Programming Homework 01

! Submission Deadline: 30.04.2025 - 23:55

- Your homework solution has to be handed in as a group solution via Moodle.
- Your notebooks must run without errors in our environment on the RWTH-Cluster. See here for more information on how to get started.

! Participant list

Please add your names and student ID (Matrikelnummer) here:

1. Name, ID
2. Name, ID
3. Name, ID
4. Name, ID

1 Learning goals

- Derive a **weak formulation** for a scalar advective PDE.
- Learn the basics about the **Discontinuous Galerkin (DG)** method for said PDE
- Learn how to **weakly enforce boundary conditions** in a DG context
- Switch between **explicit and implicit** time stepping in FenicX
- Visualize Streamlines and Pathlines

! Before you start

Make sure that you update the `library` folder in your `CMM` directory. Download link: library.tar.gz
Make sure that you update your `dolfinx_latest.sif` container image.
You can use the following command on the RWTH cluster/Linux:

```
wget https://mbd.pages.rwth-aachen.de/courses/cmm/content/exercises/notebooks/homework0
rm -rf library
wget https://mbd.pages.rwth-aachen.de/courses/cmm/content/exercises/notebooks/library.t
tar -xvzf library.tar.gz
rm library.tar.gz
rm dolfinx_latest.sif
apptainer pull oras://registry.git.rwth-aachen.de/mbd/courses/containers/dolfinx:latest
```

2 Imports

```
import os
from mpi4py import MPI
import numpy as np
import dolfinx
from dolfinx import fem
import basix
import tqdm
from petsc4py import PETSc
import numpy as np
import ufl
import pyvista
from  dolfinx.fem import petsc
from IPython.display import Image, display

import library.plot
import library.helpers
import library.limiter


CMM_DIR = os.getcwd()
os.environ["PYTHONPATH"] = f"{CMM_DIR}:{os.environ.get('PYTHONPATH', '')}"

output_dir = library.helpers.make_unique_dir(os.path.join(CMM_DIR, 'ex01'))
```

3 Problem statement

In the lecture, you learned about the general template for a **generic transport equation** of the form

$$\partial_t \Psi + \nabla \cdot (\Psi \mathbf{v}) = \nabla \cdot (\mathbf{D} \nabla \Psi) + S$$

where Ψ is our transported quantity, e.g. a density, the height of a water wave, a temperature, momentum or energy. v is the convective transport velocity, $\mathbf{D}$ is the diffusion coefficient (it can be a scalar or a matrix) and a source of production of decay S.

In the tutorial from last week 'My first FenicsX program', you already learned about the weak formulation of the diffusive term for a steady state problem in conjunction with its numerical approximation using the FEM.

In this tutorial, we will focus on **unsteady convective transport** which we want to solve using the Discontinuous Galerkin (DG) method.

Unsteady advection equation:

Given the scalar PDE

$$\begin{aligned} \partial_t q + \nabla \cdot (q\mathbf{v}) &= 0 \quad q \in \Omega \\ \nabla q \cdot \mathbf{n} &= 0 \quad \text{on } \partial\Omega \end{aligned} \quad ,$$

where $q \in \mathbb{R}$ is the unknown of the system, $v(t, x) = (2, 1)^T \in \mathbb{R}^2$ is a given transport velocity field with the domain $\Omega = [0, 3] \times [0, 3] \in \mathbb{R}^2$ and it's boundary $\partial\Omega$.

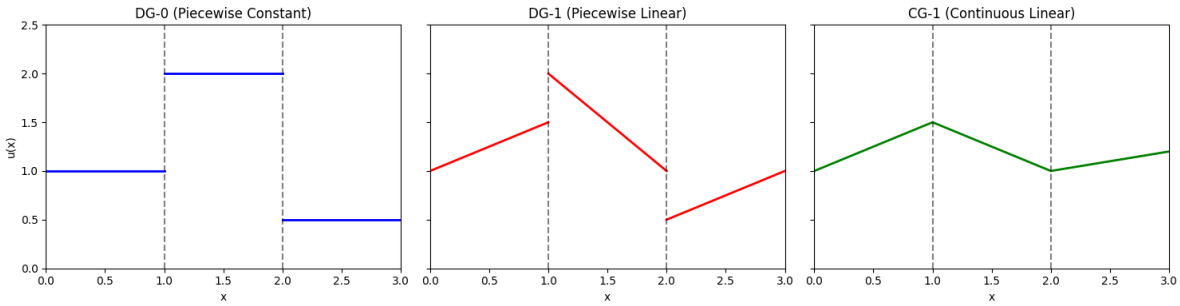
Weak formulation:

We want to approximate our unknown $q \in V = DG(1)$, where $DG(1)$ are Legendre polynomials of first order. Note that $DG(1)$ and $CG(1)$ can both be Legendre polynomials of first order. However, in $DG(1)$ the basis functions are **not required to be continuous** across cell interfaces (facets). Hence the name of discontinuous and continuous Galerkin elements. This has two immediate implications:

1. We need two degrees of freedom (dof) for each vertex/node in the inner part of our domain. This increases the storage demand of DG compared to a CG-FEM formulation.
2. Accounting for these **jumps** (two function values at one physical location) introduces new terms in our weak formulation

An illustration of DG(0), DG(1) and CG(1) is given below:

```
fig = library.plot.DG_illustration()
```



4 A primer to the Discontinuous Galerkin method

Let's now start with our weak formulation. In contrast to the usual weak form derivation in FEM, where we multiply the PDE with a test function $\phi \in V$ over the whole domain Ω, we want to start our derivation for a single element e in our mesh. This is only didactic reasons and will result in the identical weak form as if one starts with the integration over the full domain. We will do the derivation step by step by

1. mutiplying the PDE with a test function $\phi^e \in V = DG(1)$ and integrate over the domain Ω^e, the domain where the basisfunction ϕ^e is non-zero. Note that ϕ^e indicates the basis on **one element of the mesh only**. We also perform integration by parts with the goal to rewrite the advective term in divergence form.
2. we use the divergence theorem to transform the volume integral into a surface (facet) integral.
3. introducing the concept of a **numerical flux** in order to resolve the ambiguity of the value of q^n on the facet between two elements.
4. separating internal and boundary facets. The boundary conditions on the boundary facets will be **weakly enforced** in the numerical flux.
5. Sum over all elements to recover the integral over the full domain $\int_\Omega \cdot\, dx = \sum_e \int_{\Omega^e} \cdot\, dx$
6. Recast the formulation in a form that is close to the final formulation in FenicX.

Step 1. (integration by parts)

$$\int_{\Omega^e} \partial_t q \phi^e \, dx + \int_{\Omega^e} \nabla \cdot (q\mathbf{v})\, \phi^e \, dx = 0$$
$$\Leftrightarrow \int_{\Omega^e} \left(\frac{q^{n+1} - q^n}{\Delta t} \right) \phi^e \, dx + \int_{\Omega^e} \nabla \cdot (q^n \mathbf{v} \phi^e) \, dx - \int_{\Omega^e} \nabla \phi^e \cdot (q^n \mathbf{v}) \, dx = 0$$

Note that in the second step, we used the **explicit Euler** time stepping scheme to approximate the time derivative. If we replace $q^n \to q^{n+1}$ in the convective term, the formulation will result in an **implicit Euler** scheme. The term $\nabla \phi^e \cdot (q^n \mathbf{v})$ is a **jump term** which we need to account for in our weak formulation. We will see later how this term can be interpreted.

Step 2. (divergence theorem)

$$\Leftrightarrow \int_{\Omega^e} \left(\frac{q^{n+1} - q^n}{\Delta t} \right) \phi^e \, dx + \int_{\partial\Omega^e} (q^n \mathbf{v} \phi^e \cdot \mathbf{n}^e) \, ds - \int_{\Omega^e} \nabla \phi^e \cdot (q^n \mathbf{v}) \, dx = 0$$

After using the divergence theorem on the convective term, we obtain an integral over the boundaries of our element $\partial\Omega^e$ and the normal vector $\mathbf{n}^e$ pointing out of the element. Now look at the image of the 1D representation of the DG(1) element above and note that there are two different types of facets:

- **interior facets**: facets between two elements
- **boundary facets**: facets between an element and the domain boundary

This is different to the CG-FEM formulation, where we only had a boundary integral over the domain boundary. It is now important to realize that in $\int_{\partial\Omega^e} (q^n \mathbf{v} \phi^e \cdot \mathbf{n}^e) \, ds$ the value q^n is not unique on the facet f. It has two values, one on the element e that we are currently looking at and another value at its neighboring element. We will now denote $q^n|_{f^e} = q^{n,+}$ and $q^n|_{f^{neighbor}} = q^{n,-}$. Similarly, we denote $n^e = n^+$.

Step 3. (numerical flux)

It is still unclear which value we should assign to q^n on the facet. However, a solution can be borrowed from what we have already learned in the Finite Volume Method (FVM), as this method is build around discontinuous solutions at the facets. Without going into any details, the idea is to replace the flux $F := q^n \mathbf{v} \cdot \mathbf{n}^e$ in the boundary integral $\int_{\partial\Omega^e} (q^n \mathbf{v} \phi^e \cdot \mathbf{n}^e) \, ds$ with a **numerical flux** F^{num}. Typical choices for this numerical flux are the **upwind** and the **Lax-Friedrichs** flux. We will focus on the latter which is defined as

$$F^{num} = \frac{1}{2} (q^{n,+} + q^{n,-}) \, \mathbf{v} \cdot \mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max} \right) (q^{n,-} - q^{n,+})$$

where $|\lambda|_{max}$ is the maximal eigenvalue of the Jacobian of the flux function. In our case $|\lambda|_{max} = |\mathbf{v} \cdot \mathbf{n}^+|$.

