Master Recherche "Aérodynamique et Aéroacoustique"

"High-Fidelity Simulations for Turbulent Flows"

TP: High-order methods for compressible flows

Contents

1 Aim of the TP

The goal of the present TP is to study the influence of several numerical ingredients (grid resolution, time integration, spatial discretization and related parameters) on the quality of the solution for canonical 1D and 2D configurations.

2 Installation of the code

The code is written in `Fortran 90`[1] and can be compiled by means of GNU (`gfortran`) or Intel (`ifort`) compilers. It uses the software `CMake` to check file dependencies and compile the code.

Installation on Linux

For installation on Linux (Ubuntu Distribution), open a terminal and type:

1. `sudo apt update`

[1] More information on the Fortran programming language can be found here: http://www.idris.fr/formations/fortran/

2. `sudo apt upgrade`

3. `sudo apt install gfortran g++ cmake`

4. `sudo apt install python3-numpy python3-scipy python3-matplotlib`

This allows to install all the libraries needed to use the code. For Linux Distros other than Ubuntu, just use the correct syntax of its related package manager, the name of the packages to install are the same.

Installation on Windows

For installation on Windows, the simplest way is to install a virtual Ubuntu distribution in Windows. To do this, you should enable the "Windows Subsystem for Linux" option, and then download the Ubuntu app directly from the Microsoft Store. Once the installation is terminated, launch the app and the Ubuntu terminal will appear. You can then execute the same three steps shown above. If you need to directly access your Ubuntu home directory from Windows (for example, to copy or move files between Windows and Ubuntu), its complete path is the following (tested on Windows 10):
`C:\Users\[USERNAME]\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_[CODE]\LocalState\rootfs`

Working on DynFluid workstations

The workstations available at the DynFluid Laboratory are ready to use, no need to install or update any package. In this case, the use of the Fortran compiler of Intel (`ifort`) is to be preferred. To set up the environment, open a new terminal and write `module load compiler/latest`. Note that this has to be done for each new terminal that you will use.

Code structure and compilation

Once that all the packages have been installed and the terminal is ready, you can explore the code. Note that there are 5 directories:

- `bin`: folder where the executable will be stored when installing the code;
- `debug`: folder to use when you want to compile the code in `DEBUG` mode;
- `post`: folder where the simulation and its post-processing will be run.
- `release`: folder to use when you want to compile the code in `RELEASE` mode;
- `script`: folder containing some scripts needed to install the code;
- `src`: folder containing all the `Fortran` sources.

The difference between the `DEBUG` and `RELEASE` versions of the code resides in the different flags used to compile the sources. `DEBUG` will give you many information about possible problems in the program, whereas `RELEASE` is conceived for high performance.
For each test case, a Python script is provided in the related `post` folder (it is assumed that you already have `Python3` installed on your machine, and you already know the basics of the language). The scripts are provided as a guideline to show how the binary files written by the code can be read and post-processed. Note that these are just examples, you can build from them to develop your analysis. However, any other tool/script/language can be used to perform the post-processing of the simulations.

The procedure to install the code is the following:

1. Go to the `release` or `debug` folders, according to the version of the code you want to compile;

2. type `source ../scripts/setgfortran` or `source ../scripts/setintel` according to the libraries you installed (GNU or Intel);

3. type `../scripts/clean_and_cmake` or `../scripts/clean_and_cmake_debug` according to the version of the code you want to compile (and therefore the folder you are in). This will automatically construct the Makefile for the program;

4. type `make` to compile the all `Fortran` files;

5. type `make install` to create the executable `program.x`. It will be found in the `bin` folder.

Note that step 4 can be skipped if are not debugging your code. Once the executable has been created, it can be copied in the folder where the simulation will be run (which should contain the input file `param.inp`). To run the simulation, type `./program.x`. Once the simulation is finished, the post-processing can be launched by typing `python3 filename.py`, where `filename` is the name of python file available in the same folder.

3 Description of the code

3.1 Governing equations

The code solves the two-dimensional compressible Navier-Stokes equations in differential form, that read:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0 \tag{1}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \left(\rho u_i u_j + p\delta_{ij}\right)}{\partial x_j} = \frac{\partial \tau_{ij}}{\partial x_j} \tag{2}$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \left[\left(\rho E + p\right) u_j\right]}{\partial x_j} = \frac{\partial (u_i \tau_{ij} - q_j)}{\partial x_j} \tag{3}$$

