

# Series 2



Advanced Numerical Methods for  
CSE

**Last edited:** November 21, 2019

**Due date:** December 8, 2019

Template codes are available on the course's webpage at <https://metaphor.ethz.ch/x/2019/hs/401-4671-00L/>.

## IMPORTANT INFORMATION

**In order to take the exam for the course, you must submit a solution to this exercise sheet.** After the deadline, a **mandatory code review** will take place, in which you will individually discuss your solution with one of the assistants. Details will follow.

**To take the final exam**, a partial solution to the exercise sheet is enough – a complete solution to **at least one** of the exercises, which doesn't need to be perfect, suffices. This interview is not an exam; you will not be assigned a numerical grade. The purpose of the interview is to ensure that you've either solved the exercise yourself, or solved it in a small group and you are familiar with every aspect of the code.

A **bonus of +0.25** to your exam grade will be awarded for handing in and defending either a complete, but not necessarily perfect, solution to both exercises; or a complete, correct solution to one.

Theory questions (2a-2d) will not be discussed, although they will be necessary to solve the exercise.

## Exercise 1 Finite volume and discontinuous Galerkin methods for Euler equations in 1D

Euler equations in 1D are written as:

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x = \mathbf{0}, \quad (1a)$$

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho v \\ E \end{pmatrix}, \quad \mathbf{f}(\mathbf{u}) = \begin{pmatrix} \rho v \\ \rho v^2 + p \\ (E + p)v \end{pmatrix}, \quad (1b)$$

where the density  $\rho$ , velocity  $v$  and energy  $E$  are unknown, and the pressure  $p$  is determined by the equation of state:

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho v^2, \quad \text{for } \gamma = 7/5 = 1.4. \quad (2)$$

Additionally, the speed of sound  $c$  and enthalpy  $H$  are given by:

$$c = \sqrt{\frac{\gamma p}{\rho}}, \quad H = \frac{E + p}{\rho}. \quad (3)$$

In this exercise, we implement finite volume and discontinuous Galerkin methods for the 1D Euler equations (1) on a domain  $\Omega$  discretized by a uniform grid  $\mathcal{T}_h$ , with the cell-centers denoted by  $x_i$ , the interfaces denoted by  $x_{i+1/2}$  and the cell size  $h$ .

**Hint:** Please read the README.md which comes with the code. It will briefly walk you through the individual parts of the exercise.

**Hint:** The subproblems are ordered, and it is recommended that you solve them in the stated order.

**Hint:** The file config.json is used to configure the simulation. The path the this file has been hard-coded. Depending on your setup you might need to change edit src/ancse/config.cpp to used the right path for your system. This is likely the case if you're using VS2019. The path should be relative to the executable.

**Hint:** You can also create new config, e.g. filename.json, and run with this configuration as

```
./fvm_euler path_relative_to_exec/filename.json
```

**Hint:** If you run into compilation issues due to library filesystem missing in snapshot\_writer.cpp, your compiler is very likely not fully-C++17 complying; updating it should fix the issue.

### Finite volume method

The semi-discrete formulation of FVM is

$$\frac{d}{dt} \mathbf{U}_i + \frac{1}{\Delta x} (\mathbf{F}_{i+1/2} - \mathbf{F}_{i-1/2}) = 0 \quad (4)$$

where  $\mathbf{U}_i$  is the approximate cell-average of  $\mathbf{u}$  in cell  $i$  and  $\mathbf{F}_{i+1/2}$  is the numerical flux through the interface  $i + 1/2$ . We consider two-point numerical fluxes given by

$$\mathbf{F}_{i+1/2} = \mathbf{F}(\mathbf{U}_{i+1/2}^-, \mathbf{U}_{i+1/2}^+), \quad (5)$$

where the traces  $\mathbf{U}_{i+1/2}^-$ ,  $\mathbf{U}_{i+1/2}^+$  are approximate values of  $\mathbf{u}(x_{i+1/2}, t)$  to the left and to the right of the interface, respectively.

### 1a)

Templates: `include/ancse/model.hpp`, `src/ancse/model.cpp`.

Implement the Euler equations model.

**Hint:** The class `Burgers` is given as an example for the Burgers' equation. You can add additional routines to the interfacing class `Model`, if needed.

**Hint:** Write tests to check the accuracy of the Euler equations model in `tests/test_model.cpp`

### 1b)

Templates: `include/ancse/limiters.hpp`, `include/ancse/numerical_flux.hpp`

Implement the following numerical fluxes:

- Rusanov's
- Lax-Friedrichs
- Roe
- HLL
- HLLC

**Hint:** The class `CentralFlux` is given as an example for implementing the central flux.

**Hint:** Implement tests in `tests/test_numerical_flux.cpp`.

### 1c)

Template: `include/ancse/reconstruction.hpp`

Implement piecewise linear reconstruction of the trace values  $\mathbf{U}_{i+1/2}^\pm$  based on

- conservative variables  $\mathbf{u}_{\text{cons}} = (\rho, \rho v, E)$
- primitive variables  $\mathbf{u}_{\text{prim}} = (\rho, v, p)$

with the following slope-limiters:

- minmod
- superbee
- monotized central

**Hint:** The class `PWConstantReconstruction` is given as an example for implementing piecewise constant reconstruction.

**Hint:** Consider implementing this using a template class which accepts the slope-limiter as a template parameter.

**Hint:** Implement tests in `tests/test_reconstruction.cpp`.

### 1d)

Template: `include/ancse/fvm_rate_of_change.hpp`

Complete the loop that applies your fluxes and numerical reconstructions.

