This homework is about advection in dimension 2: find $\phi(\mathbf{x}, t)$ solution of

$$\partial_t \phi + \nabla \cdot (\boldsymbol{u}\phi) = \phi \nabla \cdot \boldsymbol{u}$$

with appropriate boundary conditions. Here, we assume that velocity vector $\boldsymbol{u}(\mathbf{x})$ is given as an input, and we consider it to be divergence free. We use here a nodal discretization on triangular elements:

$$\mathbf{x} \in D^k \quad : \quad \phi_h^k(\mathbf{x}, t) = \sum_{i=1}^{N_p} \phi_h^k(\mathbf{x}_i, t) l_i^k(\mathbf{x}),$$

where $l_i(\mathbf{x})$ is the multidimensional Lagrange polynomial defined by appropriate grid points, $\mathbf{x}_i$, on the element D^k and $N_p = (p+1)(p+2)/2$. Flux vector is

$$\boldsymbol{f} = \boldsymbol{u}(\boldsymbol{x})\phi(\boldsymbol{x}, t) = (u_x \phi, u_y \phi).$$

The DG formulation can be written in extenso as:

$$\int_{D^k} \partial_t \phi_k^k l_i^k dV - \int_{D^k} \phi_k^k u_x \partial_x l_i^k dV - \int_{D^k} \phi_k^k u_y \partial_y l_i^k dV + \int_{\partial D^k} \hat{\mathbf{n}} \cdot \mathbf{f}^* \; l_i^k d\mathbf{S} = 0 \quad , \quad i = 1, \ldots, N_p$$

$$\mathbf{f}^* \cdot \hat{\mathbf{n}} = \begin{cases} \phi^-(\mathbf{u} \cdot \hat{\mathbf{n}}) & \text{if} \quad (\mathbf{u} \cdot \hat{\mathbf{n}}) > 0 \\ \phi^+(\mathbf{u} \cdot \hat{\mathbf{n}}) & \text{if} \quad (\mathbf{u} \cdot \hat{\mathbf{n}}) < 0 \end{cases}$$

where ϕ^- is the value of ϕ taken at the inside of the element, and ϕ^+ is taken at the outside of the element, i.e. inside the neighbouring element. In other words, we consider an upwind scheme.

1. You must first install Gmsh's SDK that can be found at `http://gmsh.info/`. Python examples can be found in `gmsh/tutorials/python`. As we are using the `Python` interface, the easiest is to install the Gmsh API via `pip` with the following command in your usual terminal:

   ```
   pip install gmsh
   ```

2. Implement the quadrature free formulation applied to the scalar advection case. Gmsh will provide you high order elements with equi-distributed node systems. The conditioning of such systems is not optimal. Yet, it is sufficiently stable for orders $p \leq 7$. You should thus write a `python` code that solves the problem with the following signature

   ```
   sol = Advection2d ( meshFileName, dt, m, f, u, rktype, interactive )
   ```

   Input parameters of `Advection2d` are described below:

   - A mesh is provided as input in the Gmsh format. The path for finding that mesh is given in `meshFileName`. The information about polynomial order p as well as the mesh topology and geometry is part of that file.

- We assume a constant timestep `dt`. There are m time steps in the simulation so the final time is $m \times dt$.
- Parameter f is a python function that describes initial conditions for ϕ. It must be of the form:

```
def my_initial_condition (x) :
   xc = x[:,0]-1/2
   yc = x[:,1]-1/2
   l2 = xc**2+yc**2
   return numpy.exp(-l2*10)
```

- Parameter u is a python function that describes the velocity field $\boldsymbol{u}$. It must be of the form:

```
def my_velocity_condition (x) :
   return x * 0 + 1
```

- Parameter `rktype` is a string that defines the time stepping scheme that is used during the simulation. You must implement the following schemes: `rktype = ''ForwardEuler''`, `rktype = ''RK22''` and `rktype = ''RK44''`.
- When parameter `interactive` is set to `True`, the solution should be plotted interactively using Gmsh's API.

The output parameter of `sol` is a `numpy` array of size $N_T \times N_p$ that corresponds to the solution at time step m. Here N_T is the number of triangles in the mesh.

3. We propose here two test cases that are classical validation benchmarks.

   (a) The first one is called the Zalezak disk (see `https://www.youtube.com/watch?v=p03yDzcdv6c`). The advection test of Zalesak is often used in the literature to demonstrate the accuracy of an interface advection algorithm. In this example, the initial data is a slotted disk centered at $(50, 75)$ with a radius of 15, a width of 5, and a slot length of 25. The advection is driven by a velocity field:

$$u_x = (\pi/314)(50 - y)$$

$$u_y = (\pi/314)(x - 50)$$

   The disk completes one revolution every 628 time units. The computational domain is a circle of radius 50 so that the flow satisfies $\boldsymbol{u} = \boldsymbol{0}$ on the boundaries. The solution must be initialized as follows. Assume $d(\boldsymbol{x})$ to be the distance to the notched disk at time $t = 0$. Solution is initialized as

$$\phi(\boldsymbol{x}, 0) = \begin{cases} -1 & \text{if} \quad (\exp(d) - 1) < -1 \\ (\exp(d) - 1) & \text{if} \quad -1 \leq (\exp(d) - 1) \leq 1 \\ 1 & \text{if} \quad (\exp(d) - 1) > 1. \end{cases}$$

Figure 1 shows the initial data.

