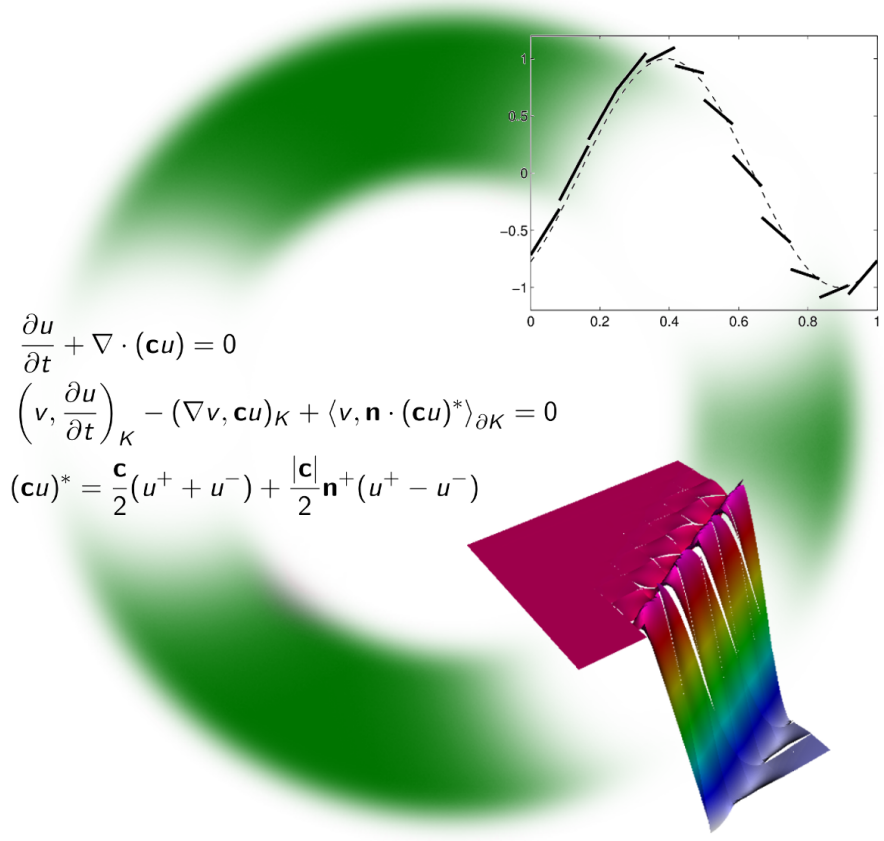
Lehrstuhl für Numerische Mechanik

Technische Universität München

Discontinuous Galerkin Methods for Computational Mechanics



$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{c}u) = 0$$

$$\left(v, \frac{\partial u}{\partial t}\right)_K - (\nabla v, \mathbf{c}u)_K + \langle v, \mathbf{n} \cdot (\mathbf{c}u)^* \rangle_{\partial K} = 0$$

$$(\mathbf{c}u)^* = \frac{\mathbf{c}}{2}(u^+ + u^-) + \frac{|\mathbf{c}|}{2}\mathbf{n}^+(u^+ - u^-)$$

Dr. Martin Kronbichler

Version: SS 2016

TEACHER VERSION

compiled on July 15, 2016

How to use these lecture notes
These lecture notes are designed for active participation during the lectures. It is highly recommended to actively participate in the lectures. For further studies, we recommend to also study the course literature, in particular the main course book

J. S. Hesthaven, T. Warburton
Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications
Springer, 2008

The authors invite the readers to point out mistakes in the notes as well as to suggest possibilities for improvement or additions.

Lecture notes "Discontinuous Galerkin Methods for Computational Mechanics"

Dr. Martin Kronbichler

2. Auflage: SS 2016

Technische Universität München
Lehrstuhl für Numerische Mechanik
Prof. Dr.-Ing. W. A. Wall
Boltzmannstraße 15
D-85747 Garching b. München
Telefon: +49 (0) 89 - 289 - 15300
Fax: +49 (0) 89 - 289 - 15301
E-Mail: sekretariat@lnm.mw.tum.de
www: http://www.lnm.mw.tum.de

Contents

1 Introduction to Discontinuous Galerkin

1.1 Related discretization methods

Consider the generic transport equation:

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = g, \tag{1.1}$$

where $f(u)$ is a flux function, e.g. $f(u) = au$ for linear transport with wave speed a, and $g(x,t)$ is a possible forcing. Here, we consider the flow in the computational domain $\Omega = (0,1)$, but other flow examples are possible.
The equation is completed by an initial condition,

$$u(x,0) = u_0(x),$$

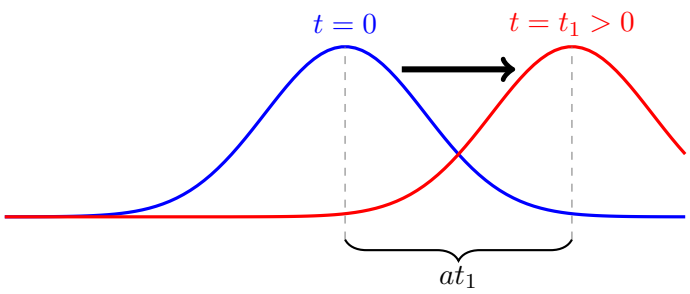
and a boundary condition on the inflow part of the domain,

$$u(0,t) = h_0(t) \quad \text{if } a(0,t) > 0 \quad \text{and/or} \quad u(1,t) = h_1(t) \quad \text{if } a(1,t) < 0.$$

The condition whether h_0 or h_1 are prescribed depends on the flow direction. In general, this can also change over time. For the equation of the type (1.1), a boundary condition is only set where information enters the domain. If a boundary condition would be set on the outflow part, the problem would not be well-posed longer, i.e., a non-physical boundary condition would be set, which can either lead to non-physical simulation results or even unstable computations. The scalar transport equation (1.1) is part of a larger problem class called *conservation laws*. Examples of the nonlinear form of this equation are discussed in Chapter 3.
The linear transport equation is particularly simple because it is possible to state the analytic solution explicitly,

$$u(x,t) = u_0(x - at),$$

i.e., the solution at time t and point x is given by the solution at the initial time some distance at away from x. This can be visualized as follows:

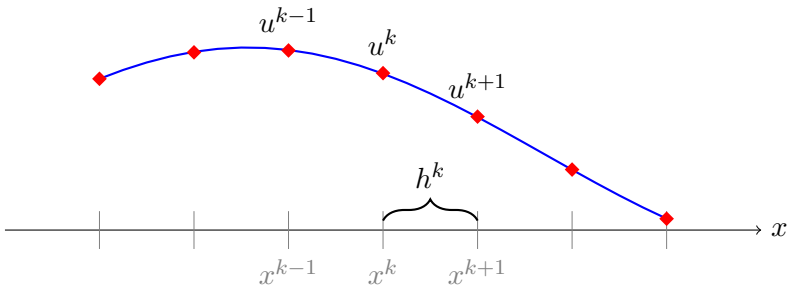


Next, we discuss a few methods for discretization of the transport equation in space and their respective pros and cons.

Finite differences

The idea of a finite difference method is to introduce a mesh consisting of points $x^1, x^2, \ldots, x^{k-1}, x^k, \ldots$ and compute approximations in these mesh points,

$$u(x^k, t) \approx u^k(t).$$



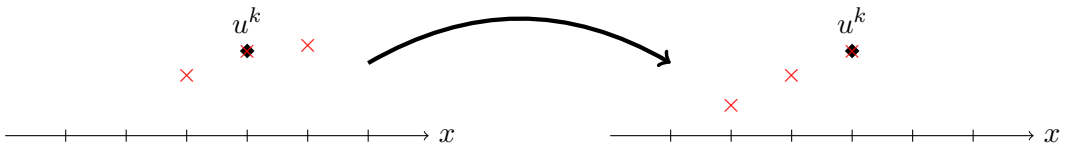
For representing the differential equation, the derivative $\frac{\partial}{\partial x}$ is replaced by a difference quotient on the mesh,

$$\frac{\mathrm{d}u^k}{\mathrm{d}t} + \frac{f(u^{k+1}) - f(u^{k-1})}{\underbrace{x^{k+1} - x^{k-1}}_{=2h \text{ for equidistant case}}} = g(x^k, t), \qquad \text{for } k = 1, 2, \ldots$$

Using the central differences involving the solution in three points u^{k-1}, u^k, u^{k+1} around x^k, a method that is second order accurate in space can be designed.

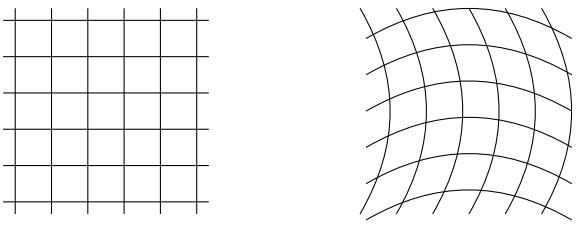
Advantages:

- Very simple and intuitive
- Computationally efficient, only stencil evaluation
- Can adapt difference stencil to problem nature, e.g. by upwinding



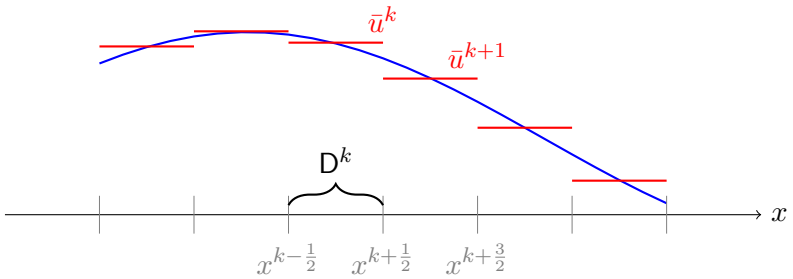
Disadvantages:

- Restricted to simple meshes (Cartesian or mapped Cartesian), difficult (or inaccurate) for complex geometries



Finite volumes

Approximate $u(x,t)$ by constants $\bar{u}_k = u_k(t)$ in volumes $\mathsf{D}^k = \left(x^{k-\frac{1}{2}}, x^{k+\frac{1}{2}}\right)$.



The resulting scheme reads as follows:

$$h^k \frac{\mathrm{d}\bar{u}^k}{\mathrm{d}t} + f^{k+\frac{1}{2}} - f^{k-\frac{1}{2}} = h^k \bar{g}^k,$$

which is derived by integration by parts on the volume average of the spatial derivative term,

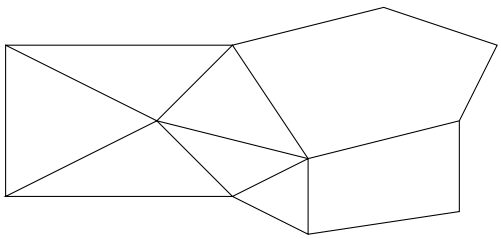
$$\int_{\mathsf{D}^k} 1 \cdot \frac{\partial}{\partial x} f(u) \mathrm{d}x = -\int_{\mathsf{D}^k} \left(\frac{\partial}{\partial x} 1\right) f \mathrm{d}x + \Big[f(u)\Big]_{x^{k-\frac{1}{2}}}^{x^{k+\frac{1}{2}}}.$$

The evaluation of the flux $f^{k+\frac{1}{2}}$ needs information at the point $x^{k+\frac{1}{2}}$. One possibility is:

$$u^{k+\frac{1}{2}} = \frac{\bar{u}^{k+1} + \bar{u}^k}{2}, \qquad f^{k+\frac{1}{2}} = f(u^{k+\frac{1}{2}}).$$

Advantages:

- Allows for arbitrary volume shapes because the reconstruction is local to each volume boundary. It only needs to access volume information in the neighborhood of $\bar{u}^k$ (e.g. $\bar{u}^{k-1}, \bar{u}^{k+1}$). This makes the method ideally suited for complex geometries.



- Many possibilities for applying finite-difference-inspired upwinding for implementing the flux $f^{k+\frac{1}{2}}$, allowing to construct schemes with different properties.
- Adaptive meshes easily possible.

Disadvantages:

- Higher order methods are difficult to realize: They need neighbors farther than one layer away, which requires more structured meshes (similar to finite difference meshes) and defeats the geometric flexibility.

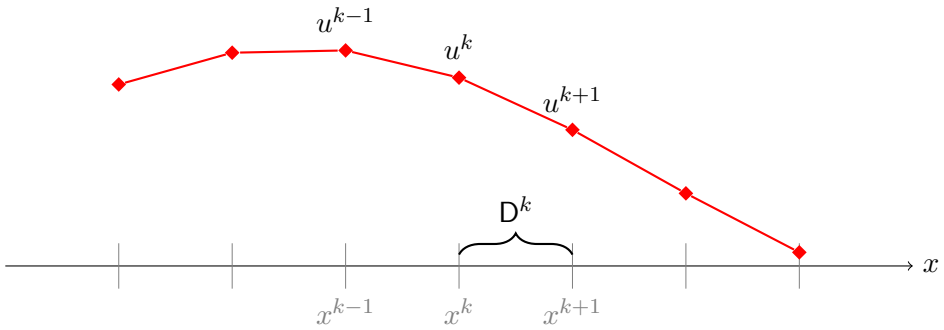
Finite volume methods on general unstructured meshes are typically low order accurate with error behavior as $\mathcal{O}(h^1)$ or $\mathcal{O}(h^2)$ at best.

Finite elements

Finite element methods rely on a mesh of non-overlapping elements $\mathcal{D}^k$ similar to finite volume methods. In 1D, the solution is approximated by a polynomial, in the simplest case by linear functions:

$$u_h(x) = \sum_{i=0}^{1} u_i^k \ell_i^k(x),$$

where ℓ_i is a Lagrange polynomial centered in the "nodes" of the element.



For the linear functions given above, the admissible solution looks as follows:

$$u_h(x) = u_0^k \frac{x^{k+1} - x}{x^{k+1} - x^k} + u_1^k \frac{x - x^k}{x^{k+1} - x^k},$$

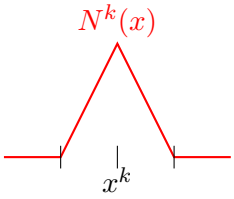
putting a linear interpolation between the two node values u_0^k and u_1^k. Furthermore, the finite element requires the solution to be globally continuous, i.e.,

$$u_0^k = u_1^{k-1} = u^k,$$

which means that the left value on D^k must be the same as the right value on element D^{k-1}, simply denoted by the variable u^k.

This gives the global version of the admissible solution:

$$u_h(x) = \sum_{k=1}^{K} u^k N^k(x),$$



where N^k is the global basis function (hat function).