### 1e)

Templates: `include/ancse/runge_kutta.hpp`, `src/ancse/runge_kutta.cpp`

Implement the second-order strong stability preserving Runge-Kutta (SSP2) scheme.

**Hint:** The class `ForwardEuler` is given as example for implementing forward Euler.

### 1f)

Templates: `include/ancse/cfl_condition.hpp`, `src/ancse/cfl_condition.cpp`

Implement the CFL condition. This should be a new class that derives from `CFLCondition`, which implements the computation of a CFL-satisfying  $dt$  for a generic problem.

**Hint:** Implement tests in `tests/test_cfl_condition.cpp`.

## 1g)

To enable selecting the different schemes at run time we need to implement factories for each component.

Start by registering your implementation of Euler equations model, see `src/ancse/model.cpp`.

Next, register the numerical flux and reconstruction. You'll find an example of how to do this in `src/ancse/fvm_rate_of_change.cpp`.

Finally, register your implementation of SSP2, see `src/ancse/runge_kutta.cpp`. Note, follow the example of how it is done for `ForwardEuler`.

## 1h)

You can try the following with your FVM code:

- Consider Sod's shock tube problem on a domain  $\Omega := (0, 1)$  with outflow boundary conditions and initial discontinuity at  $x = 0.5$ . The initial states to the left and to the right of the discontinuous interface are

$$\begin{pmatrix} \rho_L \\ v_L \\ p_L \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \quad \begin{pmatrix} \rho_R \\ v_R \\ p_R \end{pmatrix} = \begin{pmatrix} 0.125 \\ 0 \\ 0.1 \end{pmatrix}. \quad (6)$$

For details about this experiment, you can read:

- G. Sod, A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, *J. Comput. Phys.*, 27 (1978), pp. 1–31.  
URL: <https://www.archives-ouvertes.fr/hal-01635155/document>
- Section 14.13 of *Finite Volume Methods for Hyperbolic problems*, R. LeVeque, 2002.  
You can find this book through the course web page for AdvNumCSE.

Study the performance of the different numerical schemes for this experiment.

Is the piecewise linear reconstruction based on conservative variables better than the one based on primitive variables?

- Consider a “vacuum” problem on a domain  $\Omega := (0, 1)$  with outflow boundary conditions and initial discontinuity at  $x = 0.5$ . The initial states to the left and to the right of the discontinuous interface are

$$\begin{pmatrix} \rho_L \\ v_L \\ p_L \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix}, \quad \begin{pmatrix} \rho_R \\ v_R \\ p_R \end{pmatrix} = \begin{pmatrix} 1 \\ +2 \\ 1 \end{pmatrix}. \quad (7)$$

Does the numerical scheme with Roe's numerical flux work for this experiment?

**Hint:** When assessing the performance of a scheme, use a moderate number of cells, in the range of 10s to 100s.

**Hint:** Use piecewise linear reconstruction with minmod limiter, Rusanov's flux and SSP2 time-stepping on 2048 cells as the reference solution.

## Discontinuous Galerkin method

Let  $\mathbb{P}_p$  be the space of the polynomials with maximum degree  $p$ . Define a scalar *piecewise* polynomial space

$$W_h^p := \{w \in L^2(\Omega) : w|_K \in \mathbb{P}_p(K), K \in \mathcal{T}_h\}. \quad (8)$$

This means that a function  $w \in W_h^p$  is a polynomial in the cells and it is discontinuous across cell interfaces.

We can build a orthonormal basis for  $W_h^p$  using Legendre polynomials, which are given below on the domain  $[-1, 1]$  for  $p = 2$ :

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = \frac{1}{2}(3x^2 - 1).$$

The Legendre polynomials are orthogonal:

$$\int_{-1}^{+1} P_k P_l dx = \frac{2}{2k+1} \delta_{kl},$$

where  $\delta_{kl} = 1$ , if  $k = l$ , and  $\delta_{kl} = 0$ , if  $k \neq l$ . Transform  $P_k$  to the domain  $[0, 1]$  and define the normalized functions

$$\phi_k(x) := \sqrt{2k+1} P_k(2x-1) \chi_{[0,1]}(x), \text{ such that } \int_0^{+1} \phi_k \phi_l dx = \delta_{kl}.$$

Here  $\chi_{[0,1]}(x) = 1$ , if  $x \in [0, 1]$  and  $\chi_{[0,1]}(x) = 0$ , otherwise.

Given  $N$  grid cells, uniformly spaced with  $x_{i+1/2} - x_{i-1/2} = h$ , let  $\{\varphi_{i,j}\}$  denote the basis of  $W_h^p$ , i.e. a function  $w \in W_h^p$  can be written as

$$w(x) = \sum_{i=0}^N \sum_{j=0}^p w_{i,j} \varphi_{i,j}(x),$$

where we define the function

$$\varphi_{i,j}(x) := \frac{1}{\sqrt{h}} \phi_j \left( \frac{x - x_{i-1/2}}{h} \right). \quad (9)$$

Note,  $\varphi_{i,j}(x) = 0$ , if  $x \notin (x_{i-1/2}, x_{i+1/2})$ , i.e. the function  $\varphi_{i,j}$  has local support on  $(x_{i-1/2}, x_{i+1/2})$ .

For a given  $m \in \mathbb{N}$ , we can easily define a vector piecewise polynomial space

$$\mathbf{W}_h^p := [W_h^p]^m, \quad (10)$$

i.e. for a vector function  $\mathbf{w} \in \mathbf{W}_h^p$ , its components  $w_l \in W_h^p$  for  $l = 1, \dots, m$ . For 1D Euler equations  $m = 3$ .

