context as in the first example above and we assume that the kinetic energy from the second log file is much larger than the kinetic energy in the first log file. Thanks to the following script, it is possible to plot in one line:

- The kinetic energy from the first log file,
- The kinetic energy from the second log file,
- The kinetic energy from both log files,

... and save the produced figures in 3 separated PNG files.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Import the module for plotting the contents of log files
from plot_overtime_log import *

# Log file names
namelogfile1 = 'PATH/TO/NS2DDV/RESULTS/diags1_log'
namelogfile2 = 'PATH/TO/NS2DDV/RESULTS/diags2_log'

# Define legends
leg1 = 'Kinetic energy (run 1)'
leg2 = 'Kinetic energy (run 2)'

# Title of figure
ti = 'Kinetic energy'

# Define log file sets
logfileset1 = [namelogfile1]
logfileset2 = [namelogfile2]
logfileset12 = [namelogfile1, namelogfile2]

# Define legend sets
legset1 = [leg1]
legset2 = [leg2]
legset12 = [leg1, leg2]

# Output PNG files
outputpng1 = './temp1.png'
outputpng2 = './temp2.png'
outputpng12 = './temp12.png'

# Plot the 3 graphs
compare_plots([logfileset1, logfileset2, logfileset12], \
              [legset1, legset2, legset12], \
              'KINETIC_ENERGY', 'Kinetic energy', \
              [outputpng1, outputpng2, outputpng12])
```

In the following lines, we describe the prototype for the Python functions that are involved in the examples above

```
[allheaders, alldata] = read_logfile(logfile)
```

- `logfile` is an input string containing the name of the log file to be read.
- `allheaders` is a list of strings that is no more than the name of all diagnostics that are in the logfile.
- `alldata` is a list of lists, where `data[i]` is the list containing the time evolution of the diagnostic with name `headers[i]`.

```
thedata = extract_data(allheaders, alldata, theheader)
```

- `allheaders` is the whole list of strings mentioned above.
- `alldata` is the Python list of lists containing all diagnostics mentioned above.
- `theheader` is the name of the diagnostic to be plotted.
- `thedata` is the list corresponding to the time evolution of the diagnostic with name `theheader`.

```
plot_data(datas, times, legends, ytext, ylim, outputfile)
```

- `datas`, `times`, and `legends` must be lists with the same size.
- `datas` and `times` must be lists of lists such that `datas[i]` and `times[i]` have same size.
- `legends[i]` is the legend for the curve with abscissa `times[i]` and ordinates `datas[i]`. It should be a string.
- `ytext` is the name of ordinate axis. It should be a string.
- `ylim` is either a list of 2 floats exactly or the string `'default'`. If 2 floats $y_{\min}$ and $y_{\max}$ are given, the plotting limits in y-direction will be set to $y_{\min}$ and $y_{\max}$. If `'default'` is provided, Python will manage these limits according to the contents of `datas`.
- `outputfile` is a string containing the full name of the output PNG file to be generated (absolute path is recommended).

```
compare_plots(logfilesets, legendsets, field, ytext, outputfiles)
```

- `logfilesets` is a list (size N) of lists of log file names.
- `legendsets` is a list (size N) of lists of legends. This argument must have the same shape as `logfilesets`.
- `field` is the name of the diagnostic to be analyzed.
- `ytext` is the name of ordinate axis.
- `outputfiles` is a list of size N of output PNG files.

In addition, a function dedicated to the NONA test case (see paragraph 5.2.7) focuses on signal diagnostics that are listed in such log file:

```
compare_signals_sets(logfilesets, legendsets, fields, ytexts, \
                     xsignal, ysignal, xsignalalias, ysignalalias, \
                     outputprefixes)
```

- `logfilesets` is a list (size N) of lists of log file names.
- `legendsets` is a list (size N) of lists of legends. This argument must have the same shape as `logfilesets`.
- `fields` is a list of diagnostics to be analyzed. Each element of this list must be `'P'`, `'UX'` or `'UY'`.
- `ytexts` is a list (size N) of strings where `ytexts[i]` is the name of ordinate axis for the i-th log file set.
- `xsignal` and `ysignal` are lists of floats with respective sizes N_x and N_y.
- `xsignalalias` and `ysignalalias` are lists of strings with respective sizes N_x and N_y.
- `outputprefixes` is a list (size N) of strings that contains the prefix names of output PNG files.

For running this routine, the input arguments should satisfy some rules:

- Denoting x_i =`xsignal[i]`, y_j =`ysignal[j]` and `f = fields[k]`, the string `f(`x_i`,`y_j`)` must be one of the diagnostics stored in every log file in `logfilesets`.
- Denoting `xi = xsignalalias[i]`, `yj = ysignalalias[j]`, `fk = fields[k]` and `pl = outputprefixes[l]` for any $i = 0, \ldots, N_x - 1$, $j = 0, \ldots, N_y - 1$, $l = 0, \ldots, N - 1$ the routine will generate the PNG file with name `pl_fk_xi_yj.png`.

One can also have a look to the following example:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Import the module for plotting the contents of log files
from plot_overtime_log import *

# Log file names
namelogfile1 = 'PATH/TO/NS2DDV/RESULTS/diags1_log'
namelogfile2 = 'PATH/TO/NS2DDV/RESULTS/diags2_log'

# Define legends
leg1 = 'Kinetic energy (run 1)'
leg2 = 'Kinetic energy (run 2)'

# Title of figure
ti = 'Kinetic energy'

# Define log file sets
logfileset1 = [namelogfile1]
```

```
logfileset2 = [namelogfile2]
logfileset12 = [namelogfile1, namelogfile2]

# Define legend sets
legset1 = [leg1]
legset2 = [leg2]
legset12 = [leg1, leg2]

# Prefixes for output PNG files
outputprefix1 = './log1'
outputprefix2 = './log2'
outputprefix12 = './log12'

# We aim to plot ux(3,1), ux(4,1), uy(3,1) and uy(4,1)
# Specify fields in log file
fields = ['UX', 'UY']
# Associate them to aliases for legends
ytexts = ['ux', 'uy']
# x-component of the signals
xsignal = [3., 4.]
# y-component of the signals
ysignal = [1.]
# Associate them to aliases for legends
xsignalalias = ['3', '4']
ysignalalias = ['1']


compare_signals_sets([logfileset1, logfileset2, logfileset12], \
                     [legset1, legset2, legset12], \
                     fields, ytexts, \
                     xsignal, ysignal, xsignalalias, ysignalalias, \
                     ['./log1', './log2', './log12'])
```

Plotting 2D results (only for results under `.h5` format) Some Python functions are proposed for visualizing 2D results that are stored in HDF5 files.

Example 8.6. The first example is about visualizing some 2D diagnostics from a unique HDF5 file. It is possible to plot a scalar data such as the pressure p or the components of the velocity u_x and u_y with isovalues or with a color map. A typical script for this goal is given below (see also the file `EXAMPLES/plot_single_scalar_step_neumann.py`):

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

############################ WARNING ############################
# For running this script with Linux or MacOS, open a terminal in
# the subdirectory EXAMPLES, then type the command
#
```

```
#    python plot_single_scalar_step_neumann.py
#
#########################################################################

import sys
sys.path.insert(0, '../PYTHON')

# Import the module for plotting the contents of HDF5 files
from plots import *

# We aim to plot the contents of the following file:
inputh5 = '../RESULTS/EXAMPLES/step_neumann/diags_10.h5'

# ... and more precisely the x-component of the velocity
field = 'Velocity_x'


# Simple contour plot
plot_contour(inputh5, field, aliasfield='ux', \
             outputpng='./ux_contour.png', forcesave=True)

# Same with a prescribed bounding box
plot_contour(inputh5, field, aliasfield='ux', \
             list_bbox=[0.,30.,0.,3.], \
             outputpng='./ux_contour_bbox.png', forcesave=True)

# Same with prescribed isovalues
plot_contour(inputh5, field, aliasfield='ux', levels=[0.,0.5,1.], \
             outputpng='./ux_contour_levels.png', forcesave=True)


# Simple pseudocolor plot
plot_pseudocolor(inputh5, field, aliasfield='ux', \
                 outputpng='./ux_pseudocolor.png', forcesave=True)

# Same with a prescribed bounding box
plot_pseudocolor(inputh5, field, aliasfield='ux', \
                 list_bbox=[0.,30.,0.,3.], \
                 outputpng='./ux_pseudocolor_bbox.png', forcesave=True)

# Same with prescribed bounds of colorbar
plot_pseudocolor(inputh5, field, aliasfield='ux', list_minmax=[0.,1.], \
                 outputpng='./ux_pseudocolor_minmax.png', \
                 forcesave=True)
```

We describe in the following lines the prototype of the functions that have been used in the example above:

```
plot_contour(inputh5, field, \
             aliasfield='', \
             levels=[], \
             list_bbox=['default']*4, \
             tuple_anchor_legend=(1.01,1.01), \
             tuple_inches_figsize=(12., 12.), \
             outputpng='./temp.png', \
             forcesave=False)
```

Mandatory arguments

- `inputh5` (string) is the input HDF5 file to be studied.
- `field` (string) is the field within the HDF5 file to be plotted. Such field must be written one of the scalar diagnostics in Tables 4-5.

Optional arguments

- `aliasfield` (string) is the field name as it will be used as plot title and legend. Leaving the default value (empty string) means that this alias will be exactly `field`.
- `levels` (list of floats) is the list of isovalues to be plotted.
  The default value (an empty list) means that a set of 10 equally distributed isovalues will be identified accordingly with the studied data.
- `list_bbox` is a list of 4 elements that can be either floats or `'default'`. These 4 elements stands for $x_{\min,bb}$, $x_{\max,bb}$, $y_{\min,bb}$ and $y_{\max,bb}$ respectively which will be used for the visualization bounding box $[x_{\min,bb}, x_{\max,bb}] \times [y_{\min,bb}, y_{\max,bb}]$.
  Any `'default'` value means that the corresponding part of the bounding box will be identified thanks to the used nodes.
- `tuple_anchor_legend` is a Python tuple constituted of 2 floats that specifies the relative position of the legend according the upper left corner of the bounding box.
  The default value is set to $(1.01, 1.01)$.
- `tuple_inches_figsize` is a Python tuple constituted of 2 floats that are respectively the width and the height of the output PNG file in inches (by default, an inch corresponds to 80 pixels).
  The default value is set to $(12, 12)$.
- `outputpng` (string) is the name of the output PNG file.
  The default output name is `temp.png` and is located in the current directory.
- `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`.
  The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

```
plot_pseudocolor(inputh5, field, \
                 aliasfield='', \
                 list_minmax=[], \
                 list_bbox=['default']*4, \
                 tuple_leg_position=(0.8,0.45,0.2,0.3), \
                 tuple_inches_figsize=(12.,12.), \
                 outputpng='./temp.png', \
                 forcesave=False)
```

Mandatory arguments

- `inputh5` (string) is the input HDF5 file to be studied.
- `field` (string) is the field within the HDF5 file to be plotted. Such field must be written one of the scalar diagnostics in Tables 4-5.

Optional arguments

- `aliasfield` (string) is the field name as it will be used as plot title and legend. Leaving the default value (empty string) means that this alias will be exactly `field`.
- `list_minmax` is a list of 2 floats that will be the bounds of the colormap. Leaving the default value (empty list) means that `ad hoc` colormap bounds will be used.
- `list_bbox` is a list of 4 elements that can be either floats or `'default'`. These 4 elements stands for $x_{\min,bb}$, $x_{\max,bb}$, $y_{\min,bb}$ and $y_{\max,bb}$ respectively which will be used for the visualization bounding box $[x_{\min,bb}, x_{\max,bb}] \times [y_{\min,bb}, y_{\max,bb}]$. Any `'default'` value means that the corresponding part of the bounding box will be identified thanks to the used nodes.
- `tuple_legend_position` is a Python tuple constituted of 4 floats that specifies the relative position and size of the legend according to the upper left corner of the bounding box with the format $(x, y, width, height)$ in units that are relative to the bounding box.
  The default value is set to $(0.8, 0.45, 0.2, 0.3)$.
- `tuple_inches_figsize` is a Python tuple constituted of 2 floats that are respectively the width and the height of the output PNG file in inches (by default, an inch corresponds to 80 pixels).
  The default value is set to $(12, 12)$.
- `outputpng` (string) is the name of the output PNG file.
  The default output name is `temp.png` and is located in the current directory.
- `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`.

The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

Example 8.7. A second example is an extension of the previous one to the visualization of vector fields (see the file `EXAMPLES/plot_single_vector_step_neumann.py`):

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

############################ WARNING ############################
# For running this script with Linux or MacOS, open a terminal in
# the subdirectory EXAMPLES, then type the command
#
#    python plot_single_vector_step_neumann.py
#
#################################################################

import sys
sys.path.insert(0, '../PYTHON')

# Import the module for plotting the contents of HDF5 files
from plots import *

# We aim to plot the contents of the following file:
inputh5 = '../RESULTS/EXAMPLES/step_neumann/diags_10.h5'

# ... and more precisely the velocity vector field
field_x = 'Velocity_x'
field_y = 'Velocity_y'


# Simple vector plot
plot_vectorfield(inputh5, field_x, field_y, aliasfield='u', \
                 outputpng='./u_vector.png', forcesave=True)

# Same with a prescribed bounding box
plot_vectorfield(inputh5, field_x, field_y, aliasfield='u', \
                 list_bbox=[0., 30., 0., 3], \
                 outputpng='./u_vector_bbox.png', forcesave=True)

# Same with prescribed number of arrows
plot_vectorfield(inputh5, field_x, field_y, aliasfield='u', \
                 narrows=400, outputpng='./u_vector_arrows.png', \
                 forcesave=True)
```

The visualization function used in the example above is the following one:

```
plot_vectorfield(inputh5, field_x, field_y, \
                 aliasfield='', \
                 narrows=200, \
                 scale=2., \
                 width=0.01, \
                 list_bbox=['default']*4, \
                 tuple_leg_position=(0.8,0.45,0.2,0.3), \
                 tuple_inches_figsize=(12.,12.), \
                 outputpng='./temp.png', \
                 forcesave=False)
```

Mandatory arguments

- `inputh5` (string) is the input HDF5 file to be studied.
- `field_x` and `field_y` (strings) are the scalar fields within the HDF5 file to be used as the components of the vector field to be plotted. Such fields must be some of the scalar diagnostics in Tables 4-5 and they must be discretized on the same grid.

Optional arguments

- `aliasfield` (string) is the field name as it will be used as plot title and legend. Leaving the default value (empty string) means that this alias will be exactly `(field_x,field_y)`.
- `narrows` (integer) gives an indication of the number of arrows to be plotted. These arrows will be uniformly distributed on the mesh and colored by their length.
  Default value is set to 200.
- `scale` (float) is the number of data unit per arrow length unit. Taking a value smaller than 1 makes the arrows longer.
  Default value is set to 2.
- `width` (float) is the shaft width in arrow units. One can start with the value equal to 0.005 times the length of the domain bounding box diagonal.
  Default value is set to 0.01.
- `list_bbox` is a list of 4 elements that can be either floats or `'default'`. These 4 elements stands for $x_{\min,bb}$, $x_{\max,bb}$, $y_{\min,bb}$ and $y_{\max,bb}$ respectively which will be used for the visualization bounding box $[x_{\min,bb}, x_{\max,bb}] \times [y_{\min,bb}, y_{\max,bb}]$.
  Any `'default'` value means that the corresponding part of the bounding box will be identified thanks to the used nodes.
- `tuple_legend_position` is a Python tuple constituted of 4 floats that specifies the relative position and size of the legend according to the upper left corner of the bounding box with the format $(x, y, width, height)$ in units that are relative to the bounding box.
  The default value is set to $(0.8, 0.45, 0.2, 0.3)$.

- `tuple_inches_figsize` is a Python tuple constituted of 2 floats that are respectively the width and the height of the output PNG file in inches (by default, an inch corresponds to 80 pixels).
  The default value is set to $(12, 12)$.
- `outputpng` (string) is the name of the output PNG file.
  The default output name is `temp.png` and is located in the current directory.
- `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`.
  The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

Note that, for using `plot_contour`, it can be useful to prescribe a specific list of isovalues. To build such list, one can extract the minimal and maximal values of the studied field, then build a list of values from these bounds. To do this, the user can call the following function:

```
list_minmax = extract_minmax(inputh5, field)
```

where `inputh5` (string) is the HDF5 field where the data is stored, `field` (string) is the 2D scalar field to be studied, and the output `list_minmax` is a list constituted of 2 floats that are respectively the minimal and maximal bounds of the studied field.
Note that such function is also useful for prescribing the colormap bound in a use of the function `plot_pseudocolor`.