Step 4. (weakly enforced boundary conditions)

Inserting the numerical flux into our weak form, now also want to differentiate between boundary facets on the domain boundary, denoted as $\int_{\partial\Omega^e_{bnd}} \cdot \, ds$ and interior facets $\int \partial\Omega^e_{int} \cdot \, dS$. We can now write the weak form as

$$
\begin{aligned}
&\int_{\Omega^e} \left(\frac{q^{n+1} - q^n}{\Delta t} \right) \phi^e \, dx - \int_{\Omega^e} \nabla \phi^e \cdot (q^n \mathbf{v}) \, dx \\
&+ \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,-} \right) \mathbf{v} \cdot \mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max} \right) \left(q^{n,-} - q^{n,+} \right) \right] \phi^+ \right) dS \\
&+ \int_{\partial\Omega^e_{bnd}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,-} \right) \mathbf{v} \cdot \mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max} \right) \left(q^{n,-} - q^{n,+} \right) \right] \phi^+ \right) ds = 0
\end{aligned}
$$

Weak enforcement of boundary conditions means that we do not alter the PDE or linear system in order to guarantee a certain boundary value. We rather build in the boundary condition in our weak formulation. In our case, we can simply prescribe the value in the $q^{n,-}$ in the boundary integral $\int_{\partial\Omega^e_{bnd}} \cdot dS$. To realize the extrapolation or outflow boundary condition $\nabla q \cdot \mathbf{n} = 0 \quad \text{on } \partial\Omega$, we simply copy the value from the interior side of the facet to the exterior side, since a constant across the interface results in a zero gradient.

Instead of simplifying our weak form for our particular boundary condition, we only replace $q^{n,-}$ with $q^{n,G}$ in the boundary integral. $q^{n,G}$ denotes the *ghost cell* value on the exterior side of the boundary facet and can also be used to model inflow or Dirichlet boundary conditions.

The weak form now reads

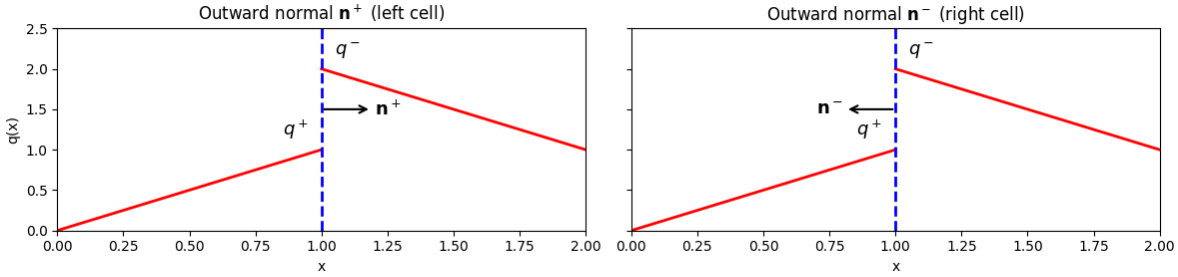
$$
\begin{aligned}
&\int_{\Omega^e} \left(\frac{q^{n+1} - q^n}{\Delta t} \right) \phi^e \, dx - \int_{\Omega^e} \nabla \phi^e \cdot (q^n \mathbf{v}) \, dx \\
&+ \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,-} \right) \mathbf{v} \cdot \mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max} \right) \left(q^{n,-} - q^{n,+} \right) \right] \phi^+ \right) dS \\
&+ \int_{\partial\Omega^e_{bnd}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,G} \right) \mathbf{v} \cdot \mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max} \right) \left(q^{n,-} - q^{n,+} \right) \right] \phi^+ \right) ds = 0
\end{aligned}
$$

Step 5. (local to global integration)

Now it is finally time to assemble the complete form by integrating over all elements $\int_\Omega \cdot dx = \sum_e \int_{\Omega^e} \cdot dx$. The important realization is here that the **interior facets** appear twice, since the facet is shared by two faces. However, the **boundary facets** only appear once. So the integral over all facets can be written as $\int_{\partial\Omega} \cdot ds = \sum_{e \in \Omega} \sum_{f \in e} \int_{\partial\Omega^f} \cdot ds$ where $f \in e$ denotes the facets f in element e.