We seek a solution of the form

$$\mathbf{u}_h(x, t) = \sum_{i=0}^N \sum_{j=0}^p U_{i,j}(t) \varphi_{i,j}(x),$$

with  $U_{i,j} : \mathbb{R} \rightarrow \mathbb{R}^m$  the coefficient of  $\varphi_{i,j}$  in the expansion of  $\mathbf{u}_h(\cdot, t)$ . Note that, for fixed  $t \geq 0$ ,  $\mathbf{u}_h(\cdot, t)$  can be uniquely identified with a vector  $(U_{i,j,l})_{l=1, j=0, i=1}^{l=m, j=p, i=N}$  of coefficients in  $\mathbb{R}^{m(p+1)N}$ .

To derive the DG formulation for the Euler equations (1), we take its inner product with a test function  $\mathbf{w}_h \in \mathbf{W}_h^p$ , integrate over an arbitrary cell  $K_i := (x_{i-1/2}, x_{i+1/2}) \in \mathcal{T}_h$  and use integration by parts:

$$\begin{aligned} \int_{K_i} (\mathbf{u}_h)_t \cdot \mathbf{w}_h \, dx + \int_{K_i} \mathbf{f}(\mathbf{u}_h)_x \cdot \mathbf{w}_h \, dx &= 0 \\ \frac{d}{dt} \left( \int_{K_i} \mathbf{u}_h \cdot \mathbf{w}_h \, dx \right) + (\mathbf{f}(\mathbf{u}) \cdot \mathbf{w}_h) \Big|_{x_{i+1/2}^-} - (\mathbf{f}(\mathbf{u}) \cdot \mathbf{w}_h) \Big|_{x_{i-1/2}^+} - \int_{K_i} \mathbf{f}(\mathbf{u}_h) \cdot (\mathbf{w}_h)_x \, dx &= 0. \end{aligned}$$

Here  $\mathbf{u}_h \in \mathbf{W}_h^p$  is the approximate solution and the flux through the interface  $i + 1/2$ ,  $\mathbf{f}(\mathbf{u}(x_{i+1/2}))$ , can be approximated with a numerical flux  $\mathbf{F}$  by

$$\mathbf{f}(\mathbf{u}(x_{i+1/2})) \approx \hat{\mathbf{f}}_{i+1/2} := \mathbf{F} \left( \mathbf{u}_h(x_{i+1/2}^-), \mathbf{u}_h(x_{i+1/2}^+) \right). \quad (11)$$

The semi-discrete formulation of the DG method is

$$\frac{d}{dt} \left( \int_{K_i} \mathbf{u}_h \cdot \mathbf{w}_h \, dx \right) = \hat{\mathbf{f}}_{i-1/2} \cdot \mathbf{w}_h \Big|_{x_{i-1/2}^+} - \hat{\mathbf{f}}_{i+1/2} \cdot \mathbf{w}_h \Big|_{x_{i+1/2}^-} + \int_{K_i} \mathbf{f}(\mathbf{u}_h) \cdot (\mathbf{w}_h)_x \, dx. \quad (12)$$

**Remark:** If  $\mathbf{W}_0$  is used, i.e. the space of piecewise constant functions, the volume integral on the right-hand side in (12) vanishes as  $(\mathbf{w}_h)_x = \mathbf{0}$ , and dividing by the cell size  $h$  we re-obtain the FVM semi-discrete formulation (4).

**Remark:** By construction of the DG method, it is easy to evaluate the values of  $\mathbf{u}$  to the left and to the right of cell interfaces for computing numerical fluxes. No reconstruction is needed!

**Remark:** For  $p \geq 1$ , we need to limit the DG solution coefficients corresponding to polynomials of degree greater than equal to 1. However, one can separate the evolution step and the limiting procedure. For example - if we use forward Euler time integrator to solve (12), then first we will evolve the solution coefficients from  $t_n$  to  $t_{n+1}$  and then perform the limiting.

**Remark:** Since  $\mathbf{u}_h(\cdot, t) \in \mathbf{W}_h^p$ , we have

$$\int_{K_i} \mathbf{u}_h(\cdot, t) \cdot \mathbf{w}_h \, dx = \int_{K_i} \sum_{k=0}^p U_{i,k}(t) \varphi_{i,k} \cdot \mathbf{w}_h \, dx.$$

Due to orthonormality of the basis polynomials, taking  $\mathbf{w}_h := \varphi_{i,j} \mathbf{e}_l$ , with  $\mathbf{e}_l \in \mathbb{R}^m$  the unit vector with  $(\mathbf{e}_l)_i = \delta_{li}$ , we obtain

$$\int_{K_i} \mathbf{u}_h \cdot \mathbf{w}_h \, dx = \sum_{k=0}^p U_{i,k,l}(t) \delta_{jk} = U_{i,j,l}(t)$$

**Remark:** For implementation, we re-arrange the vector of coefficients  $(U_{i,j,l})_{l=1,j=0,i=1}^{l=m,j=p,i=N} \in \mathbb{R}^{m(p+1)N}$  into a matrix  $\mathbf{U} \in \mathbb{R}^{m(p+1) \times N}$  of  $m(p+1)$  rows and  $N$  columns, with entries

$$U_{i, j+(p+1)l} := U_{i+1,j,l+1}, \text{ for } j = 0, \dots, p, \quad l = 0, \dots, m-1, \quad i = 0, \dots, N-1.$$

This particular arrangement is used in the initialization step, which has already been provided, of the DG solver.

## 1i)

Template: `src/ancse/polynomial_basis.cpp`.

Implement the basis functions  $\phi_k$ , with maximum degree 2, as described above.