In the finite element method, it is solved for the following weak form:

$$\int_\Omega \left(\frac{\partial u_h}{\partial t} + \frac{\partial f(u_h)}{\partial x} - g_h \right) N^j(x) \mathrm{d}x = 0, \qquad \text{for } j = 1, \ldots, K.$$

This gives rise to a matrix system of the form

$$\mathcal{M} \frac{\mathrm{d}\boldsymbol{u}_h}{\mathrm{d}t} + \mathcal{S}\boldsymbol{f}_h = \mathcal{M}\boldsymbol{g}_h,$$

where $\boldsymbol{f}_h$ and $\boldsymbol{g}_h$ are vectors collecting the node values of the flux $f(u_h)$ and the right hand side g.

Advantages:

- High order methods are simple to realize (more nodes *inside* the elements, higher order Lagrange polynomials ℓ_i^k)

- Geometric flexibility
- Adaptive meshes easily possible

Disadvantages:

- Globally defined basis functions ⇒ implicit (matrix) system even for explicit time integration
- "Symmetric" basis functions not optimal for transport problems where information flow is strongly directional ⇒ needs suitable modifications, called stabilization (SUPG, GLS, ...), which can be non-trivial to derive.

1.2 Derivation of 1D discontinuous Galerkin schemes

The discontinuous Galerkin method aims to combine advantageous properties of finite element and finite volume methods in terms of capability of high order methods and usage of directional information. By construction on an arbitrary mesh, the method inherits the geometric flexibility of finite element and finite volume methods.

Reading instructions: Further information about the content of the remainder of this chapter can be found in chapter 2 of the course book [Hesthaven and Warburton, 2008].

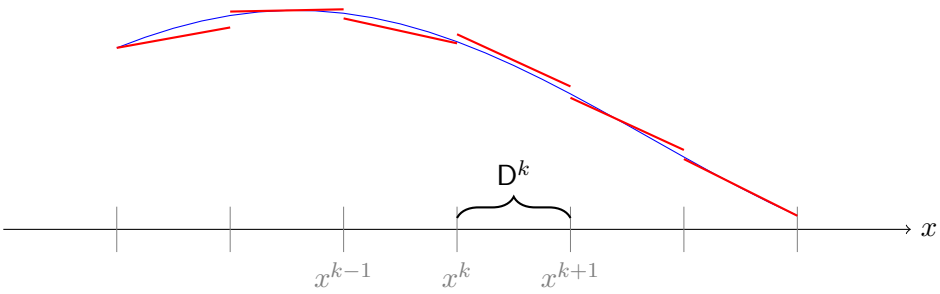
With the discontinuous Galerkin method, called DG-FEM, we divide the computational domain into K elements $\mathsf{D}^k, k = 1, \ldots, K$. On each element, we assume a solution expansion of the form

$$u_h^k(x,t) = \sum_{j=0}^{N} \ell_j^k(x) u_j^k(t), \tag{1.2}$$

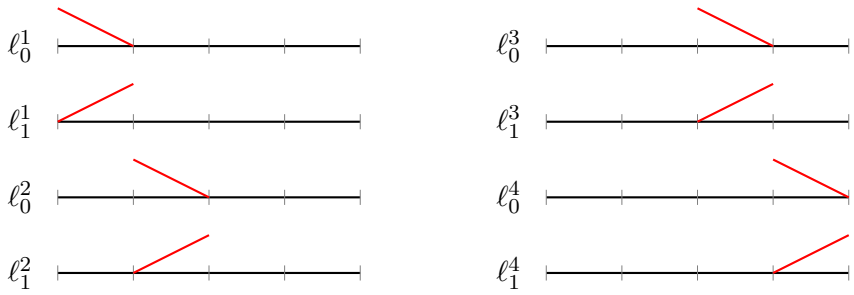
where the functions $\ell_j^k(x)$ are Lagrange polynomials up to degree N and $u_j^k(t)$ are the unknown coefficient values. In this chapter, we consider the case of linear polynomials with $N = 1$, i.e., the Lagrange polynomials are the usual linear shape functions known from finite elements. As opposed to the finite element method, we do not assume continuity of the solution space over element boundary. Instead, we choose *independent* solution functions on each element, duplicating the nodal values at interior element boundaries. For K elements, this gives the following vector of nodal unknowns:

$$\boldsymbol{u} = \left[u_0^1, u_1^1, u_0^2, u_1^2, \ldots, u_0^k, u_1^k, \ldots u_0^K, u_1^K\right].$$

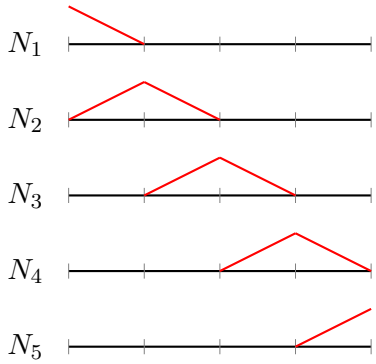
Let us visualize the construction of the tentative solution:



In terms of the contributing basis functions, each element has an independent set of functions without continuity to the neighbor,



This construction is opposed to the continuous finite element method where the same element subdivision would result in the picture



Since we approximate the solution with discontinuous functions, we need to be careful in the mathematical manipulations. As opposed to finite elements where we can pose the problem on the whole domain, we start with the formulation on one element D^k only in terms of the solution u_h^k on that element. We multiply by a test function, represented by ℓ_j^k and integrate the flux term $\int_{\mathsf{D}^k} f(u_h^k)\ell_j^k dx$ by parts. This gives the following weak form:

$$\int_{\mathsf{D}^k} \left(\frac{\partial u_h^k}{\partial t}\ell_j^k - f_h^k \frac{d\ell_j^k}{dx} - g\ell_j^k \right) dx = -\left[f_h^k \ell_j^k \right]_{x^k}^{x^{k+1}} = -\int_{\partial \mathsf{D}^k} \hat{\boldsymbol{n}} f_h^k \ell_j^k ds, \tag{1.3}$$

The term on the right hand side is the result of the partial integration in one dimension. The last term on the right hand side rewrites the boundary term as an boundary integral with outer unit normal vector $\hat{\boldsymbol{n}}$, which is $+1$ on the right boundary of D^k and -1 on the left boundary. This will be the form in 2D and 3D.
In DG-FEM, the boundary terms f_h^k are evaluated at x^k and x^{k+1}. The value $f_h^k(x^k)$ is expressed by the numerical flux $f^*(x^k) = f^*\left(u^{k-1}(x^k), u^k(x^k)\right)$, which for linear elements is $f^*(x^k) = f^*\left(u_1^{k-1}, u_0^k\right)$. We will detail the numerical flux term below. Its role is to connect the problems on each element in order to solve the global partial differential equation.

Derivation of a matrix form

Inserting the ansatz (1.2) into the weak form and performing the element integrals in the usual sense, we obtain the following matrix form on D^k:

$$\mathcal{M}^k \frac{\partial \boldsymbol{u}^k}{\partial t} - (S^k)^T \boldsymbol{f}^k = -\boldsymbol{f}^{k,*}.$$

In the above equation, we write $\boldsymbol{f}^k$ for the evaluation of the flux $f(u)$ for the nodal values. This is allowed for the simple form $f(u) = au$ with constant a. For non-constant coefficients, evaluating the equation using this concise form introduces a slight error because we only modify the node values to express a polynomial of degree N. If a is variable and u_h^k is a polynomial of degree N, the product au_h^k will no longer be a polynomial of degree N. Thus, the evaluation needs to be more careful in order to avoid so-called aliasing problems. We will discuss this issue later in the course. Moreover, we denote by $\boldsymbol{f}^{k,*}$ the numerical flux, to be specified below.
The element matrices involved in this form are:

Mass matrix

$$\mathcal{M}^k = \left[\int_{\mathsf{D}^k} \ell_i^k(x)\ell_j^k(x)dx\right]_{i,j} = \frac{h^k}{6}\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}$$

Advection matrix

$$\mathcal{S}^k = \left[\int_{\mathsf{D}^k} \ell_i^k(x)\frac{d\ell_j^k(x)}{dx}dx\right]_{i,j} = \frac{1}{2}\begin{bmatrix}-1 & 1\\ -1 & 1\end{bmatrix}$$

Before we discuss the matrix $\mathcal{F}$ in more detail, let us consider the global matrix problem. Since the solution space between the elements is independent (there are new sets of basis functions for each element), the global mass and advection matrices are simply a collection of the matrices $\mathcal{M}^k$ and $\mathcal{S}^k$, i.e., small 2×2 blocks, on the diagonal. For four elements, four elements of size $h = \frac{1}{K} = \frac{1}{4}$, the global mass and stiffness matrices are:

$$\mathcal{M} = \frac{h}{6}\begin{bmatrix}
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 2 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 2 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 2 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 2 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 2 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2
\end{bmatrix}, \qquad
\mathcal{S} = \frac{1}{2}\begin{bmatrix}
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1
\end{bmatrix}.$$

To clarify the notation, skip the zero entries in the matrix

$$\mathcal{M} = \frac{1}{24}\begin{bmatrix}
2 & 1 & & & & & & \\
1 & 2 & & & & & & \\
 & & 2 & 1 & & & & \\
 & & 1 & 2 & & & & \\
 & & & & 2 & 1 & & \\
 & & & & 1 & 2 & & \\
 & & & & & & 2 & 1\\
 & & & & & & 1 & 2
\end{bmatrix}, \qquad
\mathcal{S} = \frac{1}{2}\begin{bmatrix}
-1 & 1 & & & & & & \\
-1 & 1 & & & & & & \\
 & & -1 & 1 & & & & \\
 & & -1 & 1 & & & & \\
 & & & & -1 & 1 & & \\
 & & & & -1 & 1 & & \\
 & & & & & & -1 & 1\\
 & & & & & & -1 & 1
\end{bmatrix}.$$

If we approximate the time derivative with the backward Euler method and insert the global matrix form, we obtain the following system:

$$\mathcal{M}\frac{\boldsymbol{u}^n - \boldsymbol{u}^{n-1}}{\Delta t} - \mathcal{S}^T \boldsymbol{f}^n = -\mathcal{F}\boldsymbol{f}^n - \mathcal{F}^{\text{bound}} \begin{bmatrix} f(h_0(t^n)) \\ f(h_1(t^n)) \end{bmatrix}. \tag{1.4}$$

In the global form of the equation, we use the superscript n to denote the time level. It denotes different information than than the superscript k for the node values on element D^k.
For explicitly stating the "numerical flux" matrix $\mathcal{F}$ and the "boundary flux" matrix $\mathcal{F}^{\text{bound}}$, consider the specific forms of the numerical flux for two variants, the upwind flux and the central flux, respectively.

Specification of flux matrix $\mathcal{F}$ for upwind flux

Let us assume $f(u) = au$ with transport direction $a > 0$. Then, the upwind flux on D^k is given by the following formula:

Left boundary x^k: $f^*(x^k) = au_1^{k-1}$

Right boundary x^{k+1}: $f^*(x^{k+1}) = au_1^k$

To derive the matrix expression underlying this formulation for the case $K = 4$, consider the leftmost element first:

Left boundary $x^1 = 0$: $f^*(x^1) = au^{\text{bound}}(t) = ah_0(t)$ (only non-zero when multiplied by $\ell_0^1(0) = 1$)

Right boundary $x^2 = \frac{1}{4}$: $f^*(x^2) = au_1^1$ (only non-zero when multiplied by $\ell_1^1(\frac{1}{4}) = 1$)

Likewise, we obtain for the second element:

Left boundary $x^2 = \frac{1}{4}$: $f^*(x^1) = au_1^1$ (only non-zero when multiplied by $\ell_0^2(\frac{1}{4}) = 1$)

Right boundary $x^3 = \frac{1}{2}$: $f^*(x^2) = au_1^1$ (only non-zero when multiplied by $\ell_1^2(\frac{1}{2}) = 1$)

For the next elements, similar expressions are obtained. From this, we can extract the following matrix:

$$\mathcal{F}(a\boldsymbol{u}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} au_0^1 \\ au_1^1 \\ au_0^2 \\ au_1^2 \\ au_0^3 \\ au_1^3 \\ au_0^4 \\ au_1^4 \end{bmatrix},$$

and, for the boundary data,

$$\mathcal{F}^{\text{bound}}(a\boldsymbol{u}^{\text{bound}}(t)) = \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} ah_0(t) \\ ah_1(t) \end{bmatrix}.$$

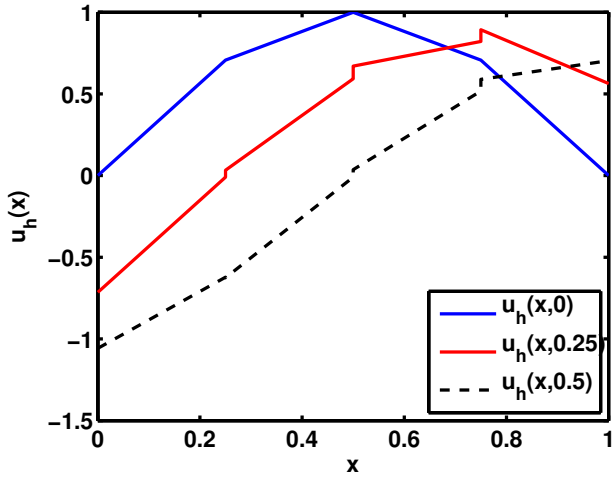
Note that we formally only should consider data from the left boundary, since for $a > 0$, no data h_1 should be prescribed. This is reflected in the matrix $\mathcal{F}^{\text{bound}}$.
Combining all terms involving the node values $\boldsymbol{u}^n$ on the left hand side and the known terms (old values $\boldsymbol{u}^{n-1}$ and boundary data) on the right hand side, we obtain the following discrete system:

$$\left(\frac{\mathcal{M}}{\Delta t} - a\mathcal{S}^T + a\mathcal{F}\right)\boldsymbol{u}^n = \frac{\mathcal{M}}{\Delta t}u^{n-1} - a\mathcal{F}^{\text{bound}}\boldsymbol{u}^{\text{bound}} \tag{1.5}$$

Let us explicitly evaluate the left hand side matrix for $K = 4$:

$$\frac{1}{24\Delta t}\begin{bmatrix} 2 & 1 & & & & & & \\ 1 & 2 & & & & & & \\ & & 2 & 1 & & & & \\ & & 1 & 2 & & & & \\ & & & & 2 & 1 & & \\ & & & & 1 & 2 & & \\ & & & & & & 2 & 1 \\ & & & & & & 1 & 2 \end{bmatrix} + \frac{a}{2}\begin{bmatrix} 1 & 1 & & & & & & \\ -1 & 1 & & & & & & \\ & & -2 & 1 & 1 & & & \\ & & & -1 & 1 & & & \\ & & & & -2 & 1 & 1 & \\ & & & & & -1 & 1 & \\ & & & & & & -2 & 1 & 1 \\ & & & & & & & -1 & 1 \end{bmatrix},$$

where the second matrix is $-a\mathcal{S}^T + a\mathcal{F}$.
Example. Consider the initial condition $u_0(x) = \sin(\pi x)$, transport velocity $a = 1$, and the boundary condition $u(0,t) = h_0(t) = \sin(-\pi t)$. This problem has the exact solution $u(x,t) = \sin(\pi(x-t))$. Using the approximation derived above and setting the time step size to $\Delta t = \frac{1}{8}$, we obtain the following picture for the solution at $t = \frac{1}{4}$ and $t = \frac{1}{2}$:



Specification of flux matrix $\mathcal{F}$ for central flux

The formula for the central flux is:

Left boundary x^k: $f^*(x^k) = \frac{a}{2}(u_1^{k-1} + u_0^k)$

Right boundary x^{k+1}: $f^*(x^{k+1}) = \frac{a}{2}(u_1^k + u_0^{k+1})$

The matrix formulation of these expressions is

$$\mathcal{F}(a\boldsymbol{u}) = \frac{1}{2}\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} au_0^1 \\ au_1^1 \\ au_0^2 \\ au_1^2 \\ au_0^3 \\ au_1^3 \\ au_0^4 \\ au_1^4 \end{bmatrix}.$$

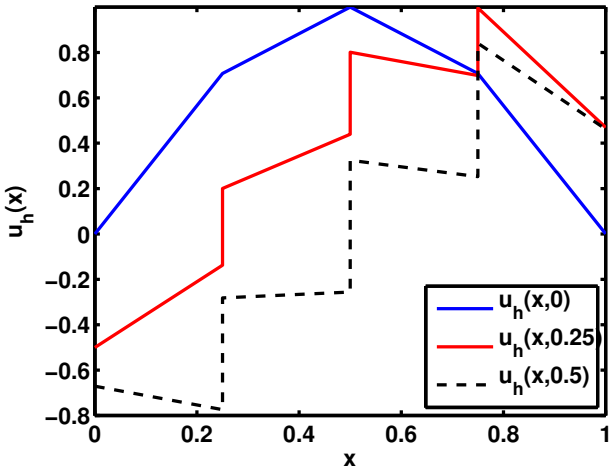
This formulation requires boundary data both on the left and right end of the boundary, h_0 and h_1. However, there is no physical boundary data at the right end of the computational domain. Nonetheless, we assume the data on the right to be given, too. In realistic settings, upwind approaches are essential on outflow boundaries.

$$\mathcal{F}^{\text{bound}}(a\boldsymbol{u}^{\text{bound}}(t)) = \frac{1}{2}\begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} ah_0(t) \\ ah_1(t) \end{bmatrix}.$$

The final system matrix on the left hand side of equation (1.5) is thus:

$$\frac{1}{24\Delta t}\begin{bmatrix} 2 & 1 & & & & & & \\ 1 & 2 & & & & & & \\ & & 2 & 1 & & & & \\ & & 1 & 2 & & & & \\ & & & & 2 & 1 & & \\ & & & & 1 & 2 & & \\ & & & & & & 2 & 1 \\ & & & & & & 1 & 2 \end{bmatrix} + \frac{a}{2}\begin{bmatrix} 0 & 1 & & & & & & \\ -1 & 0 & 1 & & & & & \\ & -1 & 0 & 1 & & & & \\ & & -1 & 0 & 1 & & & \\ & & & -1 & 0 & 1 & & \\ & & & & -1 & 0 & 1 & \\ & & & & & -1 & 0 & 1 \\ & & & & & & -1 & 0 \end{bmatrix}.$$

Example. Consider again the example with analytic solution $u(x,t) = \sin(\pi(x-t))$. Visualization at $t = \frac{1}{4}$ and $t = \frac{1}{2}$ with $\Delta t = \frac{1}{8}$ with the central flux gives the following picture:



The solution is considerably worse for the central flux than for the upwind flux. As can be seen from the advection matrix, the structure is very similar to central finite differences (three-point central difference formula). Such schemes are not ideal for convection problems. More precisely, from the matrix structure we observe that in the interior of the computational domain, there is no coupling between the neighboring solution data, such as u_0^k and u_1^k or u_1^k and u_0^{k+1}. Rather, only every second point is directly tied by the equation. Thus, there can be spurious node-to-node (sometimes also called checkerboard) oscillations that can grow as time increases.

Interpretation of upwind flux

If we compare the advection matrices for the upwind and the central fluxes, respectively, we observe that there is no entry in the (2,3), (4,5), (6,7) entries for the former flux. These entries represent coupling from the element $k+1$ to the element k. However, it is the nature of the upwind flux that information is passed along the flux direction only. Naturally, this corresponds to no information flux from right to left.
In terms of the matrix system, we see that we can first solve for the two degrees of freedom on element $k=1$, namely $u_0^{1,n}, u_1^{1,n}$ at time level n, which is driven by the old values $u_{0/1}^{1,n-1}$ and the boundary condition at time t_n. If $u_1^{1,n}$ is known, we can in turn solve for the two degrees of freedom $u_{0/1}^{2,n}$, and so on. In terms of the underlying mathematics, this procedure is a forward substitution of 2×2 blocks. In terms of the flow physics, we realize that the upwind flux at internal faces between elements $k-1$ and k is imposed the same way as the boundary condition.
This is a general principle in discontinuous Galerkin methods: Information from external boundaries is passed into the system similarly as from one element to the next. Since the fluxes have been derived from the weak form of the equation, we call this method for imposing boundary conditions a *weak* imposition of boundary conditions. Similarly, the coupling between the elements over faces (in general dimensions by face integrals), is called *weak*. This is in contrast to the common procedure in finite element methods where (Dirichlet) boundary conditions are set by injecting the node values directly, a *strong* imposition of the data.

Analogy to finite elements

Can this formulation be interpreted as an extension of continuous finite elements?
Let us consider what happens if we enforce the continuity of the basis functions and test functions in the DG scheme, i.e., $u_1^{k-1} = u_0^k$. Then, the summation over all elements means that we need to sum the rows and columns in the advection matrix corresponding to $\ell_1^{k-1} = \ell_0^k$, i.e., the second and third row, the fourth and fifth, and so on. Thus, the flux matrix $\mathcal{F}$ is simply the zero matrix.

The situation with periodic boundary conditions

Consider the slightly modified problem

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \quad \text{for } x\in\Omega=(0,1),\ t\geq 0,$$
$$u(x,0) = u_0(x) \text{ (initial condition)}, \qquad u(0,t) = u(1,t) \text{ (periodic boundary condition)}.$$

The discretization for this equation is developed similarly as above. The key difference is that no "outer" boundary is present in the sense that the domain is wrapped around at the end points. This means that the solution at the left end point, x^1, needs to be connected to the solution at the right point, x^{K+1}. In other words, the numerical flux functions at x^1 and x^{K+1} link the following quantities

$$f_h^1(x^1) = f^*(u_1^K, u_0^1), \qquad f_h^K(x^{K+1}) = f^*(u_1^K, u_0^1),$$

i.e., they refer to the exactly same numerical flux. In matrix notation, this results in the matrix

$$\mathcal{F}(a\boldsymbol{u}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} au_0^1 \\ au_1^1 \\ au_0^2 \\ au_1^2 \\ au_0^3 \\ au_1^3 \\ au_0^4 \\ au_1^4 \end{bmatrix}$$

for the upwind flux and in

$$\mathcal{F}(a\boldsymbol{u}) = \frac{1}{2}\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} au_0^1 \\ au_1^1 \\ au_0^2 \\ au_1^2 \\ au_0^3 \\ au_1^3 \\ au_0^4 \\ au_1^4 \end{bmatrix}$$

for the central flux. Note the new matrix entries $\mathcal{F}_{2K,1}$ and $\mathcal{F}_{1,2K}$ that link the left and right boundaries.

1.3 More general expressions for the numerical flux

We introduce some notation in analogy to [Hesthaven and Warburton, 2008]. We have seen that the expression of the numerical flux involves the discrete solution values on the element interface from both sides. Instead of writing the two solution values at x^k as u_1^{k-1} and u_0^k, we simplify the indices and write:

u^-: Solution on element D^k (interior solution)

u^+: Solution on neighbor, D^{k-1} or D^{k+1} (exterior solution)

We write the average of the interior and exterior values as

$$\{\{u\}\} = \frac{u^- + u^+}{2},$$

and the jump operator as

$$[\![u]\!] = \hat{\boldsymbol{n}}^- u^- + \hat{\boldsymbol{n}}^+ u^+.$$

Since the normal vector $\hat{\boldsymbol{n}}^+$ on the neighbor points exactly in the opposite direction as $\hat{\boldsymbol{n}}^-$, we find that

$$[\![u]\!] = \hat{\boldsymbol{n}}^-(u^- - u^+),$$

highlighting the fact we indeed look at the jump of the solution.
For the various forms above, we wrote the numerical flux $f^*(u^-, u^+)$ in terms of the discrete solution values from the left and right element. For example, the **upwind flux** for $f(u) = au$ obeys the following expression:

$$(au)^{*,\text{upwind}} = \{\{au\}\} + |c|\frac{1}{2}[\![u]\!],$$

whereas the **central flux** is given by:

$$(au)^{*,\mathrm{central}} = \{\{au\}\}.$$

We can combine these two fluxes by a parameter α:

$$(au)^* = \{\{cu\}\} + |a|\frac{1-\alpha}{2}[\![u]\!], \quad 0 \leq \alpha \leq 1.$$

In the expressions derived in Sec. 1.2, we included separate expressions for the boundary conditions. With the notation in terms of u^- and u^+, we can simply set u^+ to the given boundary data and use the same numerical flux and procedure as for interior faces (as long as there are boundary conditions to be imposed on that boundary, as discussed for the central flux above).

General fluxes

In the case of more general fluxes f, several generalizations can be made. A very popular one is the Lax–Friedrichs flux,

$$f^{*,LF}(u^-,u^+) = \{\{f(u)\}\} + \frac{C}{2}[\![u]\!] = \frac{f(u^-)+f(u^+)}{2} + \frac{C}{2}\hat{\boldsymbol{n}}^-(u^- - u^+).$$

The constant in the Lax–Friedrichs flux is set to

$$C \geq \max_{\inf u_h(x) \leq s \leq \sup u_h(x)} \left|\frac{\partial f}{\partial u}(s)\right|$$

for a so-called global Lax–Friedrichs flux, i.e., the largest possible value for the derivative of f with the expected range of the solution u_h. Finding this maximum in the general case is difficult, so one often resorts to the *local Lax–Friedrichs flux*

$$C \geq \max_{\min(u^-,u^+) \leq s \leq \max(u^-,u^+)} \left|\frac{\partial f}{\partial u}(s)\right|.$$

The Lax–Friedrichs flux is simply to define and often very efficient. However, it does not always deliver optimal accuracy.

Systems of equations

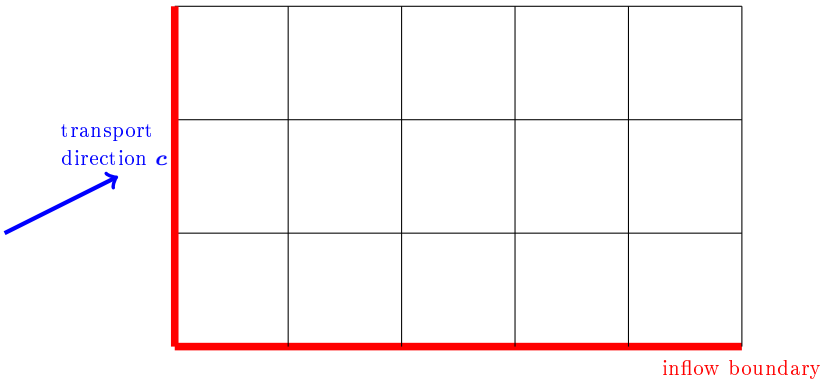
If we want to extend the formulation beyond the case where the solution u is scalar, we can in principle proceed for each component individually. We can use the Lax–Friedrichs flux as in the scalar case, but need to change the definition of the constant C. Instead of taking it as the maximum absolute value, we define it by the maximum eigenvalue of the flux Jacobian,

$$C = \max_{\boldsymbol{u}} \left|\lambda\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right)\right|.$$

1.4 Time integration

In the above, we applied the backward Euler scheme for time propagation, i.e., an implicit time stepping algorithm. The solution of the (sparse) linear system (1.4) represents the main computational cost in implicit methods. In 1D, the cost is easily controllable and factorization algorithms exist that do utilize the tridiagonal matrix structure (Thomas algorithm) and give a sparse factorization. However, the situation changes in higher dimensions. As for finite element discretizations, the factorization

process for computing an upper and lower triangular matrix gives rise to so-called fill-in, i.e., the factors L and U of the matrix contain considerably more non-zero elements than the matrix itself. This can be seen from a lexicographic ordering of degrees of freedom on a square with K^2 degrees of freedom. As compared to the originally $\mathcal{O}(K^2)$ nonzero entries, the elimination with this numbering gives rise to $\mathcal{O}(K^3)$ nonzero entries, coupling within a whole line in the direction where indices run quickly. The fill-in can be reduced by better numbering of degrees of freedom. The best algorithms in use today offer $\mathcal{O}(K^2 \log(K^2))$ nonzero entries in the factors (nested dissection or quasi-minimum degree algorithms). In 3D, the respective numbers for K^3 degrees of freedom are $\mathcal{O}(K^4)$. Due to this superlinear growth in complexity, iterative solvers are used for the largest problem sizes. However, even the best iterative solvers involve a multiple of the operations than the evaluation of the integrals on the right hand side and may involve a few to many hundreds of matrix-vector products.
In the DG method with upwind flux, an attractive solver can be constructed. The element-by-element (left-to-right) solution due to the restricted dependencies presented above can be applied in higher dimensions as well. Consider the following situation:



If we number the elements from the lower left to the upper right (several numberings possible), there are elements where all inflow data is known (either by the boundary condition or the already computed neighbor) and the solution on the element can be computed directly. Taken together, this amounts to a direct solution method. Such a downstream numbering of elements (or degrees of freedom) thus results in a very efficient solution method. However, this is only possible for simple flow patterns a. If a varies in space and time, finding an optimal ordering of cells can either be very expensive or even impossible. An example of the latter is wave propagation (acoustic wave equation), where transport is in more than one direction and no suitable numbering can be found. Likewise, even a slightly different numerical flux f^* than the pure upwind one will result in a tighter coupling.
All these reasons suggest that the highly efficient solver outlined above is only useful for a handful problems and not in the general case. A remedy can be to use an inexact downstream solution as a preconditioner in a similar fashion as the Gauss–Seidel iterative solver, i.e., by using the new solution upstream to an element and the old solution downstream. By combining sweeps in different directions through the mesh (e.g. forward and backward), flow in several directions can be represented by the preconditioner. This way, an efficiency close to the direct solution method outlined above can be reached in several cases, where "close" refers to costs within a factor of 3 to 10 from the direct method.