Written under flux form, the system can be recast into:

$$\frac{\partial \boldsymbol{U}}{\partial t} = -\frac{\partial \boldsymbol{F}^e}{\partial x} - \frac{\partial \boldsymbol{G}^e}{\partial y} + \frac{\partial \boldsymbol{F}^v}{\partial x} + \frac{\partial \boldsymbol{G}^v}{\partial y} \tag{4}$$

where $\boldsymbol{U}$ represents the state vector, $\boldsymbol{F}$ and $\boldsymbol{G}$ the fluxes in the x and y directions, and the superscripts $(\bullet)^e$ and $(\bullet)^v$ the convective flux and the viscous fluxes, respectively:

$$\boldsymbol{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix} \quad \boldsymbol{F}^e = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho E + p)u \end{bmatrix} \quad \boldsymbol{G}^e = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ (\rho E + p)v \end{bmatrix} \quad \boldsymbol{F}^v = \begin{bmatrix} 0 \\ \tau_{11} \\ \tau_{12} \\ u_i\tau_{1i} - q_1 \end{bmatrix} \quad \boldsymbol{G}^v = \begin{bmatrix} 0 \\ \tau_{21} \\ \tau_{22} \\ u_i\tau_{2i} - q_2 \end{bmatrix} \tag{5}$$

Here, $E = e + \frac{u_i^2}{2} = c_v T + \frac{u_i^2}{2}$ is the specific total energy, e the specific internal energy, c_v the isochoric specific heat and T the temperature; τ_{ij} is the viscous stress tensor, which for a Newtonian fluid reads:

$$\tau_{ij} = 2\mu S_{ij} - \frac{2}{3}\mu S_{kk}\delta_{ij} \qquad \text{with} \qquad S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{6}$$

and q_i is the heat flux, modeled by means of Fourier's law $q_i = -\lambda \frac{\partial T}{\partial x_i}$ (λ being the thermal conductivity). The system is closed by means of the perfect gas equation of state, $p = \rho R_{\text{air}} T$.

3.2 Numerical methods

3.2.1 Time Integration

The time integration of the equations is performed by means of a standard 4-step Runge–Kutta method. Consider the differential equation

$$\frac{\mathrm{d}w}{\mathrm{d}t} = F(w) \tag{7}$$

An explicit $p-$stage RK algorithm can be written as

$$\begin{cases} w^0 = w^n \\ w^l = w^n + \alpha_l \Delta t F(w^{l-1}) \quad \text{for} \quad l = 1, .., p \\ w^{n+1} = w^p \end{cases} \tag{8}$$

Which is equivalent to

$$w^{n+1} = w^n + \sum_{j=1}^{p} \underbrace{\prod_{l=p-j+1}^{p} \alpha_l}_{\gamma_j} \Delta t^j \frac{\partial^j w^n}{\partial t^j} \quad \text{with} \quad \gamma_j = \frac{1}{j!} \tag{9}$$

The coefficients can be found in Bogey and Bailly (2004) (scheme "RKs4s").
The timestep Δt is selected at each iteration in order to respect both the CFL and Fourier condition:

$$\Delta t = \min\left(\Delta t_e, \Delta t_v\right) \tag{10}$$

with

$$\mathrm{CFL} = \Delta t_e \times \max_i \left[\frac{|u_i| + a}{\Delta x_i}\right] \qquad \Longrightarrow \qquad \Delta t_e = \frac{\mathrm{CFL}}{\max\limits_i \left[\frac{|u_i|+a}{\Delta x_i}\right]} \tag{11}$$

$$\mathrm{Fo} = \Delta t_v \times \max_i \left[\frac{\nu}{\Delta x_i^2}\right] \qquad \Longrightarrow \qquad \Delta t_v = \frac{\mathrm{Fo}}{\max\limits_i \left[\frac{\nu}{\Delta x_i^2}\right]} \tag{12}$$

3.2.2 Spatial Discretization

The discretization of the convective fluxes is performed by means of high-order centred differences. For a stencil of $2N+1$ points, the derivative reads

$$\frac{\partial f}{\partial x}(x_0) = \frac{1}{\Delta x} \sum_{j=-N}^{N} a_j [f(x_0 + j\Delta x)] = \frac{1}{\Delta x} \sum_{j=1}^{N} a_j [f(x_0 + j\Delta x) - f(x_0 - j\Delta x)] \quad \text{with} \quad a_j = -a_{-j} \tag{13}$$

The coefficients a_j are computed following two strategies:

- Maximize the formal order of accuracy of the numerical scheme (standard differences): in this case, a stencil of $2N+1$ points will result in a scheme of order $2N$.
- Minimize the dispersion error of the discretization (optimized differences): in this case, some coefficients will be adjusted to obtain a given formal order, and the other ones are used to minimize the phase error in a given wavenumber range. These schemes are called DRP (Dispersion-Relation Preserving)

The coefficients for both the standard (STD) and optimized (DRP) family can be found in Bogey and Bailly (2004). The discretization of the viscous fluxes, instead, is performed by means of lower-order (usually 4^{th}) standard differences.