`PolynomialBasis::operator()` takes as input one point  $\xi \in [0, 1]$ , and return a vector of  $p+1$  components containing  $\{\phi_k(\xi)\}_{k=0}^p$ , scaled by an appropriate scaling factor, which you can assume is already contained in `PolynomialBasis::scaling_factor`.

Do the same in function `PolynomialBasis::deriv` for the spatial derivatives  $\partial_\xi \phi_k(\xi)$ .

## 1j)

Template: `src/ancse/dg_handler.cpp`.

Implement the function `build_sol`, which builds the solution at a given point in a specified cell.

That is, `DGHandler::build_sol` receives the vector of coefficients  $U_{i,\cdot} \in \mathbb{R}^{m(p+1)}$  of the approximate solution  $\mathbf{u}_h$  in cell  $K_i$ , and a point  $x$  in the cell  $K_i$ , and returns  $\mathbf{u}_h(x) = \sum_{k=0}^p U_{i,k} \varphi_{i,k}(x)$ . Note, the physical point  $x$  should be converted to the corresponding reference point  $\xi \in [0, 1]$ , because  $\varphi_{i,k}(x)$  is computed through  $\phi_k$  using equation 9.

Implement the function `build_cell_avg`, which builds the cell averages in the given cells.

That is, `DGHandler::build_cell_avg` produces a matrix  $\bar{u}_h \in \mathbb{R}^{m \times N}$ , where

$$(\bar{u}_h)_{l,i} = \frac{1}{h} \int_{K_i} (u_h)_l(x) \, dx,$$

where  $(u_h)_l$  refers to the  $l$ -th component of  $\mathbf{u}_h$ .



**Hint:** What relationship is there between  $\frac{1}{h} \int_{K_i} (u_h)_l(x) dx$  and the coefficient corresponding to  $\varphi_{i,0}$  in  $\mathbf{u}_h|_{K_i}$ ?

1k)

Templates: `include/ancse/cfl_condition.hpp`, `src/ancse/cfl_condition.cpp`

Implement the CFL condition using the cell averages  $\bar{u}_h \in \mathbb{R}^{m \times N}$ . This should be a new class that derives from `CFLCondition`, which implements the computation of a CFL-satisfying  $dt$  for a generic problem.

**Hint:** Implement tests in `tests/test_cfl_condition.cpp`.

1l)

Template: `src/ancse/dg_rate_of_change.cpp`.

Implement the functions `eval_numerical_flux` and `eval_volume_integral`, which compute the numerical flux term and volume integral on the right-hand side in the formulation (12), respectively.

Register numerical fluxes in `make_dg_rate_of_change`, as you have already done for FVM.

**Hint:** You can test your implementation without any limiting for  $p = 0$  with forward Euler, it should give the same results as the first-order FVM.

## DG Limiting

Without delving into too many details, we will provide a recipe to do limiting for the scalar case, which can then be applied component-wise for systems:

Define

$$\bar{u}_{i,0} := u_{i,0} \varphi_{i,0}, \quad (13a)$$

$$\bar{u}_i^- := + \sum_{j=1}^p u_{i,j} \varphi_{i,j}(x_{i+1/2}^-), \quad \bar{u}_i^+ := - \sum_{j=1}^p u_{i,j} \varphi_{i,j}(x_{i-1/2}^+). \quad (13b)$$

This implies

$$\begin{aligned} u_h(x_{i+1/2}^-) &= \bar{u}_{i,0} + \bar{u}_i^-, \\ u_h(x_{i-1/2}^+) &= \bar{u}_{i,0} - \bar{u}_i^+. \end{aligned}$$

Additionally, define

$$\Delta_+ \bar{u}_{i,0} := \bar{u}_{i+1,0} - \bar{u}_{i,0}, \quad \Delta_- \bar{u}_{i,0} := \bar{u}_{i,0} - \bar{u}_{i-1,0}. \quad (14)$$

Using the so-called van Leer limiter:

$$\tilde{u}_i^- := g(\bar{u}_i^-, \Delta_- \bar{u}_{i,0}, \Delta_+ \bar{u}_{i,0}), \quad \tilde{u}_i^+ := g(\bar{u}_i^+, \Delta_- \bar{u}_{i,0}, \Delta_+ \bar{u}_{i,0}), \quad (15)$$

with the minmod function

$$g(a_1, \dots, a_n) := \begin{cases} s \min_{1 \leq l \leq n} |a_l|, & s = \text{sgn}(a_1) = \dots = \text{sgn}(a_n), \\ 0, & \text{otherwise.} \end{cases} \quad (16)$$

Finally, we recompute the coefficients post limiting  $\tilde{u}_{i,j}$  from  $\tilde{u}_i^-$  and  $\tilde{u}_i^+$ . You can easily check that  $\tilde{u}_{i,j}$  can be uniquely determined for  $p = 1, 2$ .

### 1m)

Template: `src/ancse/dg_handler.cpp`.

Implement the function `build_split_sol`, which returns  $\bar{\mathbf{u}}_{i,0}$ ,  $\bar{\mathbf{u}}_i^+$  and  $\bar{\mathbf{u}}_i^-$ , c.f. (13).

Implement the function `build_limit_coeffs`, which computes the coefficients  $\tilde{U}_{i,j}$  after limiting.

### 1n)

Template: `src/ancse/dg_limiting.cpp`.

Implement the DG limiting procedure for the conserved variables.

**Hint:** You should test your implementation of DG limiting with Burgers' equation. Run two Riemann problems, one that leads to a shock and another that leads to a rarefaction.

Repeat the Sod shock tube test described previously with your DG code.