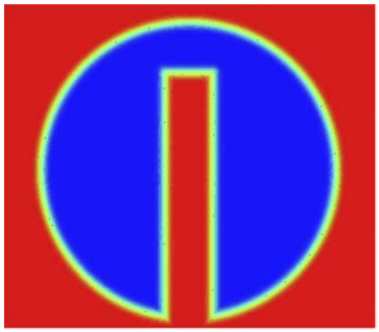

Figure 1: Initial data for the Zalesak's disk test case.

(b) The second one is the "vortex in a box". We consider the following stream function:

$$\psi(x,y) = \frac{1}{\pi}\sin^2(\pi x)\sin^2(\pi y)$$

that defines the following velocity field

$$u_x = \frac{\partial \psi}{\partial y} \quad , \quad u_y = -\frac{\partial \psi}{\partial x}.$$

We consider a disk of radius 0.15 placed at $(0.5, 0.75)$ and the initial solution $\phi(x,y,0) = (x-0.5)^2+(y-0.15)^2-0.15^2$. The computational domain is a square of size $[0,1]\times[0,1]$. The velocity satisfies $\boldsymbol{u} = \boldsymbol{0}$ on the boundaries of the unit square. The resulting velocity field stretches out the circle into a very long, thin filament, see Figure 2.

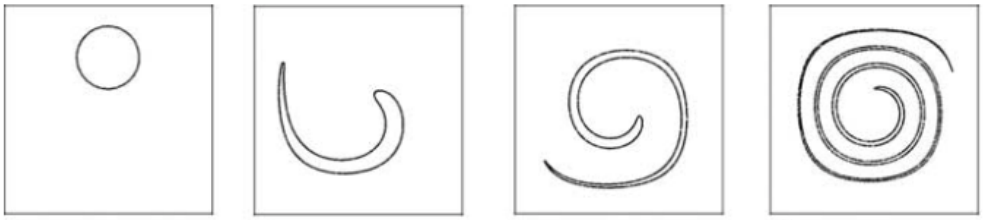

Figure 2: Time evolution of the iso-zero level set in the vortex flow.

Some instructions for the report:

The first test case (Zalezak disk) is able to give some indication on diffusion errors of an interface capturing method. At $t = 628$, the notched disk should come back to its initial position with its initial shape. Try this test case first for various polynomial orders and various mesh sizes and discuss what you observe after one full revolution. Find a way to assess the accuracy of your DG scheme, both in terms of mesh size and polynomial orders.

The second example (vortex in a box) tests the ability of the DG scheme to accurately resolve thin filaments on the scale of the mesh which can occur in stretching and tearing flows. Solve the problem up to a final time of $t = 4$ time units for various meshes and polynomial orders. Draw the iso-zero contour of $u(\boldsymbol{x}, t)$ at different times as in Figure 2. Here, you should discuss the ability of your scheme to capture features of sizes that are smaller and smaller. A question you could be able to answer is: *how many elements per wavelength do I need in order to have a good accuracy (this of course depends on p)?*

We strongly suggest you to refer to the following reference for notations and details about the quadrature-free DGM method:

Marchandise, E., Remacle, J.F., & Chevaugeon, N. (2006). A quadrature-free discontinuous Galerkin method for the level set equation. *Journal of Computational Physics, 212(1)*, 338-357.

Practical information:

Groups: The groups are the ones indicated on Moodle.

Collaboration: You are allowed, and even encouraged, to exchange ideas on how to address this assignment with students from other groups. However, you must do all the writing (report and codes) only with your own group; it is strictly forbidden to share the production of your group. Plagiarism will be checked.

Writing: The report should contain your answers to the questions as well as your figures. The report should have **maximum 4 pages (not including the header page, the figures and the bibliography)**. Please, do not include your code in the report. Take care of the writing of your report and the structure of your codes. Concision and quality of the writing, clarity of explanations and rigor will be taken into account.

Language: All reports and communications are equally accepted in French and English.

Deliverables: Each group is asked to submit on Moodle

- A report named `report.pdf`,
- A Python module named `advection2d.py` containing the function `advection2d` as specified above; this module should not execute anything so that it can be tested.
- A Python script named `run.py` that reproduces all the figures in your report.
- Optionally, animations can be provided as `.gif` files.

Deadline: The homework is due April 15th at 6pm.

Questions: You can address questions either at the Q&A session on Thursday March 31st at 2 p.m. (**be careful, there will be only one Q&A session for this homework as there are no classes the following 2 weeks!**), or by sending an email to the teaching assistants. Direct messages on Teams will not be considered.