Explicit time integration

Due to the considerable cost of solving linear systems, one often resorts to explicit time integration for problems dominated by transport. We write the DG operator for the unknown coefficient values $\boldsymbol{u}_h$ in an abstract way as

$$\frac{d\boldsymbol{u}_h}{dt} = \mathcal{L}_h(\boldsymbol{u}_h, t),$$

where the quantity $\mathcal{L}_h(\boldsymbol{u}_h, t)$ is given by

$$\mathcal{L}_h(\boldsymbol{u}_h, t) = \mathcal{M}^{-1}\left(\mathcal{S}^T - \mathcal{F}\right)\boldsymbol{f}(\boldsymbol{u}_h) - \mathcal{M}^{-1}\mathcal{F}^{\text{bound}}\begin{bmatrix} f(h_0(t)) \\ f(h_1(t)) \end{bmatrix}.$$

In an explicit time stepping method, the new values $\boldsymbol{u}^n$ can be computed explicitly in terms of knowledge of only $\boldsymbol{u}^{n-1}, \boldsymbol{u}^{n-2}$ without solving linear/nonlinear systems.
Let us consider the forward Euler method,

$$\boldsymbol{u}^n = \boldsymbol{u}^{n-1} + \Delta t \mathcal{L}_h(\boldsymbol{u}^{n-1}, t^{n-1}).$$

Inserting the operator $\mathcal{L}$, this is nothing else than the scheme

$$\mathcal{M}\frac{\boldsymbol{u}^n - \boldsymbol{u}^{n-1}}{\Delta t} - \mathcal{S}^T\boldsymbol{f}^{n-1} = -\mathcal{F}\boldsymbol{f}^{n-1} - \mathcal{F}^{\text{bound}}\begin{bmatrix} f(h_0(t^{n-1})) \\ f(h_1(t^{n-1})) \end{bmatrix}.$$

This explicit formula for $\boldsymbol{u}^n$ is attractive in the DG context. In standard finite elements, the inverse $\mathcal{M}^{-1}$ requires either factorization or iterative solution of a matrix system, thus eliminating the desirable property of explicitness. (Note, though, that finite element methods with approximately diagonal mass matrices exist also in the high order case, so called mass-lumped methods.) However, the particular form of the DG basis functions makes the matrix $\mathcal{M}$ block-diagonal, for which the inverse is given by

$$\mathcal{M}^{-1} = \begin{bmatrix} \left(\mathcal{M}^1\right)^{-1} & & & \\ & \left(\mathcal{M}^2\right)^{-1} & & \\ & & \left(\mathcal{M}^3\right)^{-1} & \\ & & & \ddots \end{bmatrix}$$

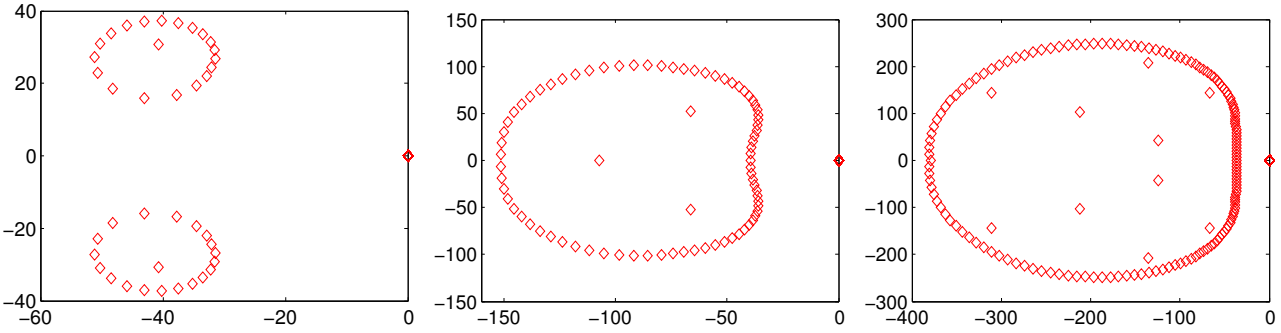
Computing the inverse of the elemental mass matrices is cheap, as is their application onto a vector because these are only 2×2 (or $(p+1) \times (p+1)$) matrices with good condition numbers.
Of course, the forward Euler method is not sufficiently accurate in the general case due to the truncation error of size $\mathcal{O}(\Delta t)$. This problem is addressed by using higher order time stepping schemes. In this lecture, we consider mainly Runge–Kutta methods. One example is the classical Runge–Kutta method of order four, which computes $\boldsymbol{u}^n$ from $\boldsymbol{u}^{n-1}$ using the following four stages:

$$\begin{aligned}
\boldsymbol{k}^{(1)} &= \mathcal{L}_h(\boldsymbol{u}^{n-1}, t^{n-1}), \\
\boldsymbol{k}^{(2)} &= \mathcal{L}_h\left(\boldsymbol{u}^{n-1} + \frac{1}{2}\Delta t\boldsymbol{k}^{(1)}, t^{n-1} + \frac{1}{2}\Delta t\right), \\
\boldsymbol{k}^{(3)} &= \mathcal{L}_h\left(\boldsymbol{u}^{n-1} + \frac{1}{2}\Delta t\boldsymbol{k}^{(2)}, t^{n-1} + \frac{1}{2}\Delta t\right), \\
\boldsymbol{k}^{(4)} &= \mathcal{L}_h\left(\boldsymbol{u}^{n-1} + \Delta t\boldsymbol{k}^{(3)}, t^{n-1} + \Delta t\right), \\
\boldsymbol{u}^n &= \boldsymbol{u}^{n-1} + \frac{1}{6}\Delta t\left(k^{(1)} + 2k^{(2)} + 2k^{(3)} + k^{(4)}\right).
\end{aligned}$$

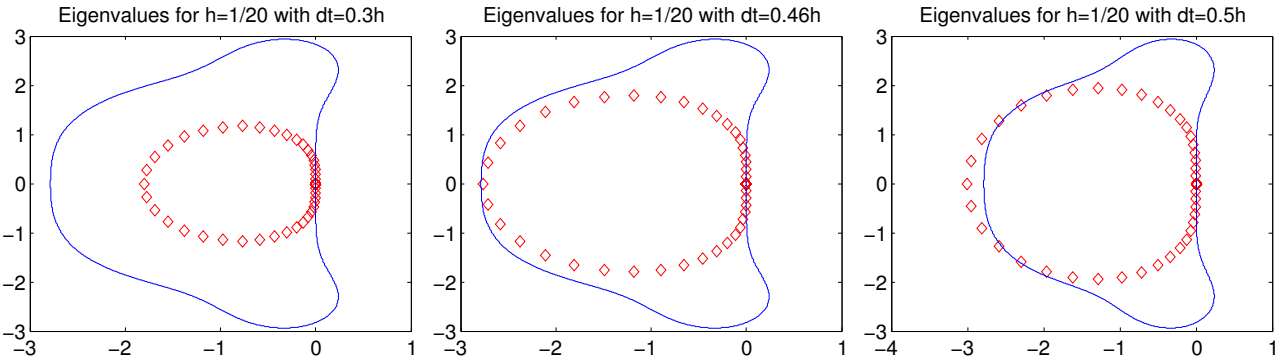
The methods discussed in the course book [Hesthaven and Warburton, 2008] are mainly so-called low-storage Runge–Kutta methods which need not store all intermediate values $\boldsymbol{k}^{(i)}$.
However, the efficiency of explicit time integration comes at a price: It introduces a time step limit for stable computations, the so-called Courant–Friedrichs–Lewy (CFL) condition. In terms of the ODE solver, the CFL condition is an expression for its stability region. The stability region is the part of the complex plane where the time propagator results in bounded solutions for the model problem

$$\frac{d\phi}{dt} = \lambda\phi.$$

For explicit methods such as the forward Euler method or the classical Runge–Kutta method of order four, the stability region is a bounded region around the origin extending into the negative complex plane. For the matrix problem expressed by $\mathcal{L}_h$, we need to consider the eigenvalue distribution.
The following figures show the distribution of eigenvalues of the matrix $\mathcal{M}^{-1}\left(\mathcal{S}^T - \mathcal{F}\right)$ in the complex plane for $K = 20$, $K = 40$, and $K = 80$ elements, respectively, for the upwind flux:



The pictures illustrate that the eigenvalues are distributed in the complex plane and all eigenvalues fulfill $\mathrm{Re}(\lambda) \leq 0$. We can recognize a pattern by checking the limits of the axes: By and large, both the largest real and imaginary part seem to behave as $\mathcal{O}(K) = \mathcal{O}\left(\frac{1}{h}\right)$.
The eigenvalues actually present in the numerical scheme are multiplied by Δt and thus pulled towards the origin. By a proper selection of Δt, we can make sure that all the eigenvalues fit into the stability region of the time stepper. To eliminate the effect of the imposition of boundary conditions, let us consider the same problem with periodic boundary conditions. The following figures show three choices of the time step size, $\Delta t = 0.3h$, $\Delta t = 0.46h$, $\Delta t = 0.5h$ for $K = 20$ elements. The figures include the eigenvalue distributions of the scaled matrices $\Delta t\mathcal{M}^{-1}\left(\mathcal{S}^T - \mathcal{F}\right)$ (diamonds), superimposed with the stability region of the classical Runge–Kutta method of order four (solid line).



The plots suggest stable results for the first two cases and unstable results for the latter. Indeed, when numerically integrating the equation we observe exactly this prediction. When moving slightly above the stability limit, e.g. with $\Delta t = 0.47h$, we observe nice solutions initially but after long times, e.g. $t > 100$, we again observe instabilities. In general, a single eigenvalue outside of the stability region is enough to destroy the solution. This can be explained by the fact that any contribution to the corresponding eigenvector, be it an initial condition or merely roundoff errors, will eventually grow large enough according to σ^n with $|\sigma| > 1$.
When a transport velocity $a \neq 1$ is used, the CFL condition is scaled in the form

$$\Delta t \leq \Delta t_{\max} \sim \frac{h}{a},$$

There is an intuitive explanation for the CFL number in the finite difference context: It specifies that the time step must not be larger than the value for which the value at grid point $k-1$ travels to k. If

it is larger, it would travel further that, and the difference stencil around k, involving $k-1, k, k+1$, could not take up the "correct" physical value which is left to $k-1$.

Behavior with variable mesh size

- Variable mesh size $h^1, h^2, h^3, \ldots$, not all the same:

$$\Delta t_{\max} \sim \frac{h_{\min}}{a}, \qquad h_{\min} = \min\{h^1, h^2, h^3, \ldots, h^K\}.$$

- Coefficients (transport speed) constant within the elements but with different values on different elements, $a^1, a^2, a^3, \ldots$,:

$$\Delta t_{\max} \sim \frac{h}{a_{\min}}.$$

1.5 Summary of discontinuous Galerkin methods

To summarize, discontinuous Galerkin methods are characterized by the following properties:

- Like finite elements, they use a subdivision of the domain into elements and use (higher order) polynomials inside the elements. This allows the method to model complex geometries and adapt the mesh to the solution, e.g. by fine meshes in boundary layers.
- Similar to finite volumes, numerical fluxes exchange information between neighboring elements in a local way. However, this can be done without affecting convergence orders because we evaluate the solution (consistently) at the interfaces between elements.
- The finite-volume inspired construction allows for upwinding and taking the directional information into account by means of suitable flux functions.
- The mass matrix is block-diagonal, which is easier to invert than the finite element mass matrix. This enables efficient explicit time integration.

The price to pay for the increased flexibility is an increased number of degrees of freedom: At each internal interface, we have two degrees of freedom instead of only one in FEM. (Finite volumes can be considered as the lowest order DG-FEM case $N = 0$, albeit extended with additional reconstruction techniques.) For linear elements in 2D, the number of degrees of freedom even increases by a factor of 4, and by a factor of 8 in 3D, as compared to FEM. With increasing polynomial degree N, the increase in the number of degrees of freedom becomes smaller according to the formula

$$\frac{(N+1)^d}{N^d}$$

for quadrilateral/hexahedral bases in d dimensions. This formula expresses the fact that in continuous elements N degrees of freedom per dimension can be associated to each element in case we disregard the boundary of the computational domain, whereas the full $N+1$ degrees of freedom to the N-th degree polynomial basis belong to an element in DG.
Be aware that the number of degrees of freedom are only a proxy for the computational work: In the end, the important metric is always the solution quality obtained for a given amount of computations, and methods with more degrees of freedom need not necessarily be inferior in terms computational time, leave alone accuracy.

Notation

At the end of the chapter, we introduce a short-hand notation for the DG method. This notation is used frequently in the DG literature, even though not in this precise form in the course book [Hesthaven and Warburton 2008]. We denote by $(\cdot,\cdot)_{\mathsf{D}^k}$ the bilinear form with one slot for the discrete solution u_h^k and one for the test function, expressing integration of the product of the functions in the two slots on element D^k. Similarly, we write $\langle\cdot,\cdot\rangle_{\partial\mathsf{D}^k}$ for the boundary bilinear form on $\partial\mathsf{D}^k$. In the one-dimensional case, $\partial\mathsf{D}^k$ consists of the two points x^k and x^{k+1}. Thus, we may write the DG problem concisely as

$$\left(\frac{\partial u_h^k}{\partial t}, v^k\right)_{\mathsf{D}^k} - \left(f(u_h^k), \frac{\partial v^k}{\partial x}\right)_{\mathsf{D}^k} + \left\langle \hat{\boldsymbol{n}} f^*(u_h^-, u_h^+), v^k\right\rangle_{\mathsf{D}^k} = \left(g, v^k\right)_{\mathsf{D}^k}.$$

Taking all elements together on the domain $\Omega_h = \bigcup_k \mathsf{D}^k$, we write the DG problem as follows:

$$\left(\frac{\partial u_h}{\partial t}, v\right)_{\Omega_h} - \left(f(u_h), \frac{\partial v}{\partial x}\right)_{\Omega_h} + \left\langle \hat{\boldsymbol{n}} f^*(u_h^-, u_h^+), v\right\rangle_{\partial\Omega_h} = (g, v)_{\Omega_h},$$

where we denote by $\partial\Omega_h = \bigcup_k \partial D^h$ the union of all element boundaries. In the definition of $\partial\Omega_h$, we visit internal interfaces (points) twice and boundary points once. We name the entities of $\partial\Omega_h$ *faces* in the general d-dimensional case. For example, a triangle is confined by three faces (i.e., lines/edges), a tetrahedron by four faces (triangles), or a hexahedron by six faces (rectangles). An alternative statement of the global DG problem is given by introducing the collection of all faces Γ_h, now with each face visited only once:

$$\left(\frac{\partial u_h}{\partial t}, v\right)_{\Omega_h} - \left(f(u_h), \frac{\partial v}{\partial x}\right)_{\Omega_h} + \left\langle f^*(u_h^-, u_h^+), [\![v]\!]\right\rangle_{\Gamma_h} = (g, v)_{\Omega_h}. \tag{1.6}$$

As we have seen above, DG schemes are formulated such that the numerical flux f^* is the same from both sides of the face. In other words, it does not matter which side of a face we designate by the superscript u^+ and which by u^-. The term $[\![v]\!]$ collects the contributions of the test function times normal vector, $v^-\hat{\boldsymbol{n}}^-$ and $v^+\hat{\boldsymbol{n}}^+$ from the two sides of the face. On boundaries where only the test function v^- is present, we set $[\![v]\!] = v\hat{\boldsymbol{n}}$. Also, the solution component $\boldsymbol{u}^+$ is obtained from boundary conditions.