It is possible to extend the jobs above to a HDF5 file set:

Example 8.8. The following example is about visualizing some 2D diagnostics from a set of HDF5 files instead of a single file. A typical script for this goal is given below and is a part of `EXAMPLES/movie_nona.py`:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

############################ WARNING ############################
# For running this script with Linux or MacOS, open a terminal in
# the subdirectory EXAMPLES, then type the command
#
#    python movie_nona.py
#
#################################################################

import sys
sys.path.insert(0, '../PYTHON')

# Import the module for plotting the contents of HDF5 files
```

```
from plots import *



# The input files are RESULTS/EXAMPLES/test_nona/diags_i.h5 with
# i=0,...,12
dir_inputh5 = '../RESULTS/EXAMPLES/test_nona'
istart = 0
iend = 12

# We aim to visualize the dynamics of
# - ux with contours
# - uy with contours
# - p with pseudocolor
# - u with vectors


# Warning: the size of output PNGs should be prescribed
figsize = (19.2,12.)

# Build the list of input files
inputh5set = []
for i in range(istart, iend+1):
        inputh5set.append(dir_inputh5+'/diags_'+str(i)+'.h5')

# Plot the contour of ux
# (Default: 10 isovalues that do not change with time)
plot_contour_set(inputh5set, 'Velocity_x', aliasfield='ux', \
                 tuple_inches_figsize=figsize, outputdir='./tmp', \
                 forcesave=True)

# Plot the contour of uy
# (Default: 10 isovalues that do not change with time)
plot_contour_set(inputh5set, 'Velocity_y', aliasfield='uy', \
                 tuple_inches_figsize=figsize, outputdir='./tmp', \
                 forcesave=True)

# Plot the pseudocolor of p
# (Default: bounds of colormap do not change with time)
plot_contour_set(inputh5set, 'Pressure', aliasfield='p', \
                 tuple_inches_figsize=figsize, outputdir='./tmp', \
                 forcesave=True)

# Plot the vector field u
# (Default: 200 arrows)
plot_vectorfield_set(inputh5set, 'Velocity_x', 'Velocity_y', \
                     aliasfield='u', tuple_inches_figsize=figsize, \
                     outputdir='./tmp', forcesave=True)
```

We describe in the following lines the prototype of the functions that have been used in the example above:

```
plot_contour_set(inputh5set, field, \
                 aliasfield='', \
                 nlevels=10, \
                 fix_isovalues=True, \
                 list_bbox=['default']*4, \
                 tuple_anchor_legend=(1.01,1.01), \
                 tuple_inches_figsize=(12., 12.), \
                 outputdir='.', \
                 forcesave=False)
```

Mandatory arguments

- `inputh5set` is a list that contains the name of all HDF5 files to be studied.
- `field` (string) is the field within both HDF5 files to be plotted. Such field must be written one of the scalar diagnostics in Tables 4-5.

Optional arguments

- `aliasfield` (string) is the field name as it will be used as plot title and legend. Leaving the default value (empty string) means that this alias will be exactly `field`.
- `nlevels` (integer) is the number of isovalues to be plotted on each figure.
  The default value is set to 10.
- `fix_isovalues` (boolean) is for considering the same isovalues for the whole file set or not.
  The default value is `True`, meaning that the isovalues are not modified over the while file set.
- `list_bbox` is a list of 4 elements that can be either floats or `'default'`. These 4 elements stands for $x_{\min,bb}$, $x_{\max,bb}$, $y_{\min,bb}$ and $y_{\max,bb}$ respectively which will be used for the visualization bounding box $[x_{\min,bb}, x_{\max,bb}] \times [y_{\min,bb}, y_{\max,bb}]$.
  Any `'default'` value means that the corresponding part of the bounding box will be identified thanks to the used nodes.
- `tuple_anchor_legend` is a Python tuple constituted of 2 floats that specifies the relative position of the legend according the upper left corner of the bounding box.
  The default value is set to $(1.01, 1.01)$.
- `tuple_inches_figsize` is a Python tuple constituted of 2 floats that are respectively the width and the height of the output PNG file in inches (by default, an inch corresponds to 80 pixels).
  The default value is set to $(12, 12)$.
- `outputdir` is the output directory where the produced PNG files will be stored. Each PNG file will be given the name `nameh5_aliasfield.png` where `nameh5` is the name of an element of `inputh5set` without its `.h5` extension.
  The default directory is the current directory.

– `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`.
The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

```
plot_pseudocolor_set(inputh5set, field, \
                     aliasfield='', \
                     list_minmax=[], \
                     list_bbox=['default']*4, \
                     tuple_leg_position=(0.8,0.45,0.2,0.3), \
                     tuple_inches_figsize=(12., 12.), \
                     outputdir='.', \
                     forcesave=False)
```

Mandatory arguments

– `inputh5set` is a list that contains the name of all HDF5 files to be studied.

– `field` (string) is the field within the HDF5 file to be plotted. Such field must be written one of the scalar diagnostics in Tables 4-5.

Optional arguments

– `aliasfield` (string) is the field name as it will be used as plot title and legend. Leaving the default value (empty string) means that this alias will be exactly `field`.

– `list_minmax` is a list of 2 floats that will be the bounds of the colormap. Leaving the default value (empty list) means that `ad hoc` colormap bounds will be used.

– `list_bbox` is a list of 4 elements that can be either floats or `'default'`. These 4 elements stands for $x_{\min,bb}$, $x_{\max,bb}$, $y_{\min,bb}$ and $y_{\max,bb}$ respectively which will be used for the visualization bounding box $[x_{\min,bb}, x_{\max,bb}] \times [y_{\min,bb}, y_{\max,bb}]$. Any `'default'` value means that the corresponding part of the bounding box will be identified thanks to the used nodes.

– `tuple_legend_position` is a Python tuple constituted of 4 floats that specifies the relative position and size of the legend according to the upper left corner of the bounding box with the format $(x, y, width, height)$ in units that are relative to the bounding box.
The default value is set to $(0.8, 0.45, 0.2, 0.3)$.

– `tuple_inches_figsize` is a Python tuple constituted of 2 floats that are respectively the width and the height of the output PNG file in inches (by default, an inch corresponds to 80 pixels).
The default value is set to $(12, 12)$.

- `outputdir` is the output directory where the produced PNG files will be stored. Each PNG file will be given the name `nameh5_aliasfield.png` where `nameh5` is the name of an element of `inputh5set` without its `.h5` extension.
  The default directory is the current directory.
- `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`.
  The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

```
plot_vectorfield_set(inputh5set, field_x, field_y, \
                     aliasfield='', \
                     narrows=200, \
                     scale=2., \
                     width=0.01, \
                     list_bbox=['default']*4, \
                     tuple_leg_position=(0.8,0.45,0.2,0.3), \
                     tuple_inches_figsize=(12.,12.), \
                     outputdir='.', \
                     forcesave=False)
```

Mandatory arguments

- `inputh5set` is a list that contains the name of all HDF5 files to be studied. item `field_x` and `field_y` (strings) are the scalar fields within the HDF5 file to be used as the components of the vector field to be plotted. Such fields must be some of the scalar diagnostics in Tables 4-5 and they must be discretized on the same grid.

Optional arguments

- `aliasfield` (string) is the field name as it will be used as plot title and legend. Leaving the default value (empty string) means that this alias will be exactly `(field_x,field_y)`.
- `narrows` (integer) gives an indication of the number of arrows to be plotted. These arrows will be uniformly distributed on the mesh and colored by their length.
  Default value is set to 200.
- `scale` (float) is the number of data unit per arrow length unit. Taking a value smaller than 1 makes the arrows longer.
  Default value is set to 2.
- `width` (float) is the shaft width in arrow units. One can start with the value equal to 0.005 times the length of the domain bounding box diagonal.
  Default value is set to 0.01.

- `list_bbox` is a list of 4 elements that can be either floats or `'default'`. These 4 elements stands for $x_{\min,bb}$, $x_{\max,bb}$, $y_{\min,bb}$ and $y_{\max,bb}$ respectively which will be used for the visualization bounding box $[x_{\min,bb}, x_{\max,bb}] \times [y_{\min,bb}, y_{\max,bb}]$. Any `'default'` value means that the corresponding part of the bounding box will be identified thanks to the used nodes.
- `tuple_legend_position` is a Python tuple constituted of 4 floats that specifies the relative position and size of the legend according to the upper left corner of the bounding box with the format $(x, y, width, height)$ in units that are relative to the bounding box.
  The default value is set to $(0.8, 0.45, 0.2, 0.3)$.
- `tuple_inches_figsize` is a Python tuple constituted of 2 floats that are respectively the width and the height of the output PNG file in inches (by default, an inch corresponds to 80 pixels).
  The default value is set to $(12, 12)$.
- `outputdir` is the output directory where the produced PNG files will be stored. Each PNG file will be given the name `nameh5_aliasfield.png` where `nameh5` is the name of an element of `inputh5set` without its `.h5` extension.
  The default directory is the current directory.
- `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`.
  The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

In addition of all these functions, one can use the following function to plot a triangulation within a HDF5 mesh file:

```
plot_mesh(meshh5, method, \
          aliasfield='', \
          list_bbox=['default']*4, \
          tuple_inches_figsize=(12.,12.), \
          outputpng='./temp.png', \
          forcesave=False)
```

Mandatory arguments

- `meshh5` (string) is the input HDF5 mesh file to be studied.
- `method` (string) is the type of Finite Element mesh to be plotted. At present time, it must be one of the values listed in Table 4.

Optional arguments

- `aliasfield` (string) is the field name as it will be used as plot title and legend. Leaving the default value (empty string) means that this alias will be exactly `field`.

- `list_bbox` is a list of 4 elements that can be either floats or `'default'`. These 4 elements stands for $x_{\min,bb}$, $x_{\max,bb}$, $y_{\min,bb}$ and $y_{\max,bb}$ respectively which will be used for the visualization bounding box $[x_{\min,bb}, x_{\max,bb}] \times [y_{\min,bb}, y_{\max,bb}]$.
  Any `'default'` value means that the corresponding part of the bounding box will be identified thanks to the used nodes.

- `tuple_inches_figsize` is a Python tuple constituted of 2 floats that are respectively the width and the height of the output PNG file in inches (by default, an inch corresponds to 80 pixels).
  The default value is set to $(12, 12)$.

- `outputpng` (string) is the name of the output PNG file.
  The default output name is `temp.png` and is located in the current directory.

- `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`.
  The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

Finally, a function is dedicated to the comparison of the isolines of a specific 2D scalar diagnostic from 2 distinct files. This can be done with the following function:

```
plot_two_contour(inputh5, field, list_alias_diff, \
                 aliasfield='', \
                 levels=[], \
                 list_bbox=['default']*4, \
                 tuple_anchor_legend=(1.01,1.01), \
                 tuple_inches_figsize=(12.,12.), \
                 outputpng='./temp.png', \
                 forcesave=False)
```

Mandatory arguments

- `inputh5` (string) is a list of 2 HDF5 file names. The data from to `inputh5[0]` will be plotted with solid lines and the data from to `inputh5[1]` will be plotted with dashed lines.

- `field` (string) is the field within both HDF5 files to be plotted. Such field must be written one of the scalar diagnostics in Tables 4-5.

- `list_alias_diff` is a list of 2 strings that will be used as legends. Taking 2 different strings is recommended.

Optional arguments

- `aliasfield` (string) is the field name as it will be used as plot title and legend.
  Leaving the default value (empty string) means that this alias will be exactly `field`.

- `levels` (list of floats) is the list of isovalues to be plotted.
  The default value (an empty list) means that a set of 10 equally distributed isovalues will be identified accordingly with the studied data.

- `list_bbox` is a list of 4 elements that can be either floats or `'default'`. These 4 elements stands for $x_{\min,bb}$, $x_{\max,bb}$, $y_{\min,bb}$ and $y_{\max,bb}$ respectively which will be used for the visualization bounding box $[x_{\min,bb}, x_{\max,bb}] \times [y_{\min,bb}, y_{\max,bb}]$.
  Any `'default'` value means that the corresponding part of the bounding box will be identified thanks to the used nodes.

- `tuple_anchor_legend` is a Python tuple constituted of 2 floats that specifies the relative position of the legend according the upper left corner of the bounding box.
  The default value is set to $(1.01, 1.01)$.

- `tuple_inches_figsize` is a Python tuple constituted of 2 floats that are respectively the width and the height of the output PNG file in inches (by default, an inch corresponds to 80 pixels).
  The default value is set to $(12, 12)$.

- `outputpng` (string) is the name of the output PNG file.
  The default output name is `temp.png` and is located in the current directory.

- `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`.
  The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

An application example can be found in the file `EXAMPLES/plot_two_scalar_step.py`:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

############################ WARNING ############################
# For running this script with Linux or MacOS, open a terminal in
# the subdirectory EXAMPLES, then type the command
#
#    python plot_two_scalar_step.py
#
#################################################################

import sys
sys.path.insert(0, '../PYTHON')

# Import the module for plotting the contents of HDF5 files
from plots import *

# We aim to compare the contents of the following file:
```

```
inputh5 = ['../RESULTS/EXAMPLES/step_neumann/diags_10.h5', \
           '../RESULTS/EXAMPLES/step_natural/diags_10.h5']

# ... and more precisely in terms of x-component of the velocity
field = 'Velocity_x'

# The difference between the input file lies in the use of
# different boundary conditions
list_alias_diff = ['Neumann BC', 'Natural BC']


# Simple comparating contour plot
plot_two_contour(inputh5, field, list_alias_diff, \
  aliasfield='ux', outputpng='./ux_compare.png', forcesave=True)

# Same with a prescribed bounding box
plot_two_contour(inputh5, field, list_alias_diff, \
                 aliasfield='ux', list_bbox=[0., 30., 0., 3], \
                 outputpng='./ux_compare_bbox.png', forcesave=True)

# Same with prescribed isovalues
plot_two_contour(inputh5, field, list_alias_diff, \
                 aliasfield='ux', levels=[0., 0.5, 1., 1.5], \
                 outputpng='./ux_compare_levels.png', forcesave=True)
```

8.4 Visualize and save an animated result

It is possible to generate movie representing the time evolution of a simulation. Several solutions have been developed to reach this goal.

8.4.1 Matlab solution

NS2DDV embeds a Matlab solution for visualizing an animated result without saving it. This job can be done with the routine `movie_from_file` that can handle `.mat` or `.h5` file sets.

This routine almost works in the same way as `plot_from_file` (see paragraph 8.3.1). Indeed, the user must move to the root path of NS2DDV, then type the following commands:

```
>> load_paths();

>> movie_from_file(inputs, XDISPLAY_LIST, LEVELS, XDISPLAY_METHOD, ...
typeplot, nbrows, nbcols);
```

The arguments of the routine `movie_from_file` are the following:

- `inputs` is a cell array that can be of size 1, 3 or 4:

2D diagnostic	Visualization in-situ	Visualization from file with	
		Matlab	Python-Matplotlib
$\mathbb{P}_1$ mesh	'MESH'		'P1'
$\mathbb{P}_2$ or $\mathbb{P}_{1,b}$ subtriangulation (according to disc. of $\mathbf{u}$)	'SUBTRI_U'		'P2' or 'P1B'
Density ρ (scalar)	'RHO'		'Density'
Velocity $\mathbf{u}$ (vector)	'U_VECTOR'		'Velocity_x' 'Velocity_y'
x-part of velocity u_x (scalar)	'UX'		'Velocity_x'
y-part of velocity u_y (scalar)	'UY'		'Velocity_y'
$\partial_x u_x$ (scalar)	'DX_UX'		'Velocity_x_dx'
$\partial_y u_x$ (scalar)	'DY_UX'		'Velocity_x_dy'
$\partial_x u_y$ (scalar)	'DX_UY'		'Velocity_y_dx'
$\partial_y u_y$ (scalar)	'DY_UY'		'Velocity_y_dy'
Shear rate $\partial_y u_x + \partial_x u_y$ (scalar)	'SHEAR'		'Shear_rate'
Velocity magnitude $\lvert\mathbf{u}\rvert^2$ (scalar)	'U_MODULE'		'Velocity_magnitude'
Vorticity ω (scalar)	'VORTIC'		'Vorticity'
Velocity streamlines ψ (scalar)	'STREAM'		'Streamlines'
Pressure p (scalar)	'P'		'Pressure'
Pressure gradient ∇p (vector)	'GRAD_P'		Pressure_gradient_x Pressure_gradient_y
$\partial_x p$ (scalar)	'P_DX'		Pressure_gradient_x
$\partial_y p$ (scalar)	'P_DY'		Pressure_gradient_y
Auxiliary velocity $\mathbf{w}$ (vector)	'W_VECTOR'		'wx' 'wy'
x-part of auxiliary velocity w_x (scalar)	'WX'		'wx'
y-part of auxiliary velocity w_y (scalar)	'WY'		'wy'
Auxiliary pressure r (scalar)	'R'		'r'

Table 4: List of 2D diagnostics to be called for visualization

2D diagnostic	Visualization in-situ	Visualization from file with	
		Matlab	Python-Matplotlib
Exact density ρ_{ex} (scalar)	'RHO_EX'		'Density_exact'
Exact velocity $\mathbf{u}_{ex}$ (vector)	'U_VECTOR_EX'		'Velocity_x_exact' 'Velocity_y_exact'
x-part of exact velocity $u_{x,ex}$ (scalar)	'UX_EX'		'Velocity_x_exact'
y-part of exact velocity $u_{y,ex}$ (scalar)	'UY_EX'		'Velocity_y_exact'
$\partial_x u_{x,ex}$ (scalar)	'DX_UX_EX'		'Velocity_x_dx_exact'
$\partial_y u_{x,ex}$ (scalar)	'DY_UX_EX'		'Velocity_x_dy_exact'
$\partial_x u_{y,ex}$ (scalar)	'DX_UY_EX'		'Velocity_y_dx_exact'
$\partial_y u_{y,ex}$ (scalar)	'DY_UY_EX'		'Velocity_y_dy_exact'
Exact shear rate $\partial_y u_{x,ex} + \partial_x u_{y,ex}$ (scalar)	'SHEAR_EX'		'Shear_rate_exact'
Exact velocity magnitude $\lvert\mathbf{u}_{ex}\rvert^2$ (scalar)	'U_MODULE_EX'		'Velocity_magnitude_exact'
Exact vorticity ω_{ex} (scalar)	'VORTIC_EX'		'Vorticity_exact'
Exact velocity streamlines ψ (scalar)	'STREAM_EX'		'Streamlines_exact'
Exact pressure p_{ex} (scalar)	'P_EX'		'Pressure_exact'
Exact pressure gradient ∇p_{ex} (vector)	'GRAD_P_EX'		Pressure_gradient_x_exact Pressure_gradient_y_exact
$\partial_x p_{ex}$ (scalar)	'P_DX_EX'		Pressure_gradient_x_exact
$\partial_y p_{ex}$ (scalar)	'P_DY_EX'		Pressure_gradient_y_exact

Table 5: List of specific 2D diagnostics to be called for visualization for manufactured test cases (EXAC, EXACNEU)

- Size ≥ 1: the first component is a string pattern like `prefix_*.h5`. In such case, the code will look for all the files that match with such pattern.
- Size ≥ 3: the second component is a starting index i_{start} and the third one is an ending index i_{end}. In such case, the code will look for all the files with name `prefix_`i_{start}`.h5`, ... , `prefix_`i_{end}`.h5`.
- Size = 4: the fourth argument is a step i_{step}. In such case, the code will look for all the files with name `prefix_`i_{start}`.h5`, `prefix_`$i_{start} + i_{step}$`.h5`, ... , `prefix_`i_{end}`.h5`.