Let's make this more explicit by looking at the following figure: The left element in $x \in [0, 1]$ has one boundary facet at $x = 0$ and an internal facet at $x = 1$. The right element in $x \in [1, 2]$ has one internal facet at $x = 1$ and one boundary facet at $x = 2$. So we have one contribution for each boundary and two condtributions for the internal facet at $x = 1$.

```
fig = library.plot.DG_facet_integral_illustration()
```



We now want to replace the sum of all faces for each element $\sum_{e\in\Omega}\sum_{f\in e}$ with a sum over all faces $\sum_{f\in\Omega} = \sum_f$, therefore only looping over each facet once. This results in two contributions for each facet, as we need to add the contributions of the $+$ and $-$ side.

$$
\begin{aligned}
&\sum_e \int_{\Omega^e} \left(\frac{q^{n+1}-q^n}{\Delta t}\right)\phi^e\,dx - \sum_e \int_{\Omega^e} \nabla\phi^e\cdot(q^n\mathbf{v})\,dx \\
&+\sum_f \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2}\left(q^{n,+}+q^{n,-}\right)\mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2}\left(|\lambda|_{max}\right)\left(q^{n,-}-q^{n,+}\right)\right]\phi^+\right)dS \\
&+\sum_f \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2}\left(q^{n,-}+q^{n,+}\right)\mathbf{v}\cdot\mathbf{n}^- - \frac{1}{2}\left(|\lambda|_{max}\right)\left(q^{n,+}-q^{n,-}\right)\right]\phi^-\right)dS \\
&+\sum_f \int_{\partial\Omega^e_{bnd}} \left(\left[\frac{1}{2}\left(q^{n,+}+q^{n,G}\right)\mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2}\left(|\lambda|_{max}\right)\left(q^{n,G}-q^{n,+}\right)\right]\phi^+\right)ds = 0
\end{aligned}
$$

The final step is to realize that $n^+ = -n^-$, which allows us to rearrange term with the intent two combine the two contributions from the numerical flux into one expression.

$$
\begin{aligned}
&+\sum_f \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,-}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,-} - q^{n,+}\right) \right] \phi^+ \right) dS \\
&+\sum_f \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2} \left(q^{n,-} + q^{n,+}\right) \mathbf{v}\cdot\mathbf{n}^- - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,+} - q^{n,-}\right) \right] \phi^- \right) dS \\
\Leftrightarrow &+\sum_f \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,-}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,-} - q^{n,+}\right) \right] \phi^+ \right) dS \\
&+\sum_f \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,-}\right) \mathbf{v}\cdot(-1)\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) (-1)\left(q^{n,-} - q^{n,+}\right) \right] \phi^- \right) dS \\
\Leftrightarrow &+\sum_f \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,-}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,-} - q^{n,+}\right) \right] \phi^+ \right) dS \\
&-\sum_f \int_{\partial\Omega^e_{int}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,-}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,-} - q^{n,+}\right) \right] \phi^- \right) dS \\
\Leftrightarrow &\sum_f \int_{\partial\Omega^e_{int}} \left[\frac{1}{2} \left(q^{n,+} + q^{n,-}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,-} - q^{n,+}\right) \right] \underbrace{\left[\phi^+ - \phi^-\right]}_{\text{jump of } \phi} dS
\end{aligned}
$$

So the weak formulation reads

$$
\begin{aligned}
&\sum_e \int_{\Omega^e} \left(\frac{q^{n+1} - q^n}{\Delta t} \right) \phi^e \, dx - \sum_e \int_{\Omega^e} \nabla\phi^e \cdot \left(q^n \mathbf{v}\right) \, dx \\
&\sum_f \int_{\partial\Omega^e_{int}} \left[\frac{1}{2} \left(q^{n,+} + q^{n,-}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,-} - q^{n,+}\right) \right] \underbrace{\left[\phi^+ - \phi^-\right]}_{\text{jump of } \phi} dS \\
&+\sum_f \int_{\partial\Omega^e_{bnd}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,G}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,G} - q^{n,+}\right) \right] \phi^+ \right) ds = 0
\end{aligned}
$$

Step 6 (domain boundary integral in FenicsX)

The last step is an implementation detail of FenicX. In FenicX, the integral over the domain $\int_{\partial\Omega^e_{bnd}} \cdot \, ds$ does not allow the the $(q^{n,-}, q^{n,+})$ syntax, as there is no ambiguity at the outer boundaries. In the dof, there is only one value, namely the interior value $q^{n,+}$. Therefore FenicX rewrites the integral with $q^{n,+} \to q^n$ such that

$$
\begin{aligned}
&\sum_f \int_{\partial\Omega^e_{bnd}} \left(\left[\frac{1}{2} \left(q^{n,+} + q^{n,G}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,G} - q^{n,+}\right) \right] \phi^+ \right) ds = 0 \\
\to &\sum_f \int_{\partial\Omega^e_{bnd}} \left(\left[\frac{1}{2} \left(q^{n} + q^{n,G}\right) \mathbf{v}\cdot\mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max}\right) \left(q^{n,G} - q^{n}\right) \right] \phi^+ \right) ds = 0
\end{aligned}
$$