It is known that centred differences are purely non dissipative. The previous schemes are not able to handle grid-to-grid oscillations (having $k\Delta x = \pi$, *i.e.* 2 points per wavelength), which often arise because of under-resolved perturbations, strong gradients or boundary conditions. These oscillations must therefore be removed in order to avoid divergence of the numerical procedure. This is usually done by introducing some form of numerical dissipation, either in the form of artificial viscosity or selective filtering. In the code, the selective filtering procedure is used; a filtered variable f^{flt} reads

$$f^{\text{flt}}(x_0) = f(x_0) - \sigma D_f(x_0) \quad \text{with} \quad 0 \leq \sigma \leq 1 \quad \text{and} \quad D_f(x_0) = \sum_{j=-N}^{N} d_j f(x_0 + j\Delta x) \tag{14}$$

The filter is applied sequentially in the different directions, after the end of the Runge–Kutta integration procedure. Since the coefficients are symmetric, the filter does not introduce dispersion errors. Similarly to the derivative discretization, the coefficients may be computed with the aim of either maximize the order of the filter (standard filters) or minimize its dissipation error (optimized DRP filters). Most of the coefficients can be found in Bogey and Bailly (2004).

3.2.3 Shock-capturing schemes

The previous high-order schemes are not able to handle strong gradients or flow discontinuities. In order to deal with non-smooth (shocked) flow configurations, a different numerical strategy must be used. An option is the use of shock-capturing methods, which rely in the addition of low-order numerical dissipation allowing to regularise the solution (hence "capture" the discontinuity). The code is equipped with three different shock-capturing schemes:

1. **DNC–Jameson**: this is essentially a high-order Jameson-like numerical dissipation. It combines a high-order and a low-order term, whose amplitudes are controlled by the adjustable coefficients k_2 and k_{10}. Classical values are $k_2 \in [0, 2]$ and $k_{10} \in [0.5, 2]$. Since this method includes a high-order dissipation term, the regular filter is switched off.

2. **Localized Artificial Diffusivity** (LAD) (Kawai et al., 2010): the main idea behind this method is to introduce artificial transport properties to regularize the solution. The strength of the artificial bulk viscosity and artificial thermal conductivity properties are controlled by means of the parameters c_β and c_λ, respectively. Classical values are $c_\beta \in [0, 1.5]$ and $c_\lambda \in [0, 0.02]$.

3. **Adaptive spatial filtering** (Bogey et al., 2009): a first-order filter is applied on top of the regular high-order one. The amplitude of the filter is controlled by the numerical parameter r_{th}. Classical values are $r_{\text{th}} \in [10^{-6}, 10^{-4}]$. Note that a lower value corresponds to apply the shock-capturing filter on a wider region, leading to smoother solutions.

For more details, one can refer to the articles provided in the bibliography or to the "High-order schemes for compressible flows" course slides.

4 Test cases

4.1 Sod's Shock Tube

The classical shock tube of Sod (Sod, 1978) corresponds to the following Riemann problem:

$$(\rho, u, p) = \begin{cases} (1, 0, 1) & x < 0 \\ (0.125, 0, 0.1) & x \geq 0 \end{cases} \tag{15}$$

The computational domain is $L = [-0.5, 0.5]$. Numerical results are compared at the time $t^* = 0.2$ with respect to the analytical solution, obtained by means of an exact Riemann solver and available in the file `post/1shocktube/REF/sod_exact.dat`. An example of results is provided in figure 1.

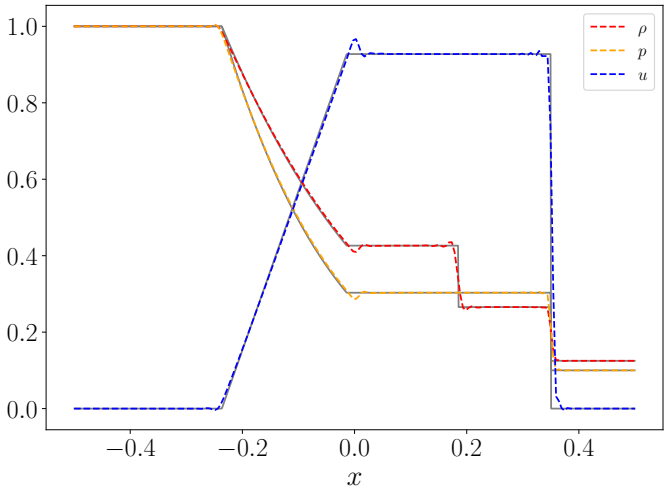

Figure 1: Example of results for Sod's shock tube problem. Profiles of density, pressure and velocity along the tube. The gray lines represent the analytical solution.