## Exercise 2 Solving the compressible Euler equations on an unstructured mesh for computing flow past an airfoil

In this exercise, you will simulate the steady state airflow over a NACA airfoil. The simulation requires you to solve the Euler equations in two dimensions numerically on an unstructured grid.

**Hint:** Please read README.md carefully as it contains a lot of important information about the code.

We start with the following definitions:

- $t \in \mathbb{R}^+$  is the time,
- $\mathbf{x} \in \mathbb{R}^2$  is the (Eulerian) position,
- $\mathbf{u}(\mathbf{x}, t) = (u_1(\mathbf{x}, t), u_2(\mathbf{x}, t)) \in \mathbb{R}^2$  is the velocity at  $(\mathbf{x}, t)$ ,
- $\rho(\mathbf{x}, t) \in \mathbb{R}^+$  is the density at  $(\mathbf{x}, t)$ ,
- $p(\mathbf{x}, t) \in \mathbb{R}$  is the pressure at  $(\mathbf{x}, t)$ ,
- $E(\mathbf{x}, t) \in \mathbb{R}$  is the total energy at  $(\mathbf{x}, t)$ .

The Euler equations are then given as:

$$\begin{aligned}\rho_t + \nabla \cdot (\rho \mathbf{u}) &= 0 \\ (\rho \mathbf{u})_t + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + p \mathbf{I}) &= 0 \\ E_t + \nabla \cdot ((E + p) \mathbf{u}) &= 0.\end{aligned}\tag{17}$$

Here

$$\mathbf{u} \otimes \mathbf{u} := \begin{pmatrix} u_1 u_1 & u_1 u_2 \\ u_2 u_1 & u_2 u_2 \end{pmatrix}$$

The total energy of the system is given by

$$E = \frac{1}{2} \rho \mathbf{u} \cdot \mathbf{u} + \rho e.$$

For the rest of this exercise, we assume that the internal energy  $e$  is given as

$$e = \frac{p}{\gamma - 1},$$

where  $\gamma$  is the adiabatic constant. We set

$$\gamma = 1.4.$$

To make notation easier, we will consider the general form of (17) given as

$$\mathbf{U}_t + \nabla_x \cdot \mathbf{F}(\mathbf{U}) = 0 \quad (18)$$

where  $\mathbf{U} : \mathbb{R}^d \times \mathbb{R}^+ \rightarrow \mathbb{R}^N$  is the vector of conserved variables, and  $\mathbf{F} : \mathbb{R}^N \rightarrow \mathbb{R}^{N \times d}$  is the flux tensor. We set

$$\nabla_x \cdot \mathbf{F}(\mathbf{U}(\mathbf{x}, t)) := \begin{pmatrix} \frac{\partial}{\partial x_1} F_{1,1}(\mathbf{U}(\mathbf{x}, t)) + \dots + \frac{\partial}{\partial x_d} F_{1,d}(\mathbf{U}(\mathbf{x}, t)) \\ \frac{\partial}{\partial x_1} F_{2,1}(\mathbf{U}(\mathbf{x}, t)) + \dots + \frac{\partial}{\partial x_d} F_{2,d}(\mathbf{U}(\mathbf{x}, t)) \\ \vdots \\ \frac{\partial}{\partial x_1} F_{N,1}(\mathbf{U}(\mathbf{x}, t)) + \dots + \frac{\partial}{\partial x_d} F_{N,d}(\mathbf{U}(\mathbf{x}, t)) \end{pmatrix},$$

where

$$\mathbf{F}(\mathbf{U}) = \begin{pmatrix} F_1(\mathbf{U}) \\ \vdots \\ F_N(\mathbf{U}) \end{pmatrix} = \begin{pmatrix} F_{1,1}(\mathbf{U}) & F_{1,2}(\mathbf{U}) & \dots & F_{1,d}(\mathbf{U}) \\ \vdots & \vdots & \vdots & \vdots \\ F_{N,1}(\mathbf{U}) & F_{N,2}(\mathbf{U}) & \dots & F_{N,d}(\mathbf{U}) \end{pmatrix}.$$

In the case of two dimensional Euler,  $d = 2$  and  $N = 4$ , and

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ E \end{pmatrix} =: \begin{pmatrix} \rho \\ m_x \\ m_y \\ E \end{pmatrix} \quad (19)$$

where we assume  $\rho > 0$ .

## 2a)

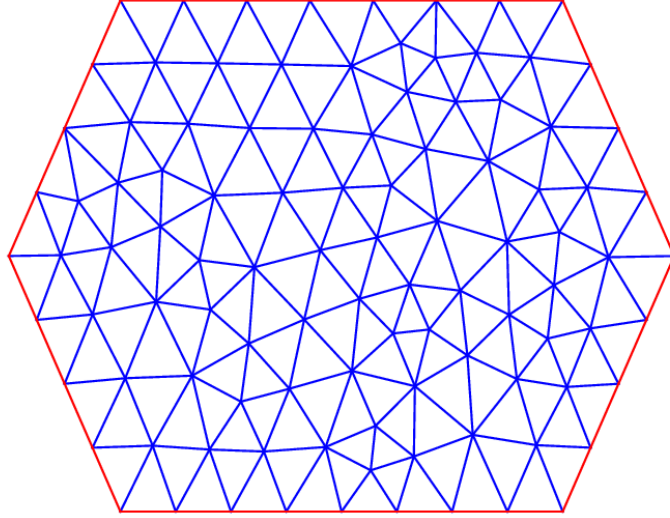
Write the compressible Euler equations (17) on the form (18). What is the flux function  $\mathbf{F}$  for the compressible Euler equations?