The advantage of this notation is that it is naturally extended to higher dimensions by replacing the spatial derivative $\partial/\partial x$ by the gradient operator ∇ or the divergence operator $\nabla\cdot$,

$$\left(\frac{\partial u_h}{\partial t}, v\right)_{\Omega_h} - (\boldsymbol{f}(u_h), \nabla v)_{\Omega_h} + \left\langle \boldsymbol{f}^*(u_h^-, u_h^+), [\![v]\!]\right\rangle_{\Gamma_h} = (g, v)_{\Omega_h}.$$

Of course, the (area/volume) integrals are over multiple spatial variables $\boldsymbol{x} = (x, y, \ldots)$ in this form, and the flux is multi-dimensional, $\boldsymbol{f}(u) = (f_1(u), f_2(u), \ldots)$.

Alternative form of the DG scheme

Equation (1.6) can be mathematically reformulated by integrating the spatial derivative by parts once again, which gives the following equation for a single element $\mathcal{D}^k$,

$$\left(\frac{\partial u_h^k}{\partial t}, v^k\right)_{\mathsf{D}^k} + \left(\frac{\partial f(u_h^k)}{\partial x}, v^k\right)_{\mathsf{D}^k} + \left\langle f^*(u_h^-, u_h^+) - f(u_h^k), \boldsymbol{n} v^k\right\rangle_{\partial\mathsf{D}^k} = \left(g, v^k\right)_{\mathsf{D}^k}.$$

The course book calls this the *strong* version of the DG scheme in order to emphasize the derivative of the flux f. This must be contrasted to the mathematical notation of the point-wise strong form of a

partial differential equation. For linear functions $f(u)$ and polynomial bases, this form is mathematically equivalent to the form in equation (1.6). From the strong DG form, we identify the numerical flux f^* not only as a link between the two values $f(u_h^-)$ and $f(u_h^+)$ and information exchange between the elements, but also as a *penalty* on the difference between $f(u_h)$ and the value f^*. Recall that our starting point was this form of the transport equation without any face integral. In other words, we identify one main distinction of the DG method as compared to continuous finite elements: It sacrifices continuity but nevertheless *weakly penalizes of solution jumps*.

1.6 Check yourself

- Give a characterization of the discontinuous Galerkin method as compared to the finite element and finite volume methods, respectively.
- Given a mesh of K elements and basis functions of degree N in 1D, how many degrees of freedom are there for a scalar conservation law?
- State the DG-FEM discretization of the linear transport equation including a numerical flux.
- Explain the role of the numerical flux in a DG scheme.
- What is the difference between an upwind flux and a central flux? State these two fluxes.
- How is the Lax–Friedrichs flux related to the upwind flux?
- Explain how inflow and outflow boundaries are handled for the transport equation.
- Explain why explicit time integration is more attractive with DG methods than with continuous finite elements.
- How does the maximum time step for the advection equation discretized with DG-FEM and explicit time integration depend on the mesh size and the polynomial degree?

2 Higher order basis functions

Reading instructions: More details on the content of this lecture can be found in chapters 3 and 4 of the course book [Hesthaven and Warburton, 2008].

2.1 Nodal basis functions

For the representation of the DG-FEM solution with polynomials of degree N with nodal Lagrangian polynomials, the following form is used:

$$u_h^k(x,t) = \sum_{j=0}^{N} \ell_j^k(x) u_j^k(t), \tag{2.1}$$

where ℓ_j^k is the Lagrange polynomial associated to element k and node j inside the element. As in finite elements, we define the polynomials in terms of the unit interval $\mathsf{I} = (-1, 1)$ with variable r. The integration is performed on the reference element, and a transformation is used between r and x. The approach is exactly the same as in finite elements and we do not detail this further. We recall that for a linear transformation $x(r) = \frac{x^{k+1}-x^k}{2} r + \frac{x^{k+1}+x^k}{2}$ in 1D, the mass matrix computed on the unit interval is scaled by $\frac{h^k}{2}$, whereas the convection matrix $\mathcal{S}$ does not depend on the element size since the integration factor $\frac{h^k}{2}$ cancels exactly with the derivative transformation $\frac{dr}{dx} = \frac{2}{h^k}$.
For Lagrange polynomials, we need to select nodal points r_j that define the Lagrange polynomials ℓ_j. The straight-forward idea of choosing equidistant points $r_j = -1 + \frac{2j}{N}$ turns out to be a bad idea, though. From an interpolation point of view, there is an intuitive explanation for this deficiency: Severe over- and undershoots at the end of the interpolation region are observed for certain functions (Runge's phenomenon). This problem is observed in DG-FEM if initial conditions are directly interpolated into the nodal values u_j^k for certain functions. However, the method turns out to produce reasonable results in most situations for moderate degree nonetheless.[1] Instead, the major problem is the conditioning of the matrices. In the following table, we list the condition number of the mass matrix on the unit interval I for various degrees of the polynomial:

[1] **Explanation:** Irrespective of the particular form of the basis functions, the polynomials represented on the element D^k are always the same. The weak form seeks to find the best polynomial approximation on the elements, i.e., among the same set of functions. In absence of roundoff errors, the same solution u_h will be found.

N	cond($\mathcal{M}^{\mathsf{I}}$)
1	3.0
2	6.88
3	9.36
4	14.1
5	23.6
8	220
11	$4.85 \cdot 10^3$
15	$5.29 \cdot 10^5$
19	$7.43 \cdot 10^7$

We observe an exponential increase in the condition number. This behavior has the following implications:

- If the initial condition is imposed by projection instead of interpolation, i.e., if we solve the equation
$$(u^k, v^k)_{\mathsf{D}^k} = (u_0, v^k)_{\mathsf{D}^k}$$
for the initial function u_0 on each element D^k, we will make an error proportional to cond(M^{I}). If the order becomes high, this amounts to a significant problem. (Note that this equation is equivalent to finding the function u^k that has minimal distance to u_0 in the L_2 norm, i.e., the minimization problem $\min_{u^k} \|u^k - u_0\|_{L_2(\mathsf{D}^k)}$ in the usual variational framework.)
- The stability of the explicit time stepping is not strongly influenced because both $\mathcal{M}^k$ and $\mathcal{S}^k$ have the same underlying bad eigenvalue distribution. For the propagation matrix $\mathcal{M}^{-1}(\mathcal{S}^T - \mathcal{F})$ the effect cancels mostly out. However, bad conditioning does influence accuracy in terms of roundoff errors.
- If implicit time stepping is considered with matrices of the form $\frac{\mathcal{M}}{\Delta t} - \mathcal{S}^T + \mathcal{F}$, the resulting matrix is very poorly conditioned and gives rise to considerably increased iteration counts with many iterative solvers.

To avoid this problem, non-equidistant node distributions need to be used for the higher order case. A well-suited node distribution for N-th order polynomials is given by the integration points of the Gauss–Lobatto quadrature rule on $N + 1$ points. The Gauss–Lobatto quadrature rule is a special quadrature rule that selects points and weights simultaneously like the Gauss quadrature but always includes the end points of the unit interval, ± 1.[2] For two to five nodes, the points are given by:

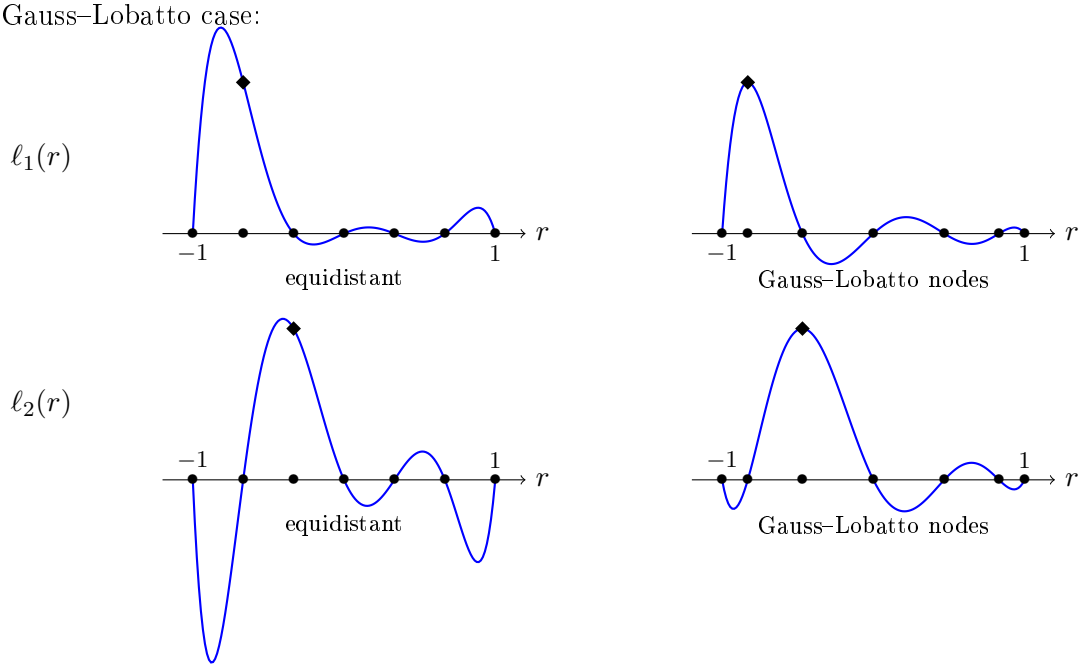
$N + 1$	GL nodes	equidistant nodes
2	± 1	± 1
3	$0, \pm 1$	$0, \pm 1$
4	$\pm\sqrt{1/5} (= 0.4472), \pm 1$	$\pm 0.3333, \pm 1$
5	$0, \pm\sqrt{3/7} (= 0.6547), \pm 1$	$0, \pm 0.5, \pm 1$

Already for $N = 5$ we observe that the points are considerably closer to the end points than with equidistant points. For $N = 10$, the point closest to 1 is 0.8 for equidistant nodes but 0.934 for Gauss–Lobatto points. If we look at the conditioning, we get the following behavior:

[2] For numerical integration, Gauss–Lobatto formulas integrate polynomials up to order $2N - 1$ exactly as opposed to Gaussian quadrature where the integrals are exact up to degree $2N + 1$ for $N + 1$ points. However, we do not intend to compute integrals here (yet).

N	$\mathrm{cond}(\mathcal{M}^{\mathsf{I}})$
1	3.0
2	6.88
3	8.65
4	10.4
5	12.0
8	16.7
11	21.4
15	27.7
19	34.0

This point distribution results in a linear increase in the condition number of $\mathcal{M}^{\mathsf{I}}$ in N. The Gauss–Lobatto nodes are thus a good set of nodal points for interpolations that work well over a wide range of orders.
Finally, we compare the shape of the basis functions ℓ_1 and ℓ_2 for $N = 6$ for the equidistant and the Gauss–Lobatto case:



Observations:

- The polynomials for equidistant points show considerable over- and undershoots at the end of the interval, in particular for the inner functions.
- The polynomials for the Gauss–Lobatto always reach the maximum +1 in the respective node and the oscillations are very moderate.

2.2 Modal basis functions

For continuous finite elements, the natural choice are Lagrange polynomials where placing nodes on the element boundary and interior, respectively, allows to neatly impose continuity constraints. Shape functions which are one in the nodes ± 1 of the reference element extend into the neighbor. The polynomial spaces in DG-FEM can be chosen much more freely since no continuity needs to be ensured. Thus, it is possible to go away from the restriction that all basis functions except one should be zero in nodes, the so-called *nodal* definition. In this section, we consider a representation in terms of a

so-called *modal* basis, i.e., an expansion in terms of functions in various degrees, such as $1, r, r^2, \ldots$. Again, it turns out that the naive selection, the *monomial basis* $1, r, r^2, \ldots$, is badly conditioned (for polynomial degree $N = 16$, the condition number of the mass matrix is on the order of 10^{11}). Instead, normalized Legendre polynomials of the form

$$\psi_j(r) = \frac{P_j(r)}{\sqrt{2/(2j+1)}}$$

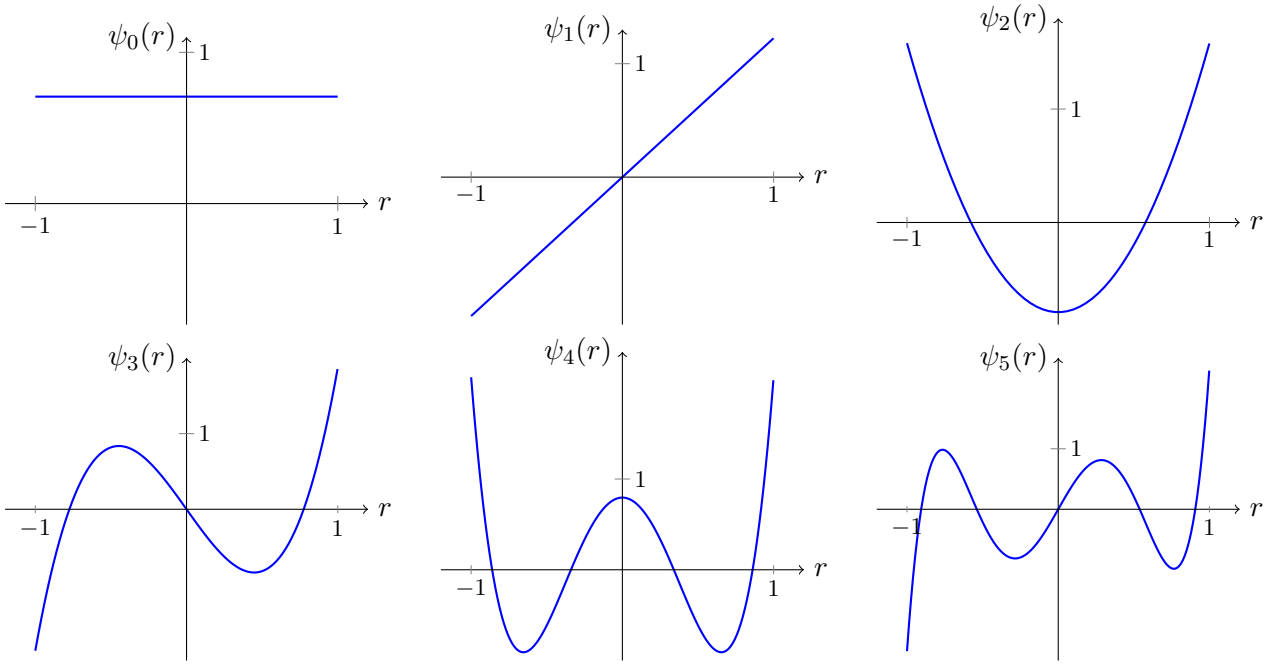
are chosen, where P_j denote the Legendre polynomials of degree j and the normalized version of these polynomials is given by the recurrence relation

$$\psi_0(r) = \frac{1}{\sqrt{2}}, \quad \psi_1(r) = \sqrt{\frac{3}{2}}r,$$
$$\psi_j(r) = \frac{1}{a_j} r \psi_{j-1}(r) + \frac{a_{j-1}}{a_j}\psi_{j-2}(r), \quad \text{with} \quad a_j = \sqrt{\frac{j^2}{(2j+1)(2j-1)}}.$$