4.2 Shu-Osher problem

The Shu-Osher problem Shu and Osher (1989) consists of a M=3 shock propagating in a perturbed density field and allows to evaluate the behavior of the numerical scheme for a simplified shock-turbulence interaction configuration. The extent of the computational domain is $[-5, 5]$ and the initial conditions are

$$(\rho, u, p) = \begin{cases} (3.857143, 2.629369, 10.33333) & x < -4 \\ (1 + 0.2\sin(5x), 0, 1) & x \geq -4 \end{cases} \tag{16}$$

The final simulation time is $t^* = 1.8$. The reference solution is obtained by using a very fine grid ($N = 2000$) and is available in the file `post/2shuosher/REF/shu_osher.dat`. An example of results is provided in figure 2.

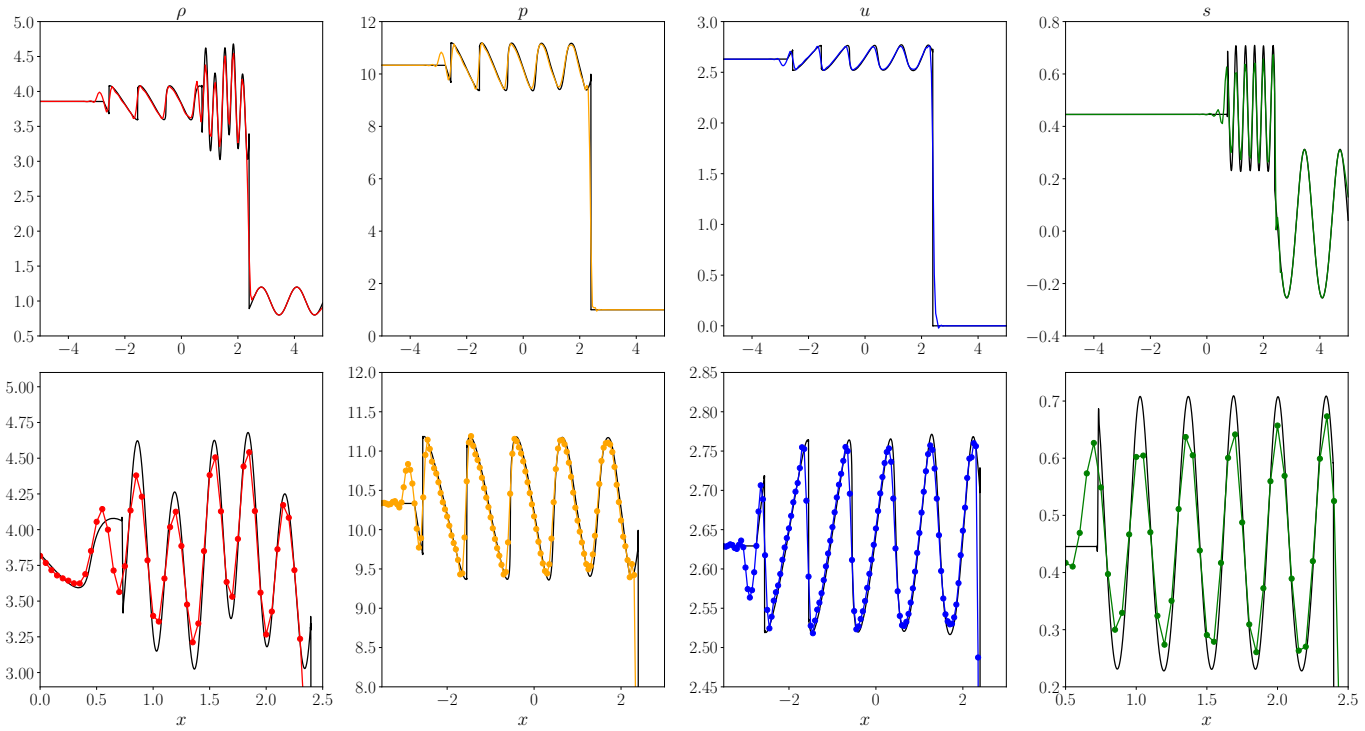

Figure 2: Example of results for the Shu-Osher problem. From left to right: Profiles of density, pressure, velocity and entropy. The bottom row shows a zoom of the computational domain. In each figure, the black lines denote the reference solution.