After all this effort, our weak form read:

Final weak form

.

$$\sum_e \int_{\Omega^e} \left(\frac{q^{n+1} - q^n}{\Delta t} \right) \phi^e \, dx - \sum_e \int_{\Omega^e} \nabla \phi^e \cdot (q^n \mathbf{v}) \, dx$$
$$\sum_f \int_{\partial \Omega^e_{int}} \left[\frac{1}{2} \left(q^{n,+} + q^{n,-} \right) \mathbf{v} \cdot \mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max} \right) \left(q^{n,-} - q^{n,+} \right) \right] \underbrace{[\phi^+ - \phi^-]}_{\text{jump of } \phi} dS$$
$$\sum_f \int_{\partial \Omega^e_{bnd}} \left(\left[\frac{1}{2} \left(q^n + q^{n,G} \right) \mathbf{v} \cdot \mathbf{n}^+ - \frac{1}{2} \left(|\lambda|_{max} \right) \left(q^{n,G} - q^n \right) \right] \phi^+ \right) ds = 0$$

which is exactly the form that we will implement in the following.

5 Implementation

We first start by creating a mesh

```
from dolfinx import mesh

number_elements_x = 100
number_elements_y = 100
P0 = [0, 0]
P1 = [3, 3]
domain = mesh.create_rectangle(
    MPI.COMM_WORLD, [P0, P1], [number_elements_x, number_elements_y], cell_type=mesh.CellTyp
)
```

and we want to compare $DG(0)$ and $DG(1)$ function spaces.

```
def generate_functionspaces(domain):
    mesh_element_name = domain.topology.cell_name()
    """ type of mesh basix.ufl.element, e.g. "quadrilateral" """


    elem_DG0 = basix.ufl.element("DG", domain.topology.cell_name(), 0)
    space_DG0 = fem.functionspace(domain, elem_DG0)

    elem_DG1 = basix.ufl.element("DG", domain.topology.cell_name(), 1)
    space_DG1= fem.functionspace(domain, elem_DG1)
```

```
    elem_Vel = basix.ufl.element("DG", domain.topology.cell_name(), 1, shape=(domain.geometry
    space_Vel = fem.functionspace(domain, elem_Vel)

    return space_DG0, space_DG1, space_Vel
```

The weak form is derived in the following. For the current explicit method, we have a stability restriction on the time step size. This criterion is called the Courant-Friedrichs-Lewy number, or CFL number. In our case, the CFL restriction reads

$$CFL = 1/(2 * \text{ polynomial degree} + 1)/\text{physical dimension}$$

and

$$dt = \frac{CFL \, \text{characteristic length}}{(2 \, |\lambda|_{max})}$$

with the characteristic lenth $= min(dx, dy)$.

```
def weak_form_advection(functionspace, q_n, q_np, t, x):

    # advection velocity
    a = ufl.as_vector((2.0, 1.0))


    # facet normals
    n = ufl.FacetNormal(domain)

    # our integration measures over the inner boundaries, the domain boundaries and the whole
    # Note that we separate the domain boundaries in order to potentially apply different bou
    # on each side
    dS = ufl.Measure("dS", domain=domain)
    facet_tags = library.helpers.generate_facets_tags(domain, P0, P1)
    ds = ufl.Measure("ds", domain=domain, subdomain_data=facet_tags)
    dx = ufl.dx


    # time step size estimation
    basis_function_degree = functionspace.ufl_element().degree
    N = number_elements_x * number_elements_y
    dim = 2
    CFL = 1/(2*basis_function_degree+1)/dim
```

```
    max_abs_a = abs(np.array(a, dtype=float)).max()
    characteristic_length = 1/max(number_elements_x, number_elements_y)
    _dt = CFL *characteristic_length / max_abs_a / 2
    dt = dolfinx.fem.Constant(domain, dolfinx.default_scalar_type(_dt))

    # implicit/explicit switch
    q = q_n
    q_extrapolation = q_n

    test_q = ufl.TestFunction(functionspace)
    trial_q = ufl.TrialFunction(functionspace)

    # numerical Lax-Friedrichs flux
    propagation_speed = ufl.dot(a, n("+"))
    propagation_speed_bnd = ufl.dot(a, n)

    numerical_flux = lambda qp, qm, a, n , propagation_speed: (
        ufl.dot(0.5 * (a*qp + a*qm), n)
            - 0.5 * propagation_speed * (qm - qp)
        )

    # weak formulation
    weak_form =  test_q * (trial_q-q)/dt * dx
    weak_form += - ufl.dot(ufl.nabla_grad(test_q), (a * q)) * dx
    weak_form += ufl.dot((test_q("+") - test_q("-")), numerical_flux(q("+"), q("-"), a, n("+"
    weak_form += ufl.dot((test_q), numerical_flux(q, q_extrapolation, a, n, propagation_speed

    weak_form_lhs = fem.form(ufl.lhs(weak_form))
    weak_form_rhs = fem.form(ufl.rhs(weak_form))

    return weak_form_lhs, weak_form_rhs, dt
```

