Modern Wave Propagation - Discontinuous Galerkin & Julia

Low-order Schemes

Vikas Kurapati, Marc Marot-Lassauzaie, David Schneller

2025-04-30

On this problem sheet, we aim to implement and test a first-order DG scheme.

1 Advection Equation

The first step is to set up the equation we want to solve. For this sheet we implement the three advection equation as described in the lecture notes.
First implement the physical flux and initial conditions in `equations/advection.jl`. A `yaml` file containing the configuration for the advection equations already exists. Later to add new equations you will need to write that as well. In `equations/advection.jl` please write the following two functions

- `get_initial_values`
- `evaluate_flux`

Bear in mind the following sizes:

```
size(cell_dof) = (order*order, ndof)
size(flux)     = (2*order*order, ndof)
```

For now $order = 1$, and $ndof = 3$.
For convenience there is a shortcut to access elements of the `dofs` vector by their pyhsical name, e.g:

```
s = AdvectionShortcuts()
celldofs[s.\rho 1]
```

Note: Julia allows you to use unicode literals in the source code. The code we provide uses ρ. On some machines this can lead to problems, if this happens to you replace the greek letter with `rho`.
We use the initial conditions:

$$\begin{aligned} \rho_1(x, y, t = 0) &= \sin(2\pi(x + y)), \\ \rho_2(x, y, t = 0) &= \sin(2\pi y), \\ \rho_3(x, y, t = 0) &= 1. \end{aligned} \tag{1}$$

The last initial condition is useful for testing: It should remain 1 during the entire simulation. For the other two solutions, you can compare with the analytical solution.
Note: The numerical solution is not going to look exactly like the analytical solution, as the numerical scheme is very diffusive and smears the solution.
We assume periodic boundary conditions for this problem. This is already the default.

2 Numerical Scheme

In the lecture notes we learned that for a first-order scheme the integral over the cell is zero and we only need to calculate the integrals over the boundaries of each cell.
So, in the kernels folder you will need to implement the Rusanov (Local Lax-Friedrichs) flux and the face integrals.
Open `kernels/surface.jl` and fill in the following two functions:

1. `evaluate_face_integral`
2. `rusanov`

   The numerical flux has the following size:

   ```
   size(numericalflux) = (order, ndof)
   ```

In TerraDG the time-stepping scheme is already implemented. When computing the update for the time step, the multiplication with Δt and the inverse mass matrix is already included. This corresponds to another factor of $\frac{1}{\text{volume}}$ for our first low-order scheme. Note that the area-factor coming from the face integrals is **not** included!

3 Visualisation

Visualise the solution to the problem in paraview from $t = 0.0$ to 1.0 for all three components (ρ_1, ρ_2, ρ_3). It should look similar to the initial condition but some dissipation is expected, as the scheme is not yet very accurate. You can compare with fig. 1.
Add a comment to your git submission (see instructions below) and describe the behaviour of the solution for the three initial conditions. Also explain, whether the results are as you would expect them.

4 Error

In addition to the initial condition, also implement the analytical solution in the `get_initial_values` function. When deriving the analytical solution, keep in mind that we use periodic boundary conditions. Then the error writer will automatically compute the error of your solution in multiple norms and print it to the terminal after the simulation.
Add a comment to your git submission containing the error table for (at least) three different mesh sizes. How does the error change? (Hint: Consider the convergence order of the method.) You can see an example of the error in table 1.

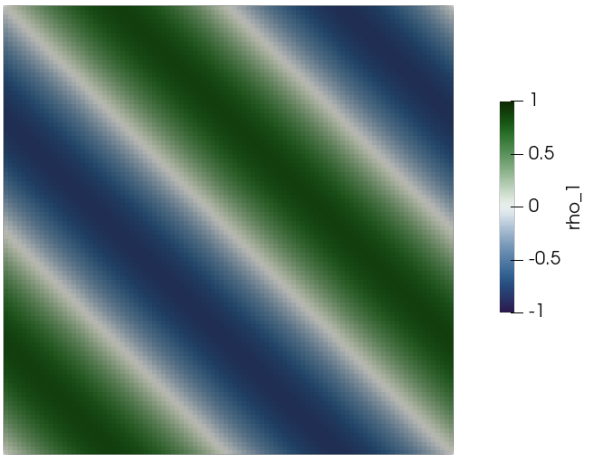

Figure 1: Solution ρ_1 at $t = 1\,\mathrm{s}$ obtained with first order scheme.

Table 1: Error for 100 elements in each direction and a Courant number of 0.5 at time $t = 1.0$

Variable	L_1	L_2	L_∞
ρ_1	1.140180e-01	1.266838e-01	1.791579e-01
ρ_2	8.762168e-02	9.731180e-02	1.375873e-01
ρ_3	0.000000e+00	0.000000e+00	0.000000e+00

5 Handing in

Create a private fork of the TerraDG repository, and add all three of us (userids: `krenzland`, `ga65jeg`, `davidschneller`) with maintainer access. Upload your solution to this fork.
Create a tag for the commit that contains the version that you want to hand in. You can follow `https://git-scm.com/book/en/v2/Git-Basics-Tagging` for a detailed walk-through.

Here's the gist: The idea behind tags is simple. They are some form of annotated branch. This means, you can create a tag by running `git tag -a worksheet-1 -m "Comments"`. Please **include all comments** to your submission in the comments. Otherwise they can easily get lost.
Note that you need to explicitly push the tag to origin (Gitlab) by running `git push origin --tags`. You can check if this was successful by checking on GitLab. The url for this is `https://gitlab.lrz.de/username/reponame/-/tags`, where you need to replace `username/reponame` by the path to the repository.