4.3 Inviscid Vortex Convection

The accuracy of the discretization scheme in smooth inviscid flow is quantified by means of the two-dimensional isentropic vortex advection problem, in which an inviscid vortex is superimposed to an uniform, perfect-gas air flow. The perturbations in velocity and temperature are given by

$$\begin{cases} \delta u = -\dfrac{y}{R}\Omega \\ \delta v = +\dfrac{x}{R}\Omega \\ \delta T = -\dfrac{\gamma-1}{2}\Omega^2 \end{cases} \qquad \text{with} \quad \Omega = \beta \exp\left(-\frac{1}{2\sigma}\left[\left(\frac{x}{R}\right)^2 + \left(\frac{y}{R}\right)^2\right]\right). \tag{17}$$

These allow to define, along with the isentropic relations, the initial flow conditions of the primitive variables:

$$\begin{cases} \tilde{\rho}_0 = (1+\delta T)^{\frac{1}{\gamma-1}} \\ \tilde{u}_0 = M_\infty \cos\alpha + \delta u \\ \tilde{v}_0 = M_\infty \sin\alpha + \delta v \\ \tilde{p}_0 = \frac{1}{\gamma}(1+\delta T)^{\frac{\gamma}{\gamma-1}} \end{cases} \tag{18}$$

where the subscript $(\bullet)_0$ denotes a quantity given at $t=0$, and the symbol $\widetilde{(\bullet)}$ indicates a nondimensional quantity (ρ_∞, T_∞ and a_∞ being the characteristic density, temperature and velocity, respectively). In the classical case (Shu, 1998; Spiegel et al., 2015), one has $R{=}\sigma{=}1$, $M_\infty{=}\sqrt{\frac{2}{\gamma}}$ and $\alpha{=}45°$. Periodic conditions are applied at the boundaries. The length of the computational domain is $[L_x, L_y] = [-10, 10]$. An example of the results is shown in figure 3.

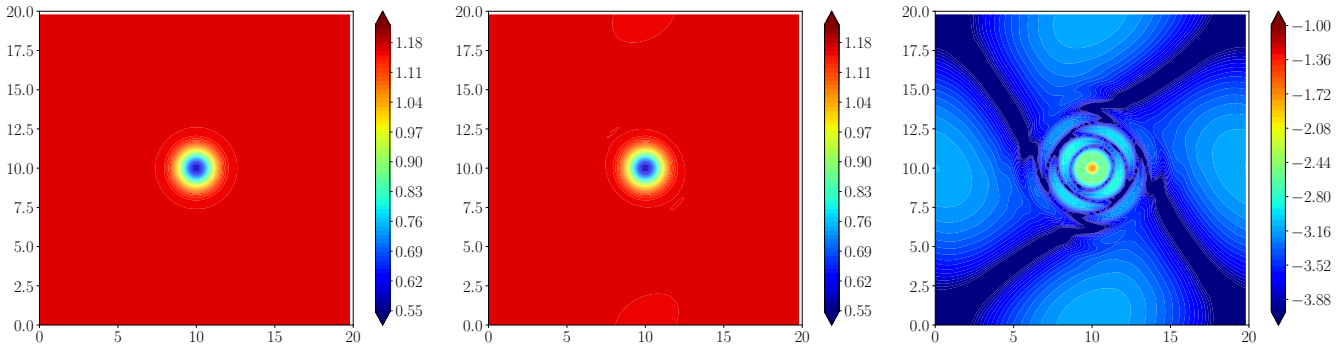

Figure 3: Example of results for the inviscid vortex convection problem. Left: density contours at $t^* = 0$. Center: density contours at $t^* = 200$ (that is, after 20 cycles). Right: Relative error between the two density fields in logarithmic scale, $\log \frac{|\rho_{\text{num}} - \rho_{\text{ex}}|}{\rho_{\text{ex}}}$.

4.4 Shock-Vortex interaction

This case considers a single Taylor vortex interacting with a planar shock wave. The numerical setup is defined in Bogey et al. (2009). The shock wave is defined by an upstream Mach number $M_s = \frac{u_\infty}{a_\infty} = 1.2$; the initial tangential and radial velocities of the vortex are expressed by

$$u_\theta(r) = M_v r \exp\left[\frac{1-r^2}{2}\right] \qquad \text{and} \qquad u_r(r) = 0 \tag{19}$$

where the distance from the vortex core r is non-dimensionalized by the vortex radius R, and the Mach number of the vortex is$M_v = \frac{u_{\theta,\max}}{a_\infty} = 0.25$. The density and pressure distributions are given by

$$\rho(r) = \left[1 - \frac{\gamma-1}{2} M_v^2 \exp(1-r^2)\right]^{\frac{1}{\gamma-1}} \qquad \text{and} \qquad p(r) = \frac{1}{\gamma}\rho(r)^\gamma \tag{20}$$

Note that $t = 0$ in the article of Bogey et al. (2009) corresponds to $t^* = 19$ in the code. An example of the results is shown in figure 4.