**Solution:** We set

$$\mathbf{F} \begin{pmatrix} \rho \\ m_x \\ m_y \\ E \end{pmatrix} = \begin{pmatrix} m_x & m_y \\ \frac{m_x m_x}{\rho} + p & \frac{m_x m_y}{\rho} \\ \frac{\rho m_y m_x}{\rho} & \frac{m_y m_y}{\rho} + p \\ \frac{m_x(E+p)}{\rho} & \frac{m_y(E+p)}{\rho} \end{pmatrix} = \begin{pmatrix} m_x & m_y \\ \rho u_1^2 + p & \rho u_1 u_2 \\ \rho u_2 u_1 & \rho u_2^2 + p \\ u_1(E+p) & u_2(E+p) \end{pmatrix}. \quad (20)$$

□

We consider a polygonal domain  $\Omega$  in  $\mathbb{R}^2$  and assume we have a triangulation  $\mathcal{T} = \{K_i\}_{i=1}^M$ . See Figure 1 for an illustration.



**Figure 1:** Example of our domain  $\Omega$ .

2b)

Let  $1 \leq i \leq M$  and let  $U : \Omega \times \mathbb{R}^+ \rightarrow \mathbb{R}^N$ , and define

$$\bar{\mathbf{U}}_i(t) := \frac{1}{|K|} \int_K \mathbf{U}(\mathbf{x}, t) \, d\mathbf{x}.$$

Furthermore, for  $\mathbf{n} \in \mathbb{R}^2$ , we denote:

$$\mathbf{F}(\mathbf{U}) \cdot \mathbf{n} := \begin{pmatrix} F_1(\mathbf{U}) \cdot \mathbf{n} \\ \vdots \\ F_N(\mathbf{U}) \cdot \mathbf{n} \end{pmatrix}.$$

This represents the flux through a surface with normal vector  $\mathbf{n}$ .

Given triangle  $K_i \in \mathcal{T}$ , and a neighbouring triangle  $K_j$  (i.e.  $K_i \cap K_j$  is a segment), we denote their common edge as  $e_{i,j}$ . For a given triangle  $K_i$ , we denote its three edges as  $\{e_{i,j_k}\}_{k=1}^3$ , with  $K_{j_k}$  its three neighbours.

**Remark:** Technically this does not work if  $K_i$  has one edge on  $\partial\Omega$ , since  $K_{j_k}$  may not be a triangle in  $\mathcal{T}$ . However, in this case the notation is not useful because you do not want to treat all edges the same because boundary conditions apply to some of the edges. Therefore, either ignore the issue or set  $K_{j_k} = \Omega^c$  (for the sole purpose of giving those edges a name).

Assume  $\mathbf{U}$  is a smooth solution of (18). Fix  $t^n, \Delta t \geq 0$ . Show that

$$\bar{\mathbf{U}}_i(t^n + \Delta t) - \bar{\mathbf{U}}_i(t^n) = -\frac{1}{|K_i|} \sum_{k=1}^3 \int_{t^n}^{t^n + \Delta t} \int_{e_{i,j_k}} \mathbf{F}(\mathbf{U}) \cdot \mathbf{n}_{i,j_k} dS dt, \quad (21)$$

where  $\{e_{i,j_k}\}_{k=1}^3$  are the edges of  $K_i$ , and  $\{\mathbf{n}_{i,j_k}\}_{k=1}^3$  are the outward-pointing unit vectors normal to the triangle along edge  $e_{i,j_k}$ .

**Solution:** Let us start from equation (18) in  $K_i$ :

$$\mathbf{U}_t + \nabla_x \cdot \mathbf{F}(\mathbf{U}) = 0 \quad (22)$$

We integrate in space (for domain  $K_i$ ) and in time (for  $t \in [t^n, t^n + \Delta t]$ ), and divide by the area of  $K_i$ ,  $|K_i|$ . Since  $\mathbf{U}$  is smooth on  $K_i$  compact, we have integrability and thus

$$0 = \frac{1}{|K_i|} \int_{K_i} \int_{t^n}^{t^n + \Delta t} \frac{\partial}{\partial t} \mathbf{U}(\mathbf{x}, t) dt d\mathbf{x} + \frac{1}{|K_i|} \int_{K_i} \int_{t^n}^{t^n + \Delta t} \nabla_x \cdot \mathbf{F}(\mathbf{U}(\mathbf{x}, t)) dt d\mathbf{x} \quad (23)$$

$$= \frac{1}{|K_i|} \int_{K_i} (\mathbf{U}(\mathbf{x}, t^n + \Delta t) - \mathbf{U}(\mathbf{x}, t^n)) dt d\mathbf{x} + \frac{1}{|K_i|} \int_{t^n}^{t^n + \Delta t} \int_{K_i} \nabla_x \cdot \mathbf{F}(\mathbf{U}(\mathbf{x}, t)) d\mathbf{x} dt \quad (24)$$

$$= \bar{\mathbf{U}}_i(t^n + \Delta t) - \bar{\mathbf{U}}_i(t^n) + \frac{1}{|K_i|} \int_{t^n}^{t^n + \Delta t} \int_{\partial K_i} \mathbf{F}(\mathbf{U}(\mathbf{x}, t)) \cdot \mathbf{n} dS dt \quad (25)$$

where in the last step, we have used the definition of  $\bar{\mathbf{U}}$ , Gauss' theorem, and  $\mathbf{n}$  is the outward-pointing unit vector normal to  $\partial K_i$ .