The normalization of the polynomials is done such that the relation

$$(\psi_j, \psi_j)_{\mathsf{I}} = 1$$

holds, i.e., the integral of the square of the polynomials over the reference element $\mathsf{I} = (-1, 1)$ is one. The first six polynomials ψ_j look the following (y-axis not scaled the same in all figures):



For this particular choice of basis functions, the mass matrix on the reference element is the unit matrix,

$$\mathcal{M}^{\mathsf{I}} = \mathcal{I}.$$

Thus, the condition number takes the best possible value one. On elements of length h^k, the mass matrix is a unit matrix scaled by $h^k/2$. This diagonal mass matrix is an obvious advantage of the modal basis. It is even cheaper to invert than the block-diagonal matrix in the nodal case, making explicit time stepping very simple. Moreover, the modal basis provides a natural decomposition of the elemental approximation into the various polynomial degrees j via the function ψ_j, the so-called

modes. A drawback is the fact that the evaluation of the polynomials on the element interfaces and at the boundary becomes more expensive: While we can simply put ℓ_0 for the left element boundary and ℓ_N for the right boundary in the nodal case, all functions ψ_j within the element D^1 are non-zero on the boundary, which results in $(N+1) \times (N+1)$ boundary exchange matrices (see example 2.2.1 below).

Note that the representation of the elemental solution u_h^k in terms of either the functions ℓ_j or ψ_j does not affect the numerical solution. In other words, the evaluation of weak forms and solution of linear systems gives rise to the same solution function. However, the computations to be done and the entries in the vectors are different. The nodal version is usually more efficient because face terms are simpler. In particular in 2D and 3D, face integrals are often the dominating cost in discontinuous Galerkin methods, which explains the popularity of nodal approaches.

It is possible to transform from the modal basis to the nodal basis in nodal points r_i, $i = 0, \dots, N$, by the so-called Vandermonde matrix

$$\mathcal{V}\hat{\boldsymbol{u}} = \boldsymbol{u}, \quad \text{where} \quad \mathcal{V}_{ij} = \psi_j(r_i).$$

The Vandermonde matrix says that the nodal value at r_i is obtained by the sum of the polynomial values ψ_j weighted with the modal contribution $\hat{u}_i$, i.e., the usual evaluation of the polynomial ansatz in the location r_i. To transform from the nodal to the modal basis, $\mathcal{V}^{-1}$ is multiplied with the vector $\boldsymbol{u}$.

2.2.1 Matrix example

Consider the one-dimensional transport equation

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \quad \text{for } x \in \Omega = (0,1),\ t \geq 0, \tag{2.2}$$

with $a = 1$ denoting the constant transport velocity. On the left boundary, an inflow boundary value is given. Consider two sets of third-order polynomials spaces $\{\ell_j, j = 0,1,2,3\}$ and $\{\psi_j, j = 0,1,2,3\}$. The polynomials ℓ_j denote (nodal) Lagrange polynomials in the Gauss–Lobatto points $r_0 = -1$, $r_1 = -\sqrt{\frac{1}{5}}$, $r_2 = \sqrt{\frac{1}{5}}$, $r_3 = 1$. The polynomials ψ_j form a modal basis in terms of the orthonormal Legendre polynomials and are given by

$$\psi_0(r) = \frac{1}{\sqrt{2}}, \qquad \psi_1(r) = \sqrt{\frac{3}{2}}r,$$
$$\psi_2(r) = \sqrt{\frac{5}{8}}(3r^2 - 1), \quad \psi_3(r) = \sqrt{\frac{7}{8}}(5r^3 - 3r).$$

Task: *Derive the matrices $\mathcal{M}$ (mass matrix), $\mathcal{S}$ (convection matrix), and $\mathcal{F}$ (flux matrix) for a discretization with $K = 2$ elements for the coefficients $\boldsymbol{u}$ in the nodal case and $\hat{\boldsymbol{u}}$ in the modal case. Assume an upwind flux.*

Nodal Lagrange polynomials, Gauss–Lobatto points

el. mass matrix
$$\mathcal{M}^1 = \left[\int_0^{0.5} \ell_i \ell_j dx\right]_{i,j} = \begin{bmatrix} 0.0357 & 0.0133 & -0.0133 & 0.00595 \\ 0.0133 & 0.179 & 0.0298 & -0.0133 \\ -0.0133 & 0.0298 & 0.179 & 0.0133 \\ 0.00595 & -0.0133 & 0.0133 & 0.0357 \end{bmatrix} = \mathcal{M}^2$$

mass matrix $\mathcal{M} = \begin{bmatrix} \mathcal{M}^1 & 0 \\ 0 & \mathcal{M}^2 \end{bmatrix}$

$$\mathcal{M} = \begin{bmatrix} 0.0357 & 0.0133 & -0.0133 & 0.00595 & & & & \\ 0.0133 & 0.179 & 0.0298 & -0.0133 & & & & \\ -0.0133 & 0.0298 & 0.179 & 0.0133 & & & & \\ 0.00595 & -0.0133 & 0.0133 & 0.0357 & & & & \\ & & & & 0.0357 & 0.0133 & -0.0133 & 0.00595 \\ & & & & 0.0133 & 0.179 & 0.0298 & -0.0133 \\ & & & & -0.0133 & 0.0298 & 0.179 & 0.0133 \\ & & & & 0.00595 & -0.0133 & 0.0133 & 0.0357 \end{bmatrix}$$

el. advection matrix $\mathcal{S}^1 = \left[\int_0^{0.5} \ell_i \frac{d\ell_j}{dx} dx\right]_{i,j} = \begin{bmatrix} -0.500 & 0.674 & -0.258 & 0.0833 \\ -0.674 & 0 & 0.932 & -0.258 \\ 0.258 & -0.932 & 0 & 0.674 \\ -0.0833 & 0.258 & -0.674 & 0.500 \end{bmatrix} = \mathcal{S}^2$

advection matrix $\mathcal{S} = \begin{bmatrix} \mathcal{S}^1 & 0 \\ 0 & \mathcal{S}^2 \end{bmatrix}$

$$\mathcal{S} = \begin{bmatrix} -0.500 & 0.674 & -0.258 & 0.0833 & & & & \\ -0.674 & 0 & 0.932 & -0.258 & & & & \\ 0.258 & -0.932 & 0 & 0.674 & & & & \\ -0.0833 & 0.258 & -0.674 & 0.500 & & & & \\ & & & & -0.500 & 0.674 & -0.258 & 0.0833 \\ & & & & -0.674 & 0 & 0.932 & -0.258 \\ & & & & 0.258 & -0.932 & 0 & 0.674 \\ & & & & -0.0833 & 0.258 & -0.674 & 0.500 \end{bmatrix}$$

flux matrix $\mathcal{F} = \begin{bmatrix} 0 & & & & & & & \\ & 0 & & & & & & \\ & & 0 & & & & & \\ & & & 1 & & & & \\ & & & -1 & 0 & & & \\ & & & & & 0 & & \\ & & & & & & 0 & \\ & & & & & & & 1 \end{bmatrix}$

The standard (weak) version of the DG problem results in the matrix

$$\mathcal{S}^T - \mathcal{F} = \begin{bmatrix} -0.500 & -0.674 & 0.258 & -0.0833 & & & & \\ 0.674 & 0 & -0.932 & 0.258 & & & & \\ -0.258 & 0.932 & 0 & -0.674 & & & & \\ 0.0833 & -0.258 & 0.674 & -0.500 & & & & \\ & & & 1 & -0.500 & -0.674 & 0.258 & -0.0833 \\ & & & & 0.674 & 0 & -0.932 & 0.258 \\ & & & & -0.258 & 0.932 & 0 & -0.674 \\ & & & & 0.0833 & -0.258 & 0.674 & -0.500 \end{bmatrix}.$$

For the strong form, on the other hand, with elemental equation

$$\left(\frac{\partial u_h}{\partial t}, v\right)_{\mathsf{D}^k} + \left(\frac{\partial f(u_h)}{\partial x}, v\right)_{\mathsf{D}^k} + \left\langle f^*(u_h^-, u_h^+) - f(u_h), \hat{\boldsymbol{n}} v \right\rangle_{\partial \mathsf{D}^k} = (u_h, g)_{\mathsf{D}^k},$$

the flux matrix also contains the additional term $\langle f(u_h), \hat{\boldsymbol{n}} v \rangle_{\partial \mathsf{D}^k}$. Thus, the flux matrix for the strong

form is

$$\mathcal{F}^{\text{strong}} = \begin{bmatrix} 1 & & & & & & & \\ & 0 & & & & & & \\ & & 0 & & & & & \\ & & & 0 & & & & \\ & & & -1 & 1 & & & \\ & & & & & 0 & & \\ & & & & & & 0 & \\ & & & & & & & 0 \end{bmatrix}.$$

Summing the flux matrix with the convection matrix yields

$$-\mathcal{S} - \mathcal{F}^{\text{str}} = \begin{bmatrix} -0.500 & -0.674 & 0.258 & -0.0833 & & & & \\ 0.674 & 0 & -0.932 & 0.258 & & & & \\ -0.258 & 0.932 & 0 & -0.674 & & & & \\ 0.0833 & -0.258 & 0.674 & -0.500 & & & & \\ & & & 1 & -0.500 & -0.674 & 0.258 & -0.0833 \\ & & & & 0.674 & 0 & -0.932 & 0.258 \\ & & & & -0.258 & 0.932 & 0 & -0.674 \\ & & & & 0.0833 & -0.258 & 0.674 & -0.500 \end{bmatrix}.$$

We see that this the final matrix for the strong form (with derivative in front of $f(u_h)$) is exactly the same as for the weak form (with derivative on the test function). This is to be expected for linear problems.

Modal Legendre polynomials

global mass matrix

$$\mathcal{M} = \frac{1}{4}\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{bmatrix}$$

element advection matrix

$$\mathcal{S}^1 = \left[\int_{-1}^{1} \psi_i \frac{d\psi_j}{dr} \underbrace{\frac{dx}{dr}}_{1/4} dr\right]_{i,j} = \begin{bmatrix} 0 & 1.73 & 0 & 2.65 \\ 0 & 0 & 3.87 & 0 \\ 0 & 0 & 0 & 5.92 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \mathcal{S}^2$$

global advection matrix

$$\mathcal{S} = \begin{bmatrix} \mathcal{S}^1 & 0 \\ 0 & \mathcal{S}^2 \end{bmatrix}$$

$$\mathcal{S} = \begin{bmatrix} 0 & 1.73 & 0 & 2.65 & & & & \\ 0 & 0 & 3.87 & 0 & & & & \\ 0 & 0 & 0 & 5.92 & & & & \\ 0 & 0 & 0 & 0 & & & & \\ & & & & 0 & 1.73 & 0 & 2.65 \\ & & & & 0 & 0 & 3.87 & 0 \\ & & & & 0 & 0 & 0 & 5.92 \\ & & & & 0 & 0 & 0 & 0 \end{bmatrix}$$

flux matrix, left

$$\mathcal{F}^{\text{left}} = \left\langle u_h^+ \hat{\boldsymbol{n}}^-, v^- \right\rangle_{\{x^k\}} = \left[-\psi_i^k(-1)\psi_j^{k-1}(1) \right]_{i,j}$$

$$\mathcal{F}^{\text{left}} = \begin{bmatrix} -0.500 & -0.866 & -1.12 & -1.32 \\ 0.866 & 1.50 & 1.94 & 2.29 \\ -1.12 & -1.94 & -2.50 & -2.96 \\ 1.32 & 2.29 & 2.96 & 3.50 \end{bmatrix}$$

flux matrix, right

$$\mathcal{F}^{\text{right}} = \left\langle u_h^- \hat{\boldsymbol{n}}^-, v^- \right\rangle_{\{x^{k+1}\}} = \left[\psi_i^k(1)\psi_j^k(1) \right]_{i,j}$$

$$\mathcal{F}^{\text{right}} = \begin{bmatrix} 0.500 & 0.866 & 1.12 & 1.32 \\ 0.866 & 1.50 & 1.94 & 2.29 \\ 1.12 & 1.94 & 2.50 & 2.96 \\ 1.32 & 2.29 & 2.96 & 3.50 \end{bmatrix}$$

global flux matrix

$$\mathcal{F} = \begin{bmatrix} \mathcal{F}^{\text{right}} & 0 \\ \mathcal{F}^{\text{left}} & \mathcal{F}^{\text{right}} \end{bmatrix}$$

$$\mathcal{F} = \begin{bmatrix} 0.500 & 0.866 & 1.12 & 1.32 & & & & \\ 0.866 & 1.50 & 1.94 & 2.29 & & & & \\ 1.12 & 1.94 & 2.50 & 2.96 & & & & \\ 1.32 & 2.29 & 2.96 & 3.50 & & & & \\ -0.500 & -0.866 & -1.12 & -1.32 & 0.500 & 0.866 & 1.12 & 1.32 \\ 0.866 & 1.50 & 1.94 & 2.29 & 0.866 & 1.50 & 1.94 & 2.29 \\ -1.12 & -1.94 & -2.50 & -2.96 & 1.12 & 1.94 & 2.50 & 2.96 \\ 1.32 & 2.29 & 2.96 & 3.50 & 1.32 & 2.29 & 2.96 & 3.50 \end{bmatrix}$$

Task: *Compare the structure of the matrices in the two cases. In each case, one of the three matrices turns out to have a simple structure. Discuss the impact on computational work for each case.*
By comparing the matrices for the nodal and the modal ansatz, we note the following:

- The mass matrix for the modal ansatz is particularly simple because it is diagonal.
- The advection matrix for the modal ansatz contains a considerable number of zero entries. In particular, all entries $\mathcal{S}_{ij}$ with $i \geq j$ are zero. This is because row i contains the integral of the function ψ_i with the derivative ψ_j'. The function ψ_j' is a polynomial up to degree $j-1$ or zero for $j = 0$. We can represent it as

  $$\psi_j' = \frac{d\psi_j}{dr} = \sum_{k=0}^{j-1} z_k \psi_k,$$

  with some coefficients z_k. However, we know that $\int_{\mathrm{I}} \psi_i \psi_k dr = 0$ for $k < i$ due to the construction of the Legendre polynomials.
- On the other hand, the flux matrix for the modal ansatz is much more dense than for the nodal one. This is because we need to evaluate functions $\psi_i \psi_j$ on the interval end points -1 and $+1$, which are all non-zero. Thus, we obtain dense $(N+1) \times (N+1) = 4 \times 4$ blocks.