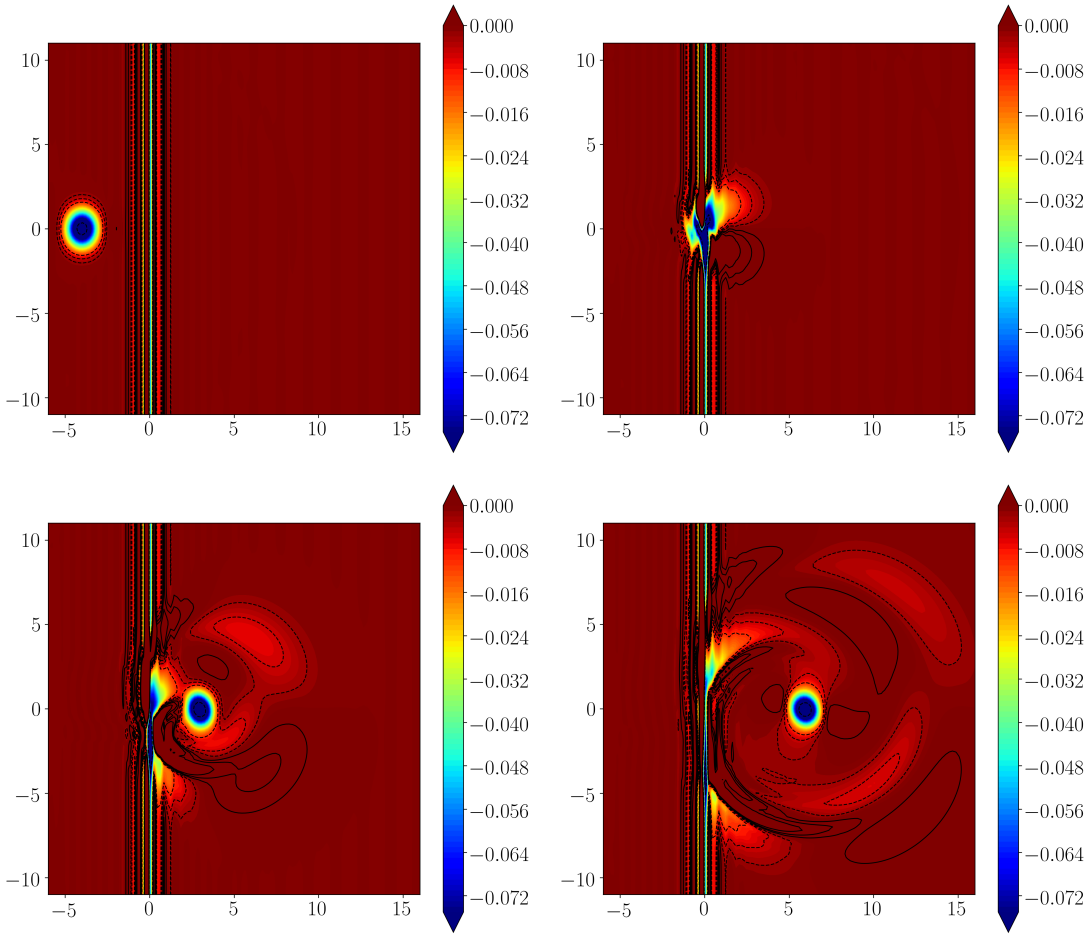

Figure 4: Example of results for the shock-vortex interaction problem. Isocontours of pressure difference Δp, defined as $\Delta p = (p - p_\infty)/p_\infty$ upstream of the shock, and $\Delta p = (p - p_s)/p_s$ downstream of it. Snapshots are shown at times $t = 16$, 20, 24 and 28.