- `XDISPLAY_LIST` is a cell line where each cell is a diagnostic to be plotted (see Tables 4-5). Here is a example of such list:

```
>> XDISPLAY_LIST = {'MESH', 'P', 'UX', 'UY', 'U_VECTOR'};
```

- `LEVELS` is a cell line that contains the isovalues for each scalar diagnostic. This cell line must have the same size as `XDISPLAY_LIST`. If the user wants to fix the isovalues, (s)he must provide an line vector containing the required isovalues. In the other cases, the user must provide an empty vector. Here is a example of such list associated to the example of `XDISPLAY_LIST` above:

```
>> LEVELS = {[], [0., 1., 2.], [], [], []};
```

  Here, we fix the isovalues for the pressure and leave them to the computer for the other diagnostics.
  Note that, if `XDISPLAY_LIST{i}` is a vector field, the value of `LEVELS{i}` is not taken into account (see the example below for further details).

- `XDISPLAY_METHOD` (string) must be either `'SINGLE_FIGURE'` or `'MULTIPLE_FIGURE'`. The `'SINGLE_FIGURE'` option produces a single Matlab figure window in which the results are plotted in subplot regions, and the `'MULTIPLE_FIGURE'` option produces a Matlab figure window per 2D result that is asked for plotting.

- `typeplot` (string) must be either `'contour'` or `'pseudocolor'`. It will specify if the plot mode for scalar diagnostics.

  Remark 8.9. At present time, the diagnostics `'P_DX'`, `'P_DY'`, `'P_DX_EX'` and `'P_DY_EX'` are automatically plotted in `'pseudocolor'` mode. We are currently looking for a `'contour'` solution for these diagnostics.

- `nbrows` and `nbcols` (integers) are respectively the number of rows and columns that define the mosaic of graphs in the case where the `'SINGLE_FIGURE'` display method is chosen. They are mandatory if this display method is chosen, and optional if `'MULTIPLE_FIGURE'` is chosen instead.

Example 8.10. This example is based on the Matlab script `EXAMPLES/movie_nona.m`: to run it, the user must start Matlab in the root directory of NS2DDV, then run the following command:

```
>> run('./EXAMPLES/movie_nona.m')
```

The idea is to study the time dynamics in the numerical results from the "ready-to-use" example `test_nona.m` (see Section 3.4.5). More precisely, we want to plot the $\mathbb{P}_1$ mesh, and the dynamics of the vector field $\mathbf{u}$, the pressure p and its first order derivatives. In addition, we want to focus on the isovalues $p = -1, 0, \ldots, 3$. This is why we can find the following lines within the script `EXAMPLES/visu_nona.m`:

```
load_paths()

XDISPLAY_LIST = {'MESH', 'U_VECTOR', 'P', 'P_DX', 'P_DY'};

LEVELS = {[], [1], [-1., 0., 1., 2., 3.], [], []};

inputs = {'./RESULTS/EXAMPLES/test_nona/diags_*.h5'};
% Equivalent definitions
% inputs = {'./RESULTS/EXAMPLES/test_nona/diags_*.h5', 0, 12};
% inputs = {'./RESULTS/EXAMPLES/test_nona/diags_*.h5', 0, 12, 1};

movie_from_file(inputs, XDISPLAY_LIST, LEVELS, 'SINGLE_FIGURE', ...
   'contour', 3, 2);
```

8.4.2 Python-FFmpeg solution (no-display)

Assuming that a set of frames under PNG format have been generated (see Section 8.3), it is possible to convert these files into a MPEG movie. To do this, the software FFmeg and a Python distribution are required. See below for complementary details:

- FFmpeg website: `https://www.ffmpeg.org/`

The advantage of using FFmpeg to convert a set of PNG files into a video is that it can be run in "no-display" mode and, consequently, is compatible with batch tools such as Slurm, Torque...

NS2DDV has been supplemented with some Python routines for using FFmpeg in a more "user-friendly" way. These routines are gathered in the file `make_movie.py`. The following script gives an idea of using it (see also the Python file `EXAMPLES/movie_nona.py`:

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-

############################ WARNING ############################
# For running this script with Linux or MacOS, open a terminal in
# the subdirectory EXAMPLES, then type the command
#
#    python movie_nona.py
#
#################################################################

import sys
sys.path.insert(0, '../PYTHON')

# Import the module for plotting the contents of HDF5 files
from plots import *
# Import the module for encoding movies
from make_movie import *



# The input files are RESULTS/EXAMPLES/test_nona/diags_i.h5 with
# i=0,...,12
dir_inputh5 = '../RESULTS/EXAMPLES/test_nona'
istart = 0
iend = 12

# We aim to encode the dynamics of
# - ux with contours
# - uy with contours
# - p with pseudocolor
# - u with vectors


# Warning: the size of output PNGs should be prescribed
figsize = (19.2,12.)

# Build the list of input files
inputh5set = []
for i in range(istart, iend+1):
        inputh5set.append(dir_inputh5+'/diags_'+str(i)+'.h5')


# Plot the contour of ux
# (Default: 10 isovalues that do not change with time)
plot_contour_set(inputh5set, 'Velocity_x', aliasfield='ux', \
                 tuple_inches_figsize=figsize, \
                 outputdir='./tmp', forcesave=True)

# Plot the contour of uy
# (Default: 10 isovalues that do not change with time)
plot_contour_set(inputh5set, 'Velocity_y', aliasfield='uy', \
                 tuple_inches_figsize=figsize, \
```

```
                  outputdir='./tmp', forcesave=True)

# Plot the pseudocolor of p
# (Default: bounds of colormap do not change with time)
plot_contour_set(inputh5set, 'Pressure', aliasfield='p', \
                  tuple_inches_figsize=figsize, \
                  outputdir='./tmp', forcesave=True)

# Plot the vector field u (Default: 200 arrows)
plot_vectorfield_set(inputh5set, 'Velocity_x', 'Velocity_y', \
                     aliasfield='u', tuple_inches_figsize=figsize, \
                     outputdir='./tmp', forcesave=True)

# For each set {diags_i_ux.png, diags_i_uy.png, diags_i_p.png,
# diags_i_u.png}, build a 2x2 mosaic named diags_i.png and
# prepare a list of PNGs for video encoding
# Warning: i = 000, 001, 002, ... 012
list_encoding = []
ndigits = int(numpy.ceil(numpy.log10(iend)+1))
for i in range(istart, iend+1):
    inputpngs = ['./tmp/diags_'+str(i).rjust(ndigits, '0')+'_ux.png', \
                 './tmp/diags_'+str(i).rjust(ndigits, '0')+'_uy.png', \
                 './tmp/diags_'+str(i).rjust(ndigits, '0')+'_p.png', \
                 './tmp/diags_'+str(i).rjust(ndigits, '0')+'_u.png']
    outputpng = './tmp/diags_'+str(i).rjust(ndigits, '0')+'.png'
    build_mosaic(inputpngs, 2, 2, outputpng)
    list_encoding.append(outputpng)

# Encode the video
encode_video(list_encoding, './tmp', 'nona.mp4', forcesave=True)

# Remove the directory ./tmp and its contents
shutil.rmtree('./tmp')
print('Directory ./tmp deleted')
```

The most important one is dedicated to the conversion PNG→MP4:

```
encode_video(inputpngs, tmppath, outputmovie, forcesave=False)
```

Mandatory arguments

- `inputpngs` is an ordered list of PNG file names.
- `tmppath` (string) is the path to a temporary directory that will be used for the conversion. Note that if this directory already exists, it should not contain any PNG file. If it does not exist, it will be created at the beginning of the job and deleted at the end of the job.
- `outputmovie` (string) is the name of the output movie file. The extension of such file should be `.mp4`.

Optional arguments

- `forcesave` is a boolean that indicates if save forcing is authorized or not in the case where there already exists a PNG file with a name that matches with `outputpng`. The default value is `False`, meaning that if the output PNG already exists, the program stops with an error message.

If ImageMagick is installed, it is possible to use the command `montage` to build a mosaic of graphs from a set of PNG files. For this purpose, one can use the following routine that is a part of `make_movie.py`:

```
build_mosaic(inputpngs, nrows, ncols, outputpng)
```

- `inputpngs` is an ordered list of PNG file names.
- `nrows` and `ncols` (integers) are the number of rows and columns in the expected mosaic. Note that the product `nrows`×`ncols` must larger or equal to the size of the list `inputpngs`.
- `outputpng` (string) is the name of the output PNG file.

More informations about ImageMagick are available at the following link:

- ImageMagick website: `http://imagemagick.org/script/index.php`
- `montage` command documentation: `http://www.imagemagick.org/Usage/montage/`

8.5 Convergence and stability analysis

Several test cases are based on analytical solutions and *ad hoc* manufactured source terms (see test cases `EXAC` in paragraphes 5.2.3 and 5.1.3, and `EXACNEU` in paragraph 5.2.4 and 5.1.4). If one these test cases in considered during the execution of the routine `generate_setup_file.m`, it is possible to run a convergence analysis with mutiple time and/or space discretizations instead of a single time-space discretization. Such framework is also adapted for testing a single time-space discretization against multiple values of the Reynolds number Re. More precisely, assuming that the user chose the test case `EXAC` or `EXACNEU` during the execution of `generate_setup_file`, (s)he will be asked the following question:

```
Do you want to run a convergence or stability analysis ?
'IN_SPACE'      Convergence analysis according to the space step only
'IN_TIME'       Convergence analysis according to the time step only
'IN_SPACETIME'  Convergence analysis according to both space and time
                steps
'IN_REYNOLDS'   Stability analysis according to the Reynolds number
'NONE'          No stability/convergence analysis, run a simple
                simulation
Your choice :
```

According to the answer, the generated setup file is configured in a slightly different way and `PARAMETERS.CV_STUDY` is given a value.

8.5.1 Convergence with space mesh refinement

In this paragraph, it is assumed that the user chose the answer `'IN_SPACE'` to the question above. Hence, (s)he will find in the non-modifiable part of the setup file the following line:

```
PARAMETERS.CV_STUDY = 'IN_SPACE';
```

The goal is to study the error between the exact solution and the approximation computed by NS2DDV when the space mesh is refined and the time mesh is common to all runs. More precisely, assuming that the space meshes $\mathfrak{M}_1, \ldots, \mathfrak{M}_K$ are considered, and denoting with $h_{k,\min}$ and $h_{k,\max}$ the minimal and maximal edge length within the mesh $\mathfrak{M}_k$, the time step bound that is used for all runs is the following:

$$\Delta t = \min_{k=1,\ldots,K} \left(C_m h_{k,\min}^{\alpha_m} + C_M h_{k,\max}^{\alpha_M} \right) ,$$

where $C_m, C_M, \alpha_m, \alpha_M$ can be specified by the user in the setup file as follows:

```
% Value of alphamax
PARAMETERS.FE.ALPHAMAX_STEP_TIME = αM;
% Value of alphamin
PARAMETERS.FE.ALPHAMIN_STEP_TIME = αm;
% Value of Cmax
PARAMETERS.FE.CMAX_STEP_TIME = CM;
% Value of Cmin
PARAMETERS.FE.CMIN_STEP_TIME = Cm;
```

Note that, as it is explained at the beginning of Section 6, such Δt defines a time mesh $(t^0, \ldots, t^N)$ that is all common to all simulations.

By default, $\alpha_m = \alpha_M = 1.5$ for any test case associated to the NS model or to the NSDV model with Strang splitting (see paragraph 6.1.2), and $\alpha_m = \alpha_M = 1$ is set by default for any test case associated to the NSDV model with Lax splitting (see paragraph 6.1.1). The default values of C_m and C_M are respectively 1 and 0.

For characterizing the space mesh family $(\mathfrak{M}_1, \ldots, \mathfrak{M}_K)$, the user must proceed as follows:

- `'NS2DDV'` mesh generation method (see paragraph 4.2):
  - `'RECTANGLE'` geometry: each mesh $\mathfrak{M}_k$ is characterized by the numbers of edges in x and y directions denoted with $n_{e,x,k}$ and $n_{e,y,k}$ respectively. These integer values can be input in the setup file as follows:

```
% Number of edges on North and South bounds
% WARNING: it must be a multiple of 2
% {value 1, value 2, ...} set of values for NBSEG_X for running
% the convergence analysis
PARAMETERS.MESH.NBSEG_X = {n_{e,x,1}, ..., n_{e,x,K}};
% Number of edges on West and East bounds
% WARNING: it must be a multiple of 2
% {value 1, value 2, ...} set of values for NBSEG_Y for running
% the convergence analysis
PARAMETERS.MESH.NBSEG_Y = {n_{e,y,1}, ..., n_{e,y,K}};
```

– 'DIHEDRON' geometry: each mesh $\mathfrak{M}_k$ is characterized by its number $n_{n,y,BL,k}$ of nodes in y-direction in the boundary layer, its number $n_{e,y,OBL,k}$ of edges in y-direction outside of the boundary layer, and its number $n_{e,x,BU,k}$ of edges by unit in x-direction. All meshes share the same height of boundary layer h_{BL} and the same geometric reason α that describes the repartition of nodes in y-direction in the boundary layer. Concretely, the user should modify the setup file as follows for characterizing the mesh family $(\mathfrak{M}_1, \ldots, \mathfrak{M}_K)$:

```
% Height of the boundary layer
PARAMETERS.MESH.HEIGHT_BL = h_{BL};
% Number of edges by segment of length 1 in x-direction
% (even values recommended)
PARAMETERS.MESH.NBSEG_PERUNIT_X = {n_{e,x,BU,1}, ..., n_{e,x,BU,K}};
% Number of nodesspace in the boundary layer in y-direction
PARAMETERS.MESH.NBNODES_INBL_Y = {n_{n,y,BL,1}, ..., n_{n,y,BL,K}};
% Geometric reason of nodes repartition in the boundary layer
% in y-direction
PARAMETERS.MESH.PROG_GEOM_BL_Y = α;
% Number of edges out of the boundary layer in y-direction
PARAMETERS.MESH.NBSEG_OUTBL_Y = {n_{e,y,OBL,1}, ..., n_{e,y,OBL,K}};
```

- 'PDET' mesh generation method (see paragraph 4.2): each space mesh $\mathfrak{M}_k$ is generated by Matlab Partial Differential Equation Toolbox thanks to the characteristic edge length $h_{0,k}$. Hence, to characterize the full mesh list $(\mathfrak{M}_1, \ldots, \mathfrak{M}_K)$, the user should modify the setup file as follows:

```
% Characteristic edge length in the mesh
% {value 1, value 2, ...} set of values for H0 for running
% the convergence analysis
PARAMETERS.MESH.H0 = {h_{0,1}, ..., h_{0,K}};
```

- 'FROM_FILE' mesh generation method (see paragraph 4.2): this mesh generation solution is not compatible with any convergence or stability analysis approach at present time.

By default, the dynamics of each simulation is not saved as `.mat`/`.h5` files but only

the log file. More precisely, if we have the following lines in the setup file

```
PARAMETERS.MODEL = 'themodel';
PARAMETERS.TESTCASE = 'thetestcase';
PARAMETERS.OUTPUT.DIRECTORY_NAME = 'thedirectory';
```

the simulation associated with the space mesh $\mathfrak{M}_k$ will produce a log file in the sub-directory `thedirectory/themodel_thetestcase_`k.

Once all simulations are finished, NS2DDV analyses the log files that have been produced by each of these simulations and focuses on some specific diagnostics including the following errors:

- Error on the velocity:

$$e_{\mathbf{u},\Delta t,\mathfrak{M}_k} = \max_{n=0,\ldots,N} \|\mathbf{u}_{\Delta t,\mathfrak{M}_k}(t^n,\cdot) - \mathbf{u}_{ex}(t^n,\cdot)\|_{(L^2(\Omega))^2} \ ,$$

- Error on the pressure:

$$e_{p,\Delta t,\mathfrak{M}_k} = \max_{n=0,\ldots,N} \|p_{\Delta t,\mathfrak{M}_k}(t^n,\cdot) - p_{ex}(t^n,\cdot)\|_{L^2(\Omega)} \ ,$$

- Error on the density (only for `NSDV` model):

$$e_{\rho,\Delta t,\mathfrak{M}_k} = \max_{n=0,\ldots,N} \|\rho_{\Delta t,\mathfrak{M}_k}(t^n,\cdot) - \rho_{ex}(t^n,\cdot)\|_{L^2(\Omega)} \ .$$

The very last part of the convergence analysis consists in plotting and saving the errors curves $h_{k,\max} \mapsto e_{\mathbf{u},\Delta t,\mathfrak{M}_k}$, $h_{k,\max} \mapsto e_{p,\Delta t,\mathfrak{M}_k}$ and $h_{k,\max} \mapsto e_{\rho,\Delta t,\mathfrak{M}_k}$ in EPS file format for summarizing the convergence of the involved numerical methods as the space mesh is refined.

8.5.2 Convergence with time mesh refinement

In this paragraph, it is assumed that the user chose the answer `'IN_TIME'` to the question in the introduction of Section 8.5. Hence, (s)he will find in the non-modifiable part of the setup file the following line:

```
PARAMETERS.CV_STUDY = 'IN_TIME';
```

The goal is to study the error between the exact solution and the approximation computed by NS2DDV when the time mesh is refined and the space mesh is common to all runs. More precisely, considering a mesh $\mathfrak{M}$ where the maximal length of an edge is denoted with $h_{\max}$, the goal is to run NS2DDV with the space mesh $\mathfrak{M}$ and various time steps Δt_k $(k = 1,\ldots,K)$, where

$$\Delta t_k = C_k\, h_{\max}^{\alpha} \ ,$$

where α is common to all simulations and $C_1, \ldots, C_K$ can be set by the user in the setup file as follows:

```
% Time step for solving Stokes equation
% WARNING : this time step is of the form C*hmax^alpha
% where hmax is the max length of an edge for the considered space mesh
% Value of alpha
PARAMETERS.FE.ALPHA_STEP_TIME = α;
% Values of C (provide several values for running a convergence
% analysis in dt)
PARAMETERS.FE.C_STEP_TIME = {C_1, ..., C_K};
```

Hence, each time step bound Δt_k is associated to a time mesh $(t_k^0, \ldots, t_k^{N_k})$ as it is explained in the first lines of Section 6.

As for convergence in space mesh refinement, the dynamics of each simulation is not saved as `.mat`/`.h5` files but only the log file in the present casen and, assuming that the following lines are in the setup file

```
PARAMETERS.MODEL = 'themodel';
PARAMETERS.TESTCASE = 'thetestcase';
PARAMETERS.OUTPUT.DIRECTORY_NAME = 'thedirectory';
```

the simulation associated with the time step Δt_k will produce a log file in the subdirectory `thedirectory/themodel_thetestcase_`k.

Once all simulations are finished, NS2DDV analyses the log files that have been produced by each of these simulations and focuses on some specific diagnostics including the following errors:

- Error on the velocity:

$$e_{\mathbf{u},\Delta t_k,\mathfrak{M}} = \max_{n_k=0,\ldots,N_k} \|\mathbf{u}_{\Delta t_k,\mathfrak{M}}(t_k^{n_k},\cdot) - \mathbf{u}_{ex}(t_k^{n_k},\cdot)\|_{(L^2(\Omega))^2} \ ,$$

- Error on the pressure:

$$e_{p,\Delta t_k,\mathfrak{M}} = \max_{n_k=0,\ldots,N_k} \|p_{\Delta t_k,\mathfrak{M}}(t_k^{n_k},\cdot) - p_{ex}(t_k^{n_k},\cdot)\|_{L^2(\Omega)} \ ,$$

- Error on the density (only for `NSDV` model):

$$e_{\rho,\Delta t_k,\mathfrak{M}} = \max_{n_k=0,\ldots,N_k} \|\rho_{\Delta t_k,\mathfrak{M}}(t_k^{n_k},\cdot) - \rho_{ex}(t_k^{n_k},\cdot)\|_{(L^2(\Omega))^2} \ .$$

The very last part of the convergence analysis consists in plotting and saving the errors curves $\Delta t_k \mapsto e_{\mathbf{u},\Delta t_k,\mathfrak{M}}$, $\Delta t_k \mapsto e_{p,\Delta t_k,\mathfrak{M}}$ and $\Delta t_k \mapsto e_{\rho,\Delta t_k,\mathfrak{M}}$ in EPS file format for summarizing the convergence of the involved numerical methods as the time mesh is refined.

8.5.3 Convergence with space-time mesh refinement

In this paragraph, it is assumed that the user chose the answer `'IN_SPACETIME'` to the question in the introduction of Section 8.5. Hence, (s)he will find in the non-modifiable part of the setup file the following line:

```
PARAMETERS.CV_STUDY = 'IN_SPACETIME';
```

The goal is to study the error between the exact solution and the approximation computed by NS2DDV when both space and time meshes are refined. More precisely, we consider a space mesh list $(\mathfrak{M}_1, \ldots, \mathfrak{M}_K)$ and a time step bound list $(\Delta t_1, \ldots, \Delta t_K)$ with

$$\Delta t_k = C_m h_{k,\min}^{\alpha_m} + C_M h_{k,\max}^{\alpha_M},$$

where $h_{k,\min}$ and $h_{k,\max}$ are respectively the minimal and maximal edge length in the space mesh $\mathfrak{M}_k$, and $C_m, C_M, \alpha_m, \alpha_M$ can be specified by the user in the setup file as follows:

```
% Value of alphamax
PARAMETERS.FE.ALPHAMAX_STEP_TIME = α_M;
% Value of alphamin
PARAMETERS.FE.ALPHAMIN_STEP_TIME = α_m;
% Value of Cmax
PARAMETERS.FE.CMAX_STEP_TIME = C_M;
% Value of Cmin
PARAMETERS.FE.CMIN_STEP_TIME = C_m;
```

Consequently, the user just has to characterize the space meshes $\mathfrak{M}_1, \ldots, \mathfrak{M}_K$ to complete the definitions of the time step bounds $\Delta t_1, \ldots, \Delta t_K$ and, implicitely, the definition of the time meshes $(t_k^0, \ldots, t_k^{N_k})$ for each $k = 1, \ldots, K$. For this purpose, we refer to the paragraph 8.5.1 dedicated to the convergence in space for a full definition of the space mesh list. Note that the default values for $C_m, C_M, \alpha_m, \alpha_M$ are the same as in paragraph 8.5.1.

As for convergence in space mesh refinement or time mesh refinement, the dynamics of each simulation is not saved as `.mat`/`.h5` files but only the log file in the present casen and, assuming that the following lines are in the setup file

```
PARAMETERS.MODEL = 'themodel';
PARAMETERS.TESTCASE = 'thetestcase';
PARAMETERS.OUTPUT.DIRECTORY_NAME = 'thedirectory';
```

the simulation associated with the couple $(\Delta t_k, \mathfrak{M}_k)$ will produce a log file in the subdirectory `thedirectory/themodel_thetestcase_`k.

Once all simulations are finished, NS2DDV analyses the log files that have been produced by each of these simulations and focuses on some specific diagnostics including the following errors:

- Error on the velocity:

$$e_{\mathbf{u},\Delta t_k,\mathfrak{M}_k} = \max_{n_k=0,\ldots,N_k} \|\mathbf{u}_{\Delta t_k,\mathfrak{M}_k}(t_k^{n_k},\cdot) - \mathbf{u}_{ex}(t_k^{n_k},\cdot)\|_{(L^2(\Omega))^2} \ ,$$

- Error on the pressure:

$$e_{p,\Delta t_k,\mathfrak{M}_k} = \max_{n_k=0,\ldots,N_k} \|p_{\Delta t_k,\mathfrak{M}_k}(t_k^{n_k},\cdot) - p_{ex}(t_k^{n_k},\cdot)\|_{L^2(\Omega)} \ ,$$

- Error on the density (only for `NSDV` model):

$$e_{\rho,\Delta t_k,\mathfrak{M}_k} = \max_{n_k=0,\ldots,N_k} \|\rho_{\Delta t_k,\mathfrak{M}_k}(t_k^{n_k},\cdot) - \rho_{ex}(t_k^{n_k},\cdot)\|_{(L^2(\Omega))^2} \ .$$

The very last part of the convergence analysis consists in plotting and saving the following errors curves in EPS file format for summarizing the convergence of the involved numerical methods as the space-time mesh is refined:

- $\Delta t_k \mapsto e_{\mathbf{u},\Delta t_k,\mathfrak{M}_k}$, $\Delta t_k \mapsto e_{p,\Delta t_k,\mathfrak{M}_k}$ and $\Delta t_k \mapsto e_{\rho,\Delta t_k,\mathfrak{M}_k}$ (convergence as the the time mesh is refined),
- $h_{k,\max} \mapsto e_{\mathbf{u},\Delta t_k,\mathfrak{M}_k}$, $h_{k,\max} \mapsto e_{p,\Delta t_k,\mathfrak{M}_k}$ and $h_{k,\max} \mapsto e_{\rho,\Delta t_k,\mathfrak{M}_k}$ (convergence as the space mesh is refined).

8.5.4 Stability analysis according to the Reynolds number

In this paragraph, it is assumed that the user chose the answer `'IN_REYNOLDS'` to the question in the introduction of Section 8.5. Hence, (s)he will find in the non-modifiable part of the setup file the following line:

```
PARAMETERS.CV_STUDY = 'IN_REYNOLDS';
```

The goal is to study the stability of the involved numerical methods for several values of the Reynolds number Re and with a single space mesh $\mathfrak{M}$ and a single time step bound Δt. Indeed, it is well known that time semi-discrete schemes such as BDF2 methods that are described in Section 6.2 become unstable for large values or Re. For measuring the stability of the numerical methods according to Re, we first consider a Reynolds number family $(Re_1, \ldots, Re_K)$ that can be precised by the user in the setup file as follows:

```
% Reynolds number (provide several values for stability analysis)
PARAMETERS.PHYSICAL.RE = {Re_1, ..., Re_K};
```

Then the stability of the method is studied in terms of error between the approximation of $(\mathbf{u}, p)$ (and ρ for `NSDV` model), so an analytical solution is required. This why such analysis can be performed with `EXAC` and `EXACNEU` test cases only.

As for convergence studies, the dynamics of each simulation is not saved as `.mat`/`.h5` files but only the log file in the present casen and, assuming that the following lines are in the setup file

```
PARAMETERS.MODEL = 'themodel';
PARAMETERS.TESTCASE = 'thetestcase';
PARAMETERS.OUTPUT.DIRECTORY_NAME = 'thedirectory';
```

the simulation associated with the the Reynolds number Re_k will produce a log file in the subdirectory `thedirectory/themodel_thetestcase_`k.

Once all simulations are finished, NS2DDV analyses the log files that have been produced by each of these simulations and focuses on some specific diagnostics including the following errors:

- Error on the velocity with Reynolds number Re_k:

$$e_{\mathbf{u},\Delta t,\mathfrak{M}}(Re_k) = \max_{n=0,\ldots,N} \|\mathbf{u}_{\Delta t_k,\mathfrak{M}}(t_k^n,\cdot) - \mathbf{u}_{ex}(t_k^n,\cdot)\|_{(L^2(\Omega))^2} \ ,$$

- Error on the pressure with Reynolds number Re_k:

$$e_{p,\Delta t,\mathfrak{M}}(Re_k) = \max_{n=0,\ldots,N} \|p_{\Delta t,\mathfrak{M}}(t^n,\cdot) - p_{ex}(t^n,\cdot)\|_{L^2(\Omega)} \ ,$$

- Error on the density with Reynolds number Re_k (only for `NSDV` model):

$$e_{\rho,\Delta t,\mathfrak{M}}(Re_k) = \max_{n=0,\ldots,N} \|\rho_{\Delta t,\mathfrak{M}}(t^n,\cdot) - \rho_{ex}(t^n,\cdot)\|_{(L^2(\Omega))^2} \ .$$

The very last part of the convergence analysis consists in plotting and saving the following errors curves in EPS file format for the evolution of the errors $Re_k \mapsto e_{\mathbf{u},\Delta t,\mathfrak{M}}(Re_k)$, $Re_k \mapsto e_{p,\Delta t,\mathfrak{M}}(Re_k)$ and $Re_k \mapsto e_{\rho,\Delta t,\mathfrak{M}}(Re_k)$ as the Reynolds number becomes larger.

9 High Performance Computing features

9.1 Matlab Parallel Computing Toolbox

A fraction of the computations led in NS2DDV have been parallelized with the Matlab Parallel Computing Toolbox (PCT) from Mathworks (not free). The user can verify if (s)he is equipped with such toolbox by typing the following Matlab command:

```
>> ver distcomp
```

In the case where the answer is positive, the user can parallelize NS2DDV runs by choosing `'PCT'` to the last question of the script `generate_setup_file.m`. This will produce a setup file with the following parameters:

```
% Parallelization parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Number of CPU workers (default value = localcluster.NumWorkers)
PARAMETERS.PARALLELIZATION.NBWORKERS = localcluster.NumWorkers;
```

`PARAMETERS.PARALLELIZATION.NBWORKERS` is set to `localcluster.NumWorkers` by default (maximal number of Matlab workers that can be used in a single use of Matlab PCT with the cluster profile `localcluster`). The user can replace this value by the number of workers (s)he wants but should not bypass the hardware limit: in such case, Matlab will prematurely stop the run.

Note that 3 cluster profiles are currently available:

- `'LOCAL'`: profile cluster for a run on the local computer (can be run on any computer),
- `'MATHCALC'`: pre-built profile cluster for a run on the cluster of Paul Painlevé Laboratory (account needed),
- `'ZEUS'`: pre-built profile cluster for a run on the Computing Ressources Center of Lille University (account needed).

9.2 Resuming a simulation

NS2DDV embeds the ability to resume a simulation that has been stopped for any external reason (keyboard interruption signal, end of "batched" job, ...). To do this, the user must use the starting routine `start_ns2ddv` on a backup file (`.mat` binary format) instead of a setup file (`.m` text format).

```
>> start_ns2ddv('./BACKUP/backup_diags_0.mat');
```

Such backup file contains all the required data for resuming a simulation and is periodically generated and/or updated by the initial simulation. For managing the backup files

generation, the user must fill the following parameters in the setup file before initializing the simulation:

```
% Backup parameters
%%%%%%%%%%%%%%%%%%%%
% Number of backup files (default = 1)
% Each backup file contains the required data to restart the
% code after a critical crash
PARAMETERS.BACKUP.NB_FILES = 1;
% Backup frequency (default = 10)
PARAMETERS.BACKUP.FREQUENCY = 10;
% Backup file location
PARAMETERS.BACKUP.DIRECTORY_NAME = './BACKUP';
```

`PARAMETERS.BACKUP.FREQUENCY` is the frequency at which NS2DDV will generate a backup file. For example, if it is set to 10, a backup file will be generated at time iteration 0, 10, 20, 30...
`PARAMETERS.BACKUP.NB_FILES` stands for the number of distinct backup files to be generated. For example, setting this parameter to 5 and taking `PARAMETERS.BACKUP.FREQUENCY = 10` mean

- the file `**_backup_0.mat` will contain the data for performing time iteration 0,
- the file `**_backup_1.mat` will contain the data for performing time iteration 10,
- the file `**_backup_2.mat` will contain the data for performing time iteration 20,
- the file `**_backup_3.mat` will contain the data for performing time iteration 30,
- the file `**_backup_4.mat` will contain the data for performing time iteration 40,
- the file `**_backup_0.mat` will be replaced for containing the data for time iteration 50,
- the file `**_backup_1.mat` will be replaced for containing the data for time iteration 60,
- ...

If the simulation to be run requires a large number of time iterations, it is highly recommended to have several and distinct backup files.

10 Contact us

For questions about the code:

- Alexandre Mouton: alexandre.mouton@univ-lille.fr

For questions about the mathematics lying behind NS2DDV:

- Emmanuel Creusé: emmanuel.creuse@uphf.fr
- Caterina Calgaro: caterina.calgaro@univ-lille.fr

References

[1] C.-H. Bruneau and P. Fabrie, *Effective downstream boundary conditions for incompressible navier-stokes equations*, Int. J. Numer. Meth. Fluids, 19 (1994), pp. 693–705.

[2] ——, *New efficient boundary conditions for incompressible navier-stokes equations:a well-posedness result*, RAIRO Modélisation Mathématique et Analyse Numérique, 30 (1996), pp. 815–840.

[3] C. Calgaro, E. Chane-Kane, E. Creusé, and T. Goudon, *L infini-stability of vertex-based muscl finite volume schemes on unstructured grids; simulation of incompressible flows with high density ratios*, J. of Comput. Phys., 229.

[4] C. Calgaro, E. Creusé, and T. Goudon, *An hybrid finite volume-finite element method for variable density incompressible flows*, J. of Comput. Phys., 227 (2008), pp. 4671–4696.

[5] S. Dong, G. E. Karniadakis, and C. Chryssostomidis, *A robust and accurate outflow boundary condition for incompressible flow simulations on severely-truncated unbounded domains*, J. Comput. Phys, 261 (2014), pp. 83–105.

[6] J.-L. Guermond, P. D. Minev, and A. Salgado, *Error analysis of pressure-correction schemes for the time-dependent stokes equations with open boundary conditions*, SIAM J. Numer. Anal., 43 (2005), pp. 239–258.

[7] J.-L. Guermond and L. Quartapelle, *A projection fem for variable density incompressible flows*, J. of Comput. Phys., 165 (2000), pp. 167–188.

[8] J.-L. Guermond and A. Salgado, *A splitting method for incompressible flows with variable density based on a pressure poisson equation*, J. of Comput. Phys., 228 (2009), pp. 2834–2846.

[9] ——, *Error analysis of a fractional time-stepping technique for incompressible flows with variable density*, SIAM J. Numer. Anal., 49 (2011), pp. 917–944.

[10] S. K. Hannani, *Calcul d'écoulements laminaires et turbulents par une méthode d'éléments finis: Influence de la formulation*, PhD thesis, Université des Sciences et Technologies de Lille, 1996.

[11] M. S. Kilic, G. B. Jacobs, J. S. Hesthaven, and G. Haller, *Reduced navier-stokes equations near a flow boundary*, Physica D Nonlinear Phenomena, 217 (2006), pp. 161–185.