Taken together, the increased cost for the flux matrix outweighs the simpler structure in the mass and advection matrices. The most important factor in higher dimensions is that *dense* matrices within the element can be handled more easily than dense matrices connecting one element to its neighbor.
Note that the difference is not very large and it depends on implementation details and the taste of the respective author whether a nodal basis or a modal basis is chosen. Besides the Gauss–Lobatto points, also other nodal sets are in use, depending on the particular application needs. Finally, the modal decomposition gets used also in the nodal case, for example for the definition of certain filters as shown in Chapter 3 below.

2.3 Convergence behavior and stability

Convergence behavior of DG approximations is in many aspects similar to the case with continuous finite elements. With increasing polynomial order, we can expect better convergence for sufficiently smooth solutions: the error decays more rapidly as the grid is refined.
For convergence, two ingredients are used (this is due to the Lax–Richtmyer equivalence theorem):

- Stability of the approximation
- Consistency of the approximation

Consistency & measured convergence rates

Convergence speed is closely related to consistency, i.e., the interpolation properties of polynomials. Consistency is quantified by the following result (course book [Hesthaven & Warburton, 2008], Theorem 4.8, page 83):

$$\|u - u_h\|_\Omega \leq C \frac{h^\sigma}{N^{p-1/2}} |u|_{\Omega,\sigma},$$

where u_h denotes the numerical solution, N the polynomial degree, p the regularity of the analytic solution u (derivatives up to order p of u exist), $\sigma = \min(N+1, p)$, and the norm $|u|_{\Omega,\sigma}$ denotes the norm of all partial derivatives of order σ of u.
The above formula includes the well-known result

$$\|u - u_h\|_\Omega \leq C h^{N+1} |u|_{\Omega,N+1}$$

known from finite elements.
In discontinuous Galerkin methods, there is usually a gap between these consistency results and the actually observed convergence speed. We do not detail the convergence behavior here but rather refer to the course book [Hesthaven & Warburton, 2008], chapter 4, and the references mentioned therein. Simple analysis would suggest rates of $\mathcal{O}(h^N)$, i.e., we lose one order compared to consistency. This estimate is however suboptimal and general results strongly depend on the choice of the numerical flux. For example, rates of order $\mathcal{O}(h^{N+1/2})$ are often reported in the literature, usually due to the specific behavior of the numerical flux (e.g. Lax–Friedrichs flux).
We exemplify the convergence behavior on one of the examples considered before:

$$u(x,t) = \mathrm{e}^{\sin(2\pi(x-t))}, \quad u(x,0) = \mathrm{e}^{\sin(2\pi x)}, \quad \text{periodic boundary conditions.}$$

As a norm, we choose the L_2 norm at the final time $T_f = 10$, i.e., we consider the integral of the square of the difference between u_h and u. This error is more representative than the "approximate" infinity error measured by comparing the values of the solution at the nodes because the node error can be smaller due to superconvergence effects.

K	$N=1$		$N=2$		$N=3$		$N=4$		$N=5$	
	L_2 error	rate	L_2 error	rate	L_2 error	rate	L_2 error	rate	L_2 error	rate
5	$6.90 \cdot 10^{-1}$		$1.53 \cdot 10^{-1}$		$2.14 \cdot 10^{-2}$		$2.32 \cdot 10^{-3}$		$1.92 \cdot 10^{-4}$	
10	$2.57 \cdot 10^{-1}$	1.43	$1.71 \cdot 10^{-2}$	3.16	$5.38 \cdot 10^{-4}$	5.32	$1.82 \cdot 10^{-5}$	7.00	$1.22 \cdot 10^{-6}$	7.31
20	$8.06 \cdot 10^{-2}$	1.67	$8.59 \cdot 10^{-4}$	4.32	$1.31 \cdot 10^{-5}$	5.36	$4.83 \cdot 10^{-7}$	5.24	$1.83 \cdot 10^{-8}$	6.05
40	$1.49 \cdot 10^{-2}$	2.44	$4.61 \cdot 10^{-5}$	4.22	$7.66 \cdot 10^{-7}$	4.10	$1.51 \cdot 10^{-8}$	5.00	$2.87 \cdot 10^{-10}$	5.99
80	$2.28 \cdot 10^{-3}$	2.71	$4.69 \cdot 10^{-6}$	3.30	$4.78 \cdot 10^{-8}$	4.00	$4.73 \cdot 10^{-10}$	5.00	$4.49 \cdot 10^{-12}$	6.00
160	$3.65 \cdot 10^{-4}$	2.64	$5.75 \cdot 10^{-7}$	3.03	$2.99 \cdot 10^{-9}$	4.00	$1.49 \cdot 10^{-11}$	4.99	$3.89 \cdot 10^{-13}$	3.53

The table contains two columns to each degree, where the first value indicates the L_2 error and the second the estimated convergence rate between two successive grid levels: The estimation is based on the formula

$$\text{rate} = \log\left(\frac{\text{error}(h_1)}{\text{error}(h_2)}\right) / \log\left(\frac{h_1}{h_2}\right) \overset{\text{example}}{=} \log\left(\frac{\text{error}(1/5)}{\text{error}(1/10)}\right) / \log(2).$$

It can be seen from the table that the convergence rate can be more than $N+1$ for coarse grid values. In particular, it seems that the case $N = 1$ converges at a faster rate than the expected second order convergence. However, further meshes with $K = 320$ and $K = 640$ show that the rates go down to 2.45 and 2.27, respectively. For $N = 2$, the rates are 3.00 and 3.00 on these finer meshes, in line with the theoretical $N + 1$ rate of convergence. These numbers illustrate that estimation of convergence rate must be done with great care in order to report the correct numbers.
Besides the general estimates stated above, the behavior under p-refinement, i.e., keeping the mesh size constant and increasing the order N of the polynomials (Theorem 4.1 in [Hesthaven & Warburton, 2008]):

$$\|v - v_h\|_{\mathsf{I}} \leq N^{-p}|v|_{\mathsf{I},p}.$$

This estimate states that the convergence is very fast for smooth functions, i.e., functions where high order derivatives p exist. In particular, analytic functions where all derivatives can be formed, this formula states that the convergence in N is actually *exponential*. In terms of the N, the factor in the estimate than becomes $\exp(-N)$.

Stability results

Besides consistency, the main ingredient to a convergent numerical method is its stability, i.e., the boundedness of the discrete operator. A method often used for stability analysis is the **energy method**.
Let us start with the continuous solution and look at the energy in the wave equation,

$$E_u(t) = \int_\Omega u(x,t)^2 dx = \|u(\cdot,t)\|^2_\Omega.$$

Consider now the time derivative of the energy:

$$\frac{dE_u}{dt} = \frac{d}{dt}\|u\|^2_\Omega = \left(u, \frac{\partial u}{\partial t}\right)_\Omega + \left(\frac{\partial u}{\partial t}, u\right)_\Omega = 2\left(u, \frac{\partial u}{\partial t}\right)_\Omega. \tag{2.3}$$

Next, we use the differential equation $\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0$ to substitute the term on the right hand side. This gives the expression

$$-2a\left(u, \frac{\partial u}{\partial x}\right)_\Omega = 2a\left(\frac{\partial u}{\partial x}, u\right)_\Omega - 2a(u^2(1) - u^2(0)),$$

where we used integration by parts for the second equality. We re-arrange the terms in the equation by moving all integral terms to the left hand side and flipping the sign. This gives

$$4a\left(u, \frac{\partial u}{\partial x}\right)_\Omega = 2a(u^2(1) - u^2(0)).$$

Inserting this into equation (2.3) gives the final evolution of the energy:

$$\frac{d}{dt}\|u\|^2_\Omega = -a(u^2(1) - u^2(0)).$$

For $a > 0$, this equation states that the energy decreases by the amount of energy $u^2(1)$ that leaves the domain at $x = 1$ and increases by the inflow part $u^2(0)$ at $x = 0$. The energy method helps us

also determine at which boundaries to set boundary conditions: It is exactly those boundaries where we may have a non-negative term, in this case the inflow value $au^2(0)$.
For the discretized equation, the same technique is applied element-by-element. We assume a problem with **homogeneous Dirichlet condition** on the left, $u(0) = 0$, where the energy in the continuous system above is non-increasing. For the numerical flux

$$f^* = (au)^* = \{\{au\}\} + |a|\frac{1-\alpha}{2}[\![u]\!],$$

the resulting discrete energy is

$$\frac{d}{dt}\|u_h\|_\Omega^2 = -|a|(1-\alpha)\sum_{k=1}^{K}[\![u_h^k(x^{k+1})]\!]^2 - (1-\alpha)a(u_h^1(x^1))^2 - a(u_h^K(x^{K+1}))^2.$$

This equation states that for $0 \leq \alpha \leq 1$, the energy in the system is non-increasing similar to the continuous case. This implies *stability*. We note from this equation that for $\alpha < 1$, there is some negative contribution proportional to the size of the jumps in the DG-FEM solution. This negative contribution acts as a numerical dissipation and illustrates the stabilization mechanism of the method. In the eigenvalue plots shown in Chapter 1, the negative real parts in the eigenvalues can be associated to these jump contributions.
For further details on stability, we refer to Chapter 2 (pages 24–26) and Chapter 4 (pages 85–88) of the course book [Hesthaven & Warburton, 2008].

2.4 Dispersion relations

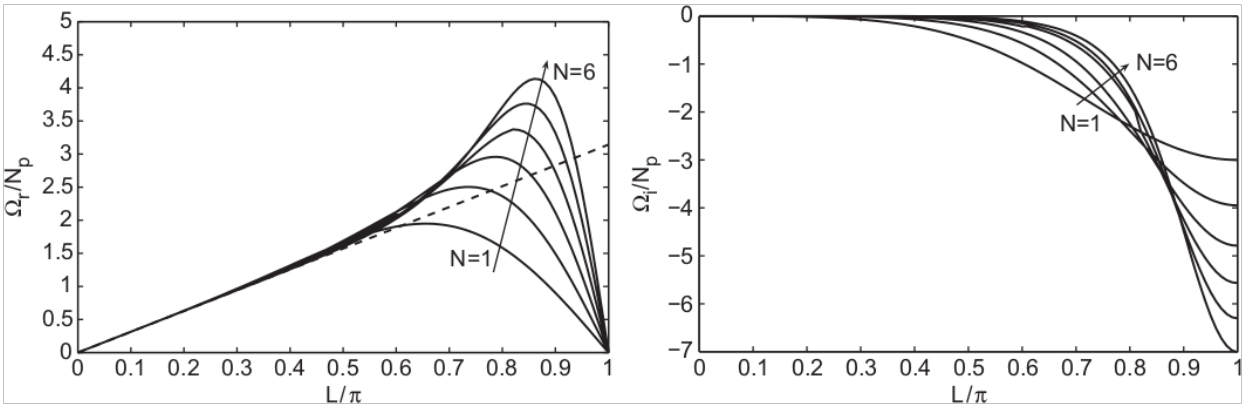
Besides the approximation quality as expressed by convergence rates, the dissipative and dispersive behavior of DG schemes are fundamental for the accuracy in practice. Put shortly, dispersive errors are errors caused by different phase speeds in different components of the solution. Think about the a problem where the exact solution consists of two components,

$$u(x,t) = \sin(2\pi(x-t)) + \sin(20\pi(x-t)).$$

Ideally, we would expect the two components to travel at the same speed. However, due to numerical errors the speed in these two components will in general be different. To this end, the dispersive and the closely related dissipative behavior. In the analysis of numerical schemes, this can be done by applying the discrete DG-FEM operator to the solution

$$u(x,t) = \exp(i(lx - \omega t)),$$

with the exact dispersion relation $\omega = al$. The figure below (reprinted from [Hesthaven and Warburton, 2008], Fig. 4.1, p. 90) shows the dispersive and dissipative behavior (left and right) of DG-FEM:



The figure shows the exact dispersion relation, $\omega = al$, as a dashed line in the left part of the figure, whereas the DG-FEM operator with upwind flux is shown as solid line in the left. For small wave numbers, L/π small, the exact behavior is closely reproduced. Similar, the dissipative behavior shown on the right show that low frequencies (relative to the mesh size) show now damping, whereas higher ones get damped (negative amplification shown in the figure).
For the highly resolved region, the error in the dispersion relation is

$$\left|\mathcal{R}(\tilde{lh}) - \mathcal{R}(lh)\right| \sim \frac{1}{2}\left[\frac{N!}{(2N+1)!}\right](lh)^{2N+3}.$$

This power is considerably higher than the value $N+1$ in the convergence behavior in general. This suggests that in case of nice waves, the main error is made in the interpolation/projection of the initial condition to the polynomials rather than in the propagation of the modes.

2.5 High order methods and CFL restrictions

The time step analysis shown in Chapter 1 suggested a behavior $\Delta t \sim \frac{h}{|a|}$ of the maximal allowed time step size for linear elements. For higher order elements, the full behavior is

$$\Delta t_{\max} = C\frac{h}{N^2|a|},$$

i.e., the maximal allowed time step size must be decreased as the square of the polynomial order N as the order increases. This is because the polynomials on the element boundary get increasingly steep due to the more densely distributed Gauss–Lobatto points for the switch between 0 and 1 in the nodes, which in turn gives rise to larger derivatives. The course book gives several remedies to reduce this effect, including mapping techniques, co-volume grids and local time stepping, see Sec. 4.8 in [Hesthaven & Warburton, 2008].