4.5 Taylor-Green Vortex

The Taylor-Green Vortex sheet (TGV 2D) The length of the computational domain is $L_x \times L_y = [2\pi, 2\pi]$; periodic conditions are applied at the boundaries. The solution is as follows:

$$u(x,y) = \quad u_\infty \sin(kx)\cos(ky)F(t) \tag{21}$$

$$v(x,y) = -u_\infty \cos(kx)\sin(ky)F(t) \tag{22}$$

$$p(x,y) = \quad p_\infty + \frac{\rho u_\infty^2}{4}\left[\cos(2kx) + \cos(2ky)\right]F(t)^2 \tag{23}$$

where $F(t) = \exp(-2\nu k^2 t)$. Plugging these conditions in the 2D incompressible Navier–Stokes equations, one can show that the convective term is canceled by the pressure gradient term, whereas the viscous term cancels the time derivative. It results that, in the inviscid limit ($\nu \to 0$), the initial vortices will never decay. The value of the dynamic viscosity is such that the initial Reynolds number is $\mathrm{Re} = 2000$.
After a given amount of time, the laminar regime breaks down and the flow becomes turbulent. The laminar-to-turbulence transition depends on the wavenumber k at which energy is initially injected. For $k = 8$, destabilization of the flow is registered approximately at $t^* \approx 12$ (an example of results is shown in figure 5). For smaller k the laminar regime is maintained for longer times, allowing a direct comparison with the analytical solution of eqs. (21)-(23).

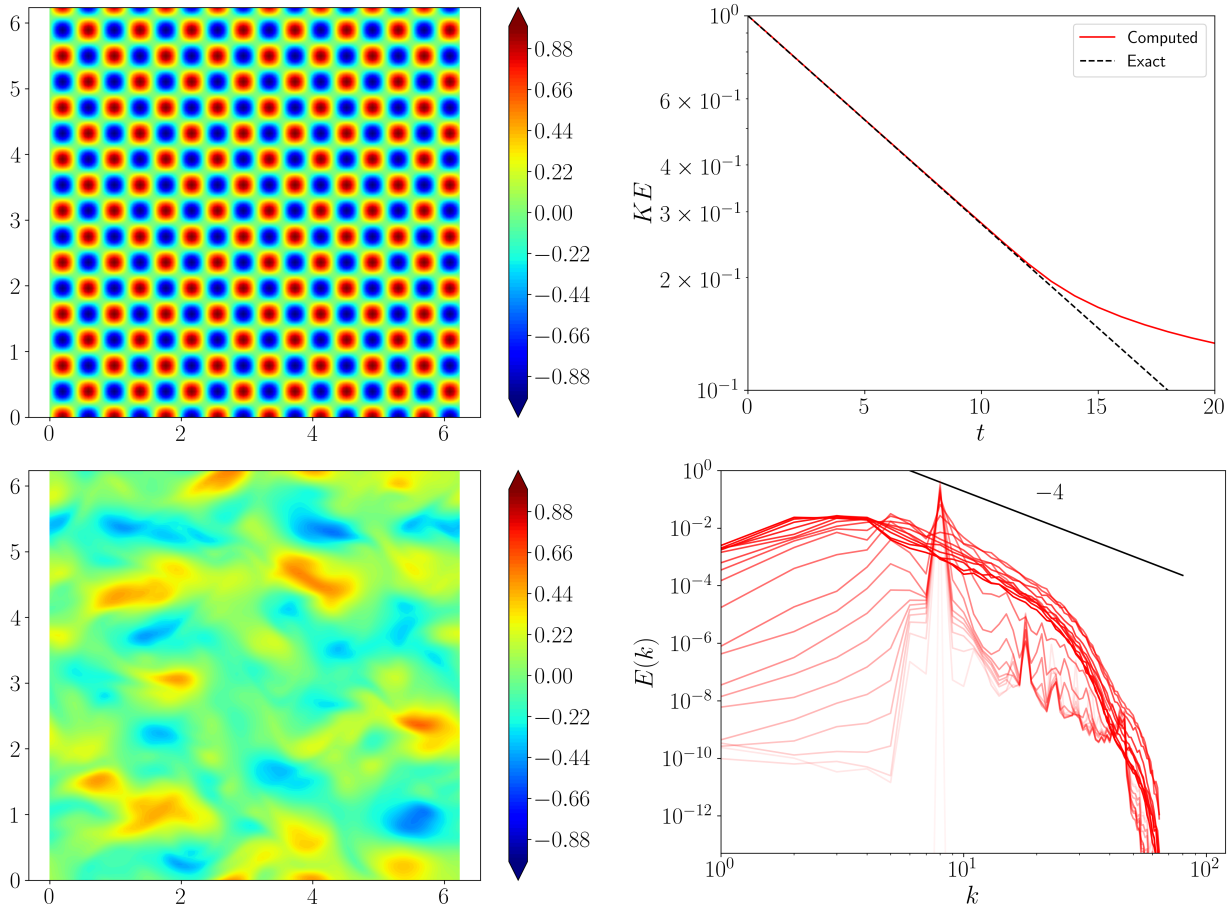

Figure 5: Example of results for the 2D Taylor Green Vortex problem. Left: isocontours of velocity at $t^* = 0$ and 20. Right: Time evolution of kinetic energy and kinetic energy spectra.