We conclude by noting that, since  $K_i$  is a triangle, its boundary can be written as the union of its three edges  $\partial K_i = \bigcup_{k=1}^3 e_{i,j_k}$ , along each of which  $\mathbf{n} \equiv \mathbf{n}_{i,j_k}$ . Since this union is disjoint (other than sets of measure 0), we can conclude

$$0 = \bar{\mathbf{U}}_i(t^n + \Delta t) - \bar{\mathbf{U}}_i(t^n) + \frac{1}{|K_i|} \sum_{k=1}^3 \int_{t^n}^{t^n + \Delta t} \int_{e_{i,j_k}} \mathbf{F}(\mathbf{U}) \cdot \mathbf{n}_{i,j_k} dS dt$$

□

**2c)**

Fix an edge with unit outward normal  $\mathbf{n}$  and a transverse unit vector  $\tau$  such that  $\mathbf{n} \cdot \tau = 0$ . Verify that

$$\mathbf{F}(\mathbf{U}) \cdot \mathbf{n} = \begin{pmatrix} \rho(\mathbf{u} \cdot \mathbf{n}) \\ \rho(\mathbf{u} \cdot \mathbf{n})u_1 + p n_1 \\ \rho(\mathbf{u} \cdot \mathbf{n})u_2 + p n_2 \\ (\mathbf{u} \cdot \mathbf{n})(E + p) \end{pmatrix}. \quad (26)$$

This looks remarkably similar to any one of the two components of the flux tensor. But not close enough to allow us to use the approximate Riemann solvers we know from structured grids.

However, for a fixed edge we could choose to perform coordinate transform  $\mathbf{x} \mapsto (\xi, \zeta)$  such that  $\mathbf{x} = \xi \mathbf{n} + \zeta \boldsymbol{\tau}$ . Define  $u_\perp = \mathbf{u} \cdot \mathbf{n}$  and  $u_\parallel = \mathbf{u} \cdot \boldsymbol{\tau}$ , then the resulting system of PDEs is

$$\partial_t \rho + \partial_\xi \rho u_\perp + \partial_\zeta \rho u_\parallel = 0 \quad (27)$$

$$\partial_t \rho u_\perp + \partial_\xi (\rho u_\perp^2 + p) + \partial_\zeta \rho u_\parallel u_\perp = 0 \quad (28)$$

$$\partial_t \rho u_\parallel + \partial_\xi \rho u_\perp u_\parallel + \partial_\zeta (\rho u_\parallel^2 + p) = 0 \quad (29)$$

$$\partial_t E + \partial_\xi u_\perp (E + p) + \partial_\zeta u_\parallel (E + p) = 0 \quad (30)$$

Note that we choose to compute the rate of change of the density, normal component of the momentum, the transverse component of the momentum and the energy. Furthermore, note that flux in  $\xi$ -direction is

$$\mathbf{f}(\mathbf{U}) = \begin{pmatrix} \rho u_\perp \\ \rho u_\perp^2 + p \\ \rho u_\perp u_\parallel \\ u_\perp (E + p) \end{pmatrix}. \quad (31)$$

Use these ideas to approximate the contribution of edge  $e$

$$\int_t^{t+\Delta t} \int_{e_{i,j}} \mathbf{F}(\mathbf{U}) \cdot \mathbf{n}_{i,j} dS dt. \quad (32)$$

to the net flux by the HLLC approximate Riemann solver known from two-dimensional FVM on *structured* grids.

Remember, the approximate Riemann solver will compute the flux of momentum normal and transverse to the interface. You need to convert this into the flux of momentum in x- and y-directions.

**Hint:** You can take a shortcut and approximate with Rusanov's numerical flux instead.

## 2d)

For  $\mathbf{f} : \mathbb{R}^N \rightarrow \mathbb{R}^N$ , let  $\{\lambda_i(\mathbf{f}, \mathbf{U})\}_{i=1}^N$  be the eigenvalues of the Jacobian of  $\mathbf{f}(\mathbf{U})$ . Define

$$\lambda_{\max}(\mathbf{f}, \mathbf{U}) := \max_i |\lambda_i(\mathbf{f}, \mathbf{U})|.$$

Show that for flux  $\mathbf{F}(\mathbf{U}) \cdot \mathbf{n}$ , it holds that:

$$\lambda_{\max}(\mathbf{F} \cdot \mathbf{n}, \mathbf{U}) = |\mathbf{u} \cdot \mathbf{n}| + \sqrt{\frac{\gamma p}{\rho}}. \quad (33)$$

Use this to derive the following CFL-condition:

$$\Delta t < C_{CFL} \frac{\Delta x}{\max_i \lambda_{max}(\mathbf{F} \cdot \mathbf{n}, \mathbf{U}_i)}, \quad (34)$$

for all unit-vectors  $\mathbf{n}$ . Here  $\Delta x$  is the small inradius and  $C_{CFL} = 0.45$ .

**Hint:** Use the trick in subproblem **2c)** to reduce the problem to computing the eigenvalues of the Jacobian of  $F_{,1}$ . Which you have either seen in the lecture or you can look them up online.

## 2e)

Implement two types of flux boundary conditions:

1. outflow flux boundary conditions
2. reflective (or solid wall) flux boundary conditions.

Flux boundary conditions specify a flux at the boundary of the domain. Therefore, they do not need any ghost-cells. Instead one needs to identify all interfaces at the boundary of the domain. For these faces one must not attempt to compute the numerical flux, instead one evaluates a flux boundary condition.