We now initialize the solver

```
def prepare_solver(weak_form_lhs, weak_form_rhs):

    A = petsc.create_matrix(weak_form_lhs)
    b = petsc.create_vector(weak_form_rhs)

    solver = PETSc.KSP().create(domain.comm)
    solver.setOperators(A)
    solver.setType(PETSc.KSP.Type.BCGS)
```

```
    preconditioner = solver.getPC()
    preconditioner.setType(PETSc.PC.Type.JACOBI)
    return solver, A, b
```

And run our time loop where we solve the solution for each time step.

Note that the limiter removes spurious osscillations for advective problems when solved with higher order methods. For $DG(0)$, no limiter is necessary. The details of limiters are beyond the scope of this class. However, we need to be aware that any polynomial degree > 0 might lead to problems when not addressed properly.

```
def solve_time_loop(name: str, weak_form_function, functionspace, initial_condition, end_time
    t = fem.Constant(domain, dolfinx.default_scalar_type(0.0))
    x = ufl.SpatialCoordinate(domain)

    q_n = fem.Function(functionspace, name=r'$q^n$')
    q_np1 = fem.Function(functionspace, name=r'$q^{n+1}$')
    weak_form_lhs, weak_form_rhs, dt = weak_form_function(functionspace, q_n, q_np1, t, x)

    solver, A, b = prepare_solver(weak_form_lhs, weak_form_rhs)
    A = petsc.create_matrix(weak_form_lhs)
    b = petsc.create_vector(weak_form_rhs)
    solver = PETSc.KSP().create(domain.comm)
    solver.setOperators(A)
    solver.setType(PETSc.KSP.Type.BCGS)
    preconditioner = solver.getPC()
    preconditioner.setType(PETSc.PC.Type.JACOBI)


    # Initial conditions
    q_n.interpolate(initial_condition)
    q_np1.interpolate(initial_condition)

    num_timesteps = int(end_time/dt.value)

    # VTK writer
    os.makedirs(output_path, exist_ok=True)
    vtk_file_abs_path_name = os.path.join(output_path, f"{name}.pvd")
    vtk_writer = dolfinx.io.VTKFile(
        domain.comm, vtk_file_abs_path_name, "w+"
)
```

```
    # We only need the following for plotting of the velocity as a vector field
    velocity_field = fem.Function(space_plot_vel, name=r'$vel$')
    velocity_field.interpolate(expr_vel(float(t.value)))

    vtk_writer.write_function([q_np1, velocity_field], t=0.0)
    n_snapshots = 30
    timeline_snapshots = np.linspace(0, num_timesteps, n_snapshots, dtype=int)




    progress = tqdm.tqdm(desc="Setup " + name + ", solving PDE", total=num_timesteps)
    if limiter:
        limiter = library.limiter.VertexBasedLimiter(functionspace, alpha=0.0)


    for i in range(num_timesteps):
        progress.update(1)
        q_n.interpolate(q_np1)


        A.zeroEntries()
        petsc.assemble_matrix(A,weak_form_lhs)
        A.assemble()
        with b.localForm() as loc:
            loc.set(0)
        petsc.assemble_vector(b,weak_form_rhs)

        b.ghostUpdate(addv=PETSc.InsertMode.ADD_VALUES, mode=PETSc.ScatterMode.REVERSE)
        solver.solve(b, q_np1.x.petsc_vec)
        if limiter:
            limiter.apply(q_np1)

        q_np1.x.scatter_forward()
        t.value += dt.value

        if i in timeline_snapshots:
            velocity_field.interpolate(expr_vel(float(t.value)))
            vtk_writer.write_function([q_np1, velocity_field], t=t.value)
```

```
    progress.close()
    return q_np1
```