5 Work to do

5.1 Implementation of a numerical method

Each group will be asked to implement one **time integration** method and one **spatial discretization** scheme, different from those already available in the code. Specifically:

- For time integration, the following schemes can be chosen:

  1. 1^{st}-order, 1-stage Runge–Kutta (i.e., Forward Euler);
  2. 2^{nd}-order, 2-stage Runge–Kutta;
  3. 3^{rd}-order, 3-stage Runge–Kutta;
  4. 4^{rd}-order, 6-stage Runge–Kutta.

  One can refer to the "Numerical simulations for unsteady flows" course slides for the first three methods, whereas the reference for the last one can be found in the article of Bogey and Bailly (2004), which is available in the bibliography folder (the method is called "RKo6s" in the article).

- For spatial discretization, the following schemes can be chosen:

  1. 2^{nd}-order, 3-points standard differences;
  2. 4^{th}-order, 5-points standard differences;
  3. 6^{th}-order, 7-points standard differences;
  4. 12^{th}-order, 13-points standard differences;
  5. 4^{th}-order, 7-points optimized differences;
  6. 4^{th}-order, 11-points optimized differences;
  7. 4^{th}-order, 13-points optimized differences.

  It should be considered that, thanks to the modularity of the code, changing the spatial discretization requires only a few simple changes, therefore it is suggested to test multiple numerical schemes. All the coefficients are already available in the subroutine `deriv_coeff.f90`. The high-order filters and shock-capturing methods do not require any changes, they are already available for use in the code.

5.2 Parametric studies

Each group should choose a 1D and a 2D test case among the configurations presented in section 4. The goal is to study the influence of several numerical ingredients on the chosen test cases. The possibilities include:

1. Grid resolution;
2. Spatial discretization scheme: order and type of centred difference;
   (do not forget to test also the scheme you implemented)
3. Amplitude of the high-order filter σ;
4. Type of shock-capturing scheme and sensitivity to its parameters (only for shocked flows);
5. Time integration scheme: sensitivity to CFL;
   (do not forget to test also the scheme you implemented)

Bonus question: try to retrieve the formal order of accuracy of the spatial scheme using the simulation results.

Important remarks:

- Change **only one** parameter per simulation in order to assess its influence on the results. It does not make any sense, for instance, to change the time integration method when performing a grid-refinement study, or to change the filter coefficient when studying the influence of the CFL number, etc.
- Do not mix standard and optimized derivatives/filters: if you are using DRP derivatives, use also DRP filters (and the other way around for STD).
- Be consistent with the number of points used for derivatives and filters: apply a filter on $2N+1$ points when using a derivative on $2N+1$ points.
- To ease comparison when doing parametric studies, adapt the Python scripts in order to plot quantities on the same figure.

5.3 Deliverables

A report describing the main results of the study performed must be submitted by each group. For each test case analysed, the report should contain, *a minima*, the description of the configuration, the type of analysis performed and a discussion on the results. The use of LaTeXfor report typewriting is strongly encouraged and will be considered a plus. The code used for the simulations and for the post-processing must also be submitted, along with an example input card for each configuration studied and any other relevant files (e.g. animations). All the material must be sent by email to luca.sciacovelli@ensam.eu; the reception will be acknowledged. The deadline for the submission is set to **24/02/22**.

For any type of problem (installation, implementation, bugs, etc.) you can send an email with a brief description of the problem, attaching the entire code and the input file allowing to reproduce the problem. For more complex problems, a TEAMS session can be organized upon request.

References

C. Bogey and C. Bailly. A family of low dispersive and low dissipative explicit schemes for flow and noise computations. *Journal of Computational Physics*, 194(1):194–214, 2004.

C. Bogey, N. De Cacqueray, and C. Bailly. A shock-capturing methodology based on adaptative spatial filtering for high-order non-linear computations. *Journal of Computational Physics*, 228(5):1447–1465, 2009.

S. Kawai, K Santhosh, and S. Lele. Assessment of localized artifical diffusivity scheme for large-eddy simulation of compressible turbulent flows. *Journal of Computational Physics*, 229(5):1739–1762, 2010.

C.-W. Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, pages 325–432. Springer, 1998.

C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, II. 83(1):32–78, 1989.

G. A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1–31, 1978.

S. C. Spiegel, H. T. Huynh, and J. R. DeBonis. A Survey of the Isentropic Euler Vortex Problem using High-Order Methods. In *22nd AIAA Computational Fluid Dynamics Conference*, page 2444, 2015.