Consider edge  $e$  of triangle  $K$ . The outflow boundary conditions are given by

$$\int_t^{t+\Delta t} \int_e \mathbf{F}(\mathbf{U}) \cdot \mathbf{n} \, dS dt = \Delta t |e| \mathbf{F}(\mathbf{U}) \cdot \mathbf{n}. \quad (35)$$

The reflective (or wall) flux boundary conditions are

$$\int_t^{t+\Delta t} \int_e \mathbf{F}(\mathbf{U}) \cdot \mathbf{n} \, dS dt = \Delta t \mathcal{F}(\mathbf{U}, \mathbf{U}^*) \quad (36)$$

where  $\mathbf{U} = (\rho, \rho \mathbf{u}, E)$  are the cell-averages of the conserved variables in triangle  $K$  and

$$\mathbf{U}^* = (\rho, -\rho(\mathbf{u} \cdot \mathbf{n})\mathbf{n} + \rho(\mathbf{u} \cdot \boldsymbol{\tau})\boldsymbol{\tau}, E). \quad (37)$$

Here  $\mathcal{F}$  denotes the numerical flux derived in **2c)**.

**Hint:** The class `Mesh` has a member `getBoundaryType` which tells you which type of boundary condition must be applied to a given edge.

## 2f)

With the result you've derived so far, implement first order FVM for the Euler equations on an unstructured grid. Carefully debug your code at this stage, before going on to the second order



FVM. Check convergence rates using the script `compute_convergence.py` and visualize the solution with `plot_on_mesh.py`.

**Hint:** See README.md for more details.

The final step in implementing the second order FVM is to compute the trace values of  $\mathbf{U}$  at the interfaces.

Analogous to the REA algorithm we will compute the reconstruction of a quantity  $q$ , such as  $\rho$ ,  $u_1$  or  $p$ , whose value  $\{Q_i\}_i$  is given at the cell-centers  $\mathbf{x}_i$ . We do this with:

$$q_i(\mathbf{x}) = Q_i + (\nabla q(\mathbf{x}_i)) \cdot (\mathbf{x} - \mathbf{x}_i). \quad (38)$$

There are two problems. The first one is that this corresponds to piecewise linear reconstruction without a slope-limiter. This was not stable for 1D problems and won't be stable in this context. The second problem is how to compute the gradient  $\nabla q$ .

2g)

Approximate the right hand side of

$$\nabla q(\mathbf{x}_i) \approx \frac{1}{|K_i|} \int_{K_i} \nabla q \, d\mathbf{x} \quad (39)$$

by Gauss' theorem (applied to the vector field  $q\mathbf{c}$ , for certain constant  $\mathbf{c} \in \mathbb{R}^2$ ), and the mid-point rule for the resulting integrals over the boundary.

**Solution:** Let  $\mathbf{c} \in \mathbb{R}^2$  a constant vector, and let us apply the divergence theorem to the function  $\mathbf{G} := q\mathbf{c}$ :

$$\int_{K_i} \nabla \cdot \mathbf{G} \, d\mathbf{x} = \int_{\partial K_i} \mathbf{G} \cdot \mathbf{n} \, dS \quad (40)$$

$$\Leftrightarrow \int_{K_i} (\nabla q \cdot \mathbf{c}) \, d\mathbf{x} = \int_{\partial K_i} (\mathbf{c} \cdot \mathbf{n}) q \, dS \quad (41)$$

$$= \sum_{k=1}^3 \int_{e_{i,j_k}} (\mathbf{c} \cdot \mathbf{n}_{i,j_k}) q \, dS \quad (42)$$

$$\approx \sum_{k=1}^3 |e_{i,j_k}| (\mathbf{c} \cdot \mathbf{n}_{i,j_k}) q(\mathbf{x}_{i,j_k}) \quad (43)$$

where in the last step we have used the mid-point rule for each edge  $e_{i,j_k}$ ; we denote the mid-point of each such edge by  $\mathbf{x}_{i,j_k}$ . We denote by  $|e_{i,j_k}|$  the length of the edge. Applying the argument above for all  $\mathbf{c} \in \{(1, 0), (0, 1)\}$ , we obtain (component-by-component):

$$\int_{K_i} \nabla q \, d\mathbf{x} = \sum_{k=1}^3 |e_{i,jk}| \mathbf{n}_{i,jk} q(\mathbf{x}_{i,jk})$$

## 2h)

Fix  $i \in \{1, \dots, M\}$ . For each edge  $e_{i,j}$  of triangle  $K_i$ , shared with triangle  $K_j$ , let us denote its mid-point by  $\mathbf{x}_{i,j}$ . One can compute a limited slope as follows

$$s_{i,j} = \xi(\nabla q(\mathbf{x}_i) \cdot \Delta \mathbf{x}_{i,j}, \nabla q(\mathbf{x}_j) \cdot \Delta \mathbf{x}_{i,j}) \quad (44)$$

with  $\Delta \mathbf{x}_{i,j} = \mathbf{x}_{i,j} - \mathbf{x}_i$  and a slope limiter  $\xi$  of your choice. The reconstructed values are then

$$q_{i,j} = Q_i + s_{i,j} \quad \text{and} \quad q_{j,i} = Q_j + s_{j,i}. \quad (45)$$

Implement piecewise linear reconstruction of the primitive variables  $(\rho, u_1, u_2, p)$  by using the previously derived expressions.

## 2i)

First run a convergence test for your code. We do this by running a smooth vortex test case on a sequence of meshes.

The numerical experiment has already been implemented in `vortex.cpp`.

**Hint:** You can find more information in `README.md`.

## 2j)

Simulate and visualize the steady state airflow over an airfoil. Try different Mach numbers and angles of attack, by modifying the appropriate lines in `naca_airfoil.cpp`.

**Hint:** You will find more information in `README.md`.