6 A DG(0)/DG(1) minimal example

```
def ic_q(x):
    R = 0.15
    r = np.sqrt((x[0] - 0.7)**2 + (x[1] - 0.7)**2)
    return np.where(r <= R, 2., 1.)
```

```
space_DG0, space_DG1, space_Vel = generate_functionspaces(domain)
expr_vel = lambda t: lambda x: np.stack((2.0 * np.ones(x.shape[1]), 1.0 * np.ones(x.shape[1])
q_DG0 = solve_time_loop(name="advection DG(0)", weak_form_function=weak_form_advection, funct
q_DG1 = solve_time_loop(name="advection DG(1)", weak_form_function=weak_form_advection, funct
```

```
Setup advection DG(0), solving PDE: 100%|
Setup advection DG(1), solving PDE: 100%|
```

```
q_0 = fem.Function(space_DG0, name=r'$q^n$')
q_0.interpolate(ic_q)
library.plot.scalar_fields(
    functions=[q_0, q_DG0],
    titles=["t=0", "t=1"],
    figurepath=output_dir,
    figurename='test_DG0',
    show_edges=False,
)

q_0 = fem.Function(space_DG1, name=r'$q^n$')
q_0.interpolate(ic_q)
library.plot.scalar_fields(
    functions=[q_0, q_DG1],
    titles=["t=0", "t=1"],
    figurepath=output_dir,
    figurename='test_DG1',
    show_edges=False,
)
```

```
2025-04-23 16:35:14.287 (  29.517s) [    14FC86D6C740]vtkXOpenGLRenderWindow.:1416  WARN| bac

image saved in /home/is086873/CMM/ex01/test_DG0.png
image saved in /home/is086873/CMM/ex01/test_DG1.png
```

```
library.plot.display_image(image_path=os.path.join(output_dir, 'test_DG0.png'))
```

```
<IPython.core.display.HTML object>
```

```
library.plot.display_image(image_path=os.path.join(output_dir, 'test_DG1.png'))
```

```
<IPython.core.display.HTML object>
```

7 A passive tracer in a transient velocity field

We now want to compute a slightly more complex case by replacing the static velocity field $v(t,x) = (2,1)^T$ with a transient and space dependent velocity field

$$\mathbf{v} = \begin{cases} + \begin{pmatrix} -2(y-1.5) \\ 2(x-1.5) \end{pmatrix} & t \leq 0.5 \\ - \begin{pmatrix} -2(y-1.5) \\ 2(x-1.5) \end{pmatrix} & t > 0.5 \end{cases}$$

```
def weak_form_advection_transient(functionspace, q_n, q_np1, t, x):

    # advection velocity
    sign = -(t-0.5)/abs(t-0.5)
    a = ufl.as_vector((-sign*2*(x[1]-1.5), sign*2*(x[0]-1.5)))


    # facet normals
    n = ufl.FacetNormal(domain)

    # our integration measures over the inner boundaries, the domain boundaries and the whole
    # Note that we separate the domain boundaries in order to potentially apply different bou
    # on each side
    dS = ufl.Measure("dS", domain=domain)
    facet_tags = library.helpers.generate_facets_tags(domain, P0, P1)
```

```
    ds = ufl.Measure("ds", domain=domain, subdomain_data=facet_tags)
    dx = ufl.dx


    # time step size estimation
    basis_function_degree = functionspace.ufl_element().degree
    N = number_elements_x * number_elements_y
    dim = 2
    CFL = 1/(2*basis_function_degree+1)/dim

    a_x = fem.Function(functionspace)
    a_x.interpolate(fem.Expression(a[0], functionspace.element.interpolation_points()))
    a_y = fem.Function(functionspace)
    a_y.interpolate(fem.Expression(a[1], functionspace.element.interpolation_points()))
    max_abs_a = max(abs(a_x.x.array).max(), abs(a_y.x.array).max())

    characteristic_length = 1/max(number_elements_x, number_elements_y)
    _dt = CFL *characteristic_length / max_abs_a / 2
    dt = dolfinx.fem.Constant(domain, dolfinx.default_scalar_type(_dt))

    # implicit/explicit switch
    q = q_n
    q_extrapolation = q_n

    test_q = ufl.TestFunction(functionspace)
    trial_q = ufl.TrialFunction(functionspace)

    # numerical Lax-Friedrichs flux
    propagation_speed = abs(ufl.dot(a, n("+")))
    propagation_speed_bnd = abs(ufl.dot(a, n))

    numerical_flux = lambda qp, qm, a, n , propagation_speed: (
        ufl.dot(0.5 * (a*qp + a*qm), n)
            - 0.5 * propagation_speed * (qm - qp)
        )

    # weak formulation
    weak_form =  test_q * (trial_q-q)/dt * dx
    weak_form += - ufl.dot(ufl.nabla_grad(test_q), (a * q)) * dx
    weak_form += ufl.dot((test_q("+") - test_q("-")), numerical_flux(q("+"), q("-"), a, n("+
    weak_form += ufl.dot((test_q), numerical_flux(q, q_extrapolation, a, n, propagation_speed
```

```
    weak_form_lhs = fem.form(ufl.lhs(weak_form))
    weak_form_rhs = fem.form(ufl.rhs(weak_form))

    return weak_form_lhs, weak_form_rhs, dt
```

```
space_DG0, space_DG1, space_Vel = generate_functionspaces(domain)
expr_vel = lambda t: lambda x: np.stack(((t-0.5)/abs(t-0.5)*2*(x[1]-1.5), -(t-0.5)/abs(t-0.5)
q_DG0 = solve_time_loop(name="advection DG(0)", weak_form_function=weak_form_advection_transi
q_DG1 = solve_time_loop(name="advection DG(1)", weak_form_function=weak_form_advection_transi
```

```
Setup advection DG(0), solving PDE: 100%|
Setup advection DG(1), solving PDE: 100%|
```

```
q_0 = fem.Function(space_DG0, name=r'$q^n$')
q_0.interpolate(ic_q)
library.plot.scalar_fields(
    functions=[q_0, q_DG0],
    titles=["t=0", "t=1"],
    figurepath=output_dir,
    figurename='passivetracer_DG0',
    show_edges=False,
)

q_0 = fem.Function(space_DG1, name=r'$q^n$')
q_0.interpolate(ic_q)
library.plot.scalar_fields(
    functions=[q_0, q_DG1],
    titles=["t=0", "t=1"],
    figurepath=output_dir,
    figurename='passivetracer_DG1',
    show_edges=False,
)
```

```
image saved in /home/is086873/CMM/ex01/passivetracer_DG0.png
image saved in /home/is086873/CMM/ex01/passivetracer_DG1.png
```

```
library.plot.display_image(image_path=os.path.join(output_dir, 'passivetracer_DG0.png'))
```

```
<IPython.core.display.HTML object>
```

```
library.plot.display_image(image_path=os.path.join(output_dir, 'passivetracer_DG1.png'))
```

```
<IPython.core.display.HTML object>
```

Comparing the two plots, we can draw two important conclusions:

1. The $DG(0)$ implementation is highly diffusive. This is due to the stabilizatoin in the local Lax-Friedrich/Rusanov numerical flux implementation. Upwinding would yield superior results
2. The $DG(1)$ implementation violates the bounds of our initial condition, $q \in [1, 2]$. As we have a pure advection equation without source (production) term, this behavior is unphysical! Usually, one would now implement so called *limiters* to control this problem. However, this is content of other courses focusing on the numerical details of such numerical methods.

8 Verification

Based on the notebook 'My first FenicsX program', you should now verify the error of the $DG(0)$ and $DG(1)$ method by compting the L2-error. Feel free to borrow as much code as possible from the previous notebook.

For the given velocity field and the particular initial condition, the analytical solution is given by $q^{analytical}(t = 1, x) = q(t = 0, x)$.

STUDENT TODO
Implement the verification

```
# Implement your code here
```

```
def test_L2_error_implementation(error_L2_DG0, error_L2_DG1):
    if abs(error_L2_DG0-0.21908372090991204) < 10**(-3):
        print("DG(0) L2 error test passed")
    else:
        print("DG(0) L2 error test failed")
        assert False
    if abs(error_L2_DG1-0.05223104872875855) < 10**(-3):
        print("DG(1) L2 error test passed")
    else:
        print("DG(1) L2 error test failed")
        assert False
test_L2_error_implementation(error_L2_DG0, error_L2_DG1)
```

9 Streamlines and Pathlines

STUDENT TODO
Visualize the streamlines and pathlines in Paraview and insert an image for each into this notebook

```
#streamline plot
library.plot.display_image(image_path=None)
```

```
No image path provided.
```

```
#pathline plot
library.plot.display_image(image_path=None)
```

```
No image path provided.
```

10 Prepare for your presentation

During the presentation, be prepared to answer the following questions/tasks. We will ask 1-2 questions per person.

1. What was this homework about?
2. State the general transport equations and explain the role of each term
3. Derive a particular part from the weak formulation of the unsteady advection equation for DG(1). E.g.
    1. Compute the weak form for a single element
    2. What is ambiguous at the facets and gives rise to the usage of a numerical flux function?
    3. What is the weak enforcement of boundary conditions? Where does this happen in the implementation of our weak form? Can we use it for Dirichlet and Neumann boundary conditions?
4. Which terms of the weak formulation drops out if we restrict ourselves to DG(0) / CG(1)?
5. How do you need to change the weak formulation if we want to introduce an inflow boundary condition with $q|_{\partial\Omega^{\text{left}}} = q_{in} = 2$ on the left side of the domain.

6. Create a streamline plot in paraview and explain what streamlines are
7. Create a pathline plot in Paraview and explain what pathlines are