2.6 Check yourself

- Why are high-order nodal basis functions not based on equidistant node distributions?
- Give a characterization of basis functions on the Gauss–Lobatto nodes.
- Explain the terms “nodal basis” and “modal basis”, respectively.
- What are the computational differences between a nodal basis and a modal basis: Do some matrices exhibit a simpler structure in either method? Why is the nodal approach often preferred in actual implementations?
- What can you say about the rate of convergence in terms of the mesh size h and the polynomial degree N for linear advection problems?
- Regarding stability of the DG-FEM method: The energy in the DG-FEM method is in general a decreasing function in time. What terms contribute to the energy dissipation in the linear advection case?

3 Nonlinear equations

Reading instructions: More details on the content of this lecture can be found in chapter 5 of the course book [Hesthaven and Warburton, 2008].

3.1 General conservation laws

We now consider the general conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0, \qquad x \in (L, R) = \Omega$$
$$u(x, 0) = u_0(x),$$

where $f(u)$ denotes the flux function. Boundary conditions are set on boundaries where

$$\hat{\boldsymbol{n}} \cdot \frac{\partial f}{\partial u} < 0.$$

The name *conservation law* is motivated by the fact that the balance of energy is given solely by the difference between what enters and what leaves any volume $\Omega' = (a, b)$ of interest,

$$\frac{d}{dt}\int_a^b u(x)dx = -\int_a^b \frac{\partial f(u)}{\partial x}dx = f(u(a)) - f(u(b)).$$

In the previous chapters, we discussed the simple equation $f(u) = au$, whereas we now consider the case where the flux function $f(u)$ is **non-linear**.
In this chapter, we will mostly look at **Burgers' equation** with

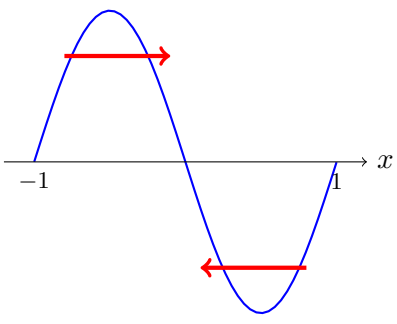
$$f(u) = \frac{1}{2}u^2.$$

With Burgers' equation, the discontinuity gives rise to some ambiguity already when trying to define the solution:

- Consider the case

  $$u_0(x) = -\sin(\pi x) \qquad \text{in } \Omega = (-1, 1) \qquad \text{with } u(\pm 1, t) = 0.$$

  Visualization:



Here, the information in the region $x < 0$ will be transported to the right, whereas the information in $x > 0$ will be transported to the left. Since mass is conserved, the mass will pile up at $x = 0$ and form a shock.

To address this problem, so-called *weak* solutions of the conservation law are defined, i.e., solutions which satisfy

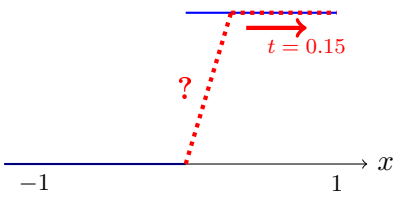
$$\int_0^\infty \int_{-\infty}^{+\infty} \left(u(x,t)\frac{\partial \phi}{\partial t} + f(u)\frac{\partial \phi}{\partial x} \right) dx\, dt = 0,$$

for all test functions $\phi(x,t)$ that are continuous on the domain Ω with weak derivative (i.e., which are in $H^1(\Omega)$ and decay to zero towards ∞). This is similar to what is required for the equations in FEM. The difference is that we have a function ϕ that is smoother (solution space: H^1) than the solution u (solution space: L_2).

- Consider the case

$$u_0(x) = \begin{cases} 0, & x \le 0, \\ 1, & 0 < x, \end{cases} \qquad \text{in } \Omega = (-\infty, \infty).$$

Visualization:



Here, we are facing the opposite problem to the pileup case: The solution moves to the right for $x > 0$ but does not move for $x \le 0$. Thus, new values need to be filled in the intermediate region. This introduces an ambiguity in extending the solution. Two possibilities are

$$u_I(x,t) = \begin{cases} 0, & x \le t/2, \\ 1, & x > t/2 \end{cases} \qquad \text{(Rankine–Hugoniot condition, shock speed } \frac{1}{2}\text{)},$$

and

$$u_{II}(x,t) = \begin{cases} 0, & x \le 0, \\ x/t, & 0 \le x \le t \\ 1 & x \ge t \end{cases} \qquad \text{(classic solution)}.$$

To find the physically relevant solution in such a case of an expansion wave, the concept of the so-called viscosity solution can be used. To this end, the equation

$$\frac{\partial u^\varepsilon}{\partial t} + \frac{\partial f(u^\varepsilon)}{\partial x} = \varepsilon \frac{\partial^2 u^\varepsilon}{\partial x^2}$$

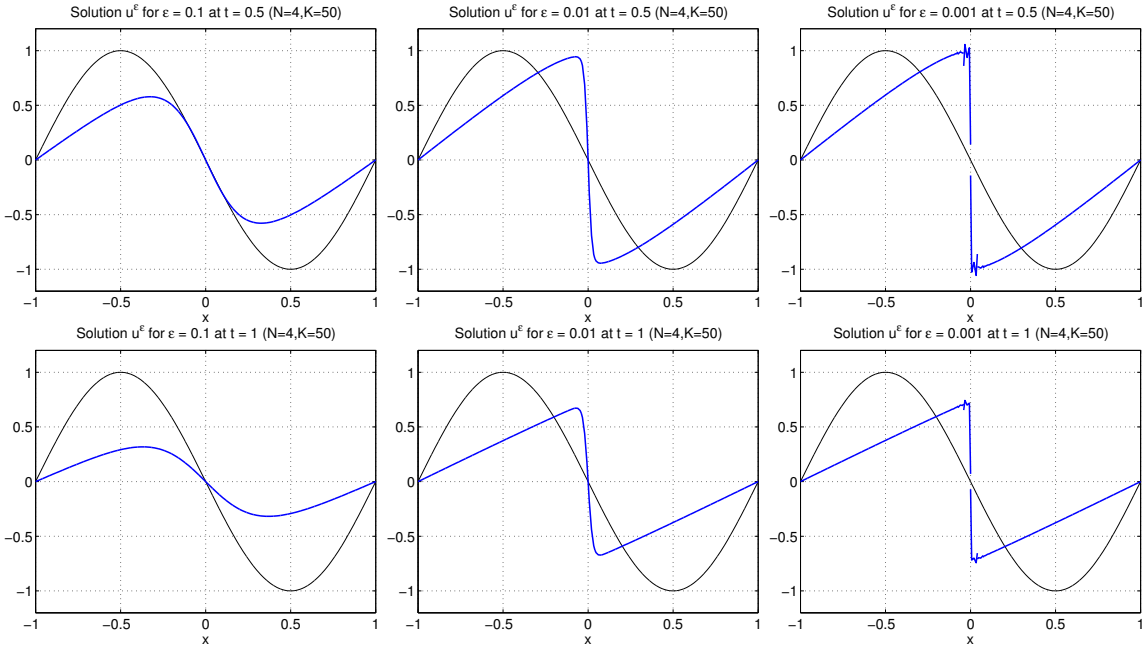
is considered, with $u^\varepsilon(x,0) = u_0(x)$. The physically relevant solution is defined as the limit

$$\lim_{\varepsilon \to 0} u^\varepsilon(x,t) = u(x,t)$$

(in case this limit exists).

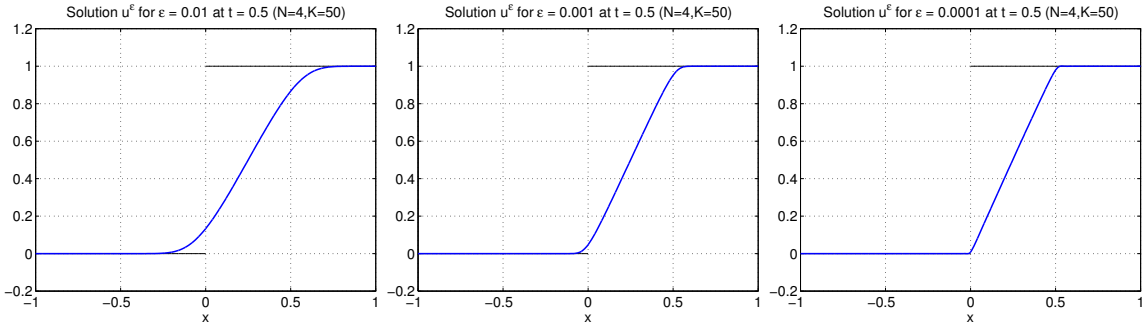
Let us consider this so-called viscosity solution for the two cases above for various choices of ε at a DG-FEM discretization with 50 elements at polynomial order $N = 4$ (we will discuss in Chapter 6 how to treat the second derivative):

- $u_0(x) = -\sin(\pi x)$



Observe the formation of the shock and the dissipation depending on ε. For $\varepsilon = 0.001$, we see some (unphysical) oscillations in the numerical solution around the shock.

- $u_0(x) = \begin{cases} 0, & x \leq 0, \\ 1, & 0 < x : \end{cases}$



We identify that the viscosity solution seems to approach u_{II} above.

A limit to the viscosity solution does not exist for all fluxes. A theory for when solutions exist is provided by entropy conditions, such as the Lax entropy condition. For the case

$$f(u) \qquad \text{convex, i.e.,} \qquad f''(u) > 0 \qquad \text{for all } u,$$

an entropy can be defined by $\eta(u) = \frac{1}{2}u^2$ and it satisfies the differential inequality

$$\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x} F(u) \leq 0,$$

where the entropy flux is

$$F(u) = \int \eta'(v) f'(v) dv.$$

In these formulas, $'$ denotes differentiation of f with respect to u, which we will also write as f_u below. If the flux is non-convex, existence and uniqueness of solutions is elaborate to establish already for scalar problems and the mathematical theory is incomplete. This requires particular care when applying numerical approximations because a problem with sound theory ensures that what we are computing makes sense.

- For non-unique solutions, a slight variation in the numerical method may give a completely different result. Which solution is correct?
- If no theory about the solution behavior exists, it is difficult to distinguish numerical artifacts (e.g. oscillations) from shocks and other effects inherent to the solution.

Further information can be found in the course book [Hesthaven & Warburton, 2008].

3.2 Discontinuous Galerkin schemes for nonlinear conservation laws

Similarly to before consider the following polynomial approximations for the solution and the flux:

$$x \in \mathsf{D}^k: \quad u_h(x,t) = \sum_{i=0}^{N} u^k(x_i,t)\ell_i^k(x), \qquad f_h^k(u_k(x,t)) = \sum_{i=0}^{N} f^k(x_i,t)\ell_i^k(x).$$

The resulting weak form is given by

$$\left(\frac{\partial u_h}{\partial t}, \ell_i^k\right)_{\mathsf{D}^k} - \left(f^k(u_h), \frac{\partial \ell_i^k}{\partial x}\right)_{\mathsf{D}^k} + \left\langle f^*, \hat{\boldsymbol{n}}\ell_i^k\right\rangle_{\partial\mathsf{D}^k} = 0, \tag{3.1}$$

or, in matrix form,

$$\mathcal{M}^k \frac{d}{dt}\boldsymbol{u}_h^k - \mathcal{S}^T \boldsymbol{f}_h^k = -\mathcal{F}^k \boldsymbol{f}^{*,k} = -\left[\boldsymbol{\ell}^k(x) f^*\right]_{x^k}^{x^{k+1}}, \tag{3.2}$$

where the matrix $\mathcal{F}^k$ denotes the flux matrix around element D^k, including contributions from elements D^{k-1} and D^{k+1} (or boundary conditions). In the matrix form, we use vectors of nodal values

$$\boldsymbol{u}_h^k = \left[u_h^k(x_0^k), \ldots, u_h^k(x_N^k)\right]^T, \qquad \boldsymbol{f}_h^k = \left[f_h(x_0^k), \ldots, f_h(x_N^k)\right]^T.$$

For the numerical flux f^*, we generally use the Lax–Friedrichs flux

$$f^*(u_h^-, u_h^+) = \{\{f_h(u_h)\}\} + \frac{C}{2}[\![u_h]\!],$$

where $C = \max\left|\frac{\partial f}{\partial u}\right|$ is an upper bound for the transport velocity (wave speed). For the local Lax–Friedrichs flux, it is typically chosen as the maximum value from $|f_u(u_h^-)|$ and $|f_u(u_h^+)|$, whereas for the global Lax–Friedrichs flux it is some bound on $f_u(u_h)$ globally. Both variants of the Lax–Friedrichs flux are monotone, i.e., they satisfy the inequality

$$\forall v \in [a,b]: \quad (f^*(a,b) - f(v))(b-a) \leq 0.$$

An interpretation of this inequality is that f^* is nondecreasing in the first argument and nonincreasing in the second argument (similar to the derivation of monotone finite volume schemes).

Conservation of mass

The DG-FEM schemes **conserves mass**, i.e., for periodic boundary conditions, we have

$$\frac{d}{dt}\int_\Omega u_h \, dx = 0$$

This can be seen by choosing a test function ϕ that is one throughout the domain Ω. This corresponds to multiplying the weak form by a node vector $\boldsymbol{\phi}_h = [1, \ldots, 1]^T$:

$$\boldsymbol{\phi}_h^T \mathcal{M}^k \frac{d}{dt}\boldsymbol{u}_h^k - \boldsymbol{\phi}_h^T \mathcal{S}^T \boldsymbol{f}_h^k + \boldsymbol{\phi}_h^T \mathcal{F}^k \boldsymbol{f}^{*,k}.$$

The product $\boldsymbol{\phi}_h^T \mathcal{S}^T = (\mathcal{S}\boldsymbol{\phi}_h)^T$ is zero for a constant function $\boldsymbol{\phi}_h$, which gives the local mass balance

$$\frac{d}{dt}\int_{x^k}^{x^{k+1}} u_h dx = f^*\big|_{x^k} - f^*\big|_{x^{k+1}}.$$

Summing over all elements yields

$$\sum_{k=1}^{K} \frac{d}{dt}\int_{x^k}^{x^{k+1}} u_h dx = \sum_{k=1}^{K} \hat{\boldsymbol{n}} \cdot [\![f^*\big|_{x^k}]\!],$$

Each flux term is of the form

$$\hat{\boldsymbol{n}} \cdot [\![f^*\big|_{x^k}]\!] = \hat{\boldsymbol{n}}^{k,-} \cdot \left[\hat{\boldsymbol{n}}^{k,-} \cdot f^*(u_0^k, u_N^{k-1}) + \hat{\boldsymbol{n}}^{k,+} \cdot f^*(u_N^{k-1}, u_0^k)\right] = f^*(u_0, u_N^{k-1}) - f^*(u_N^{k-1}, u_0^k).$$

For the periodic boundary, the corresponding term is $[\![f^*\big|_{x^1}]\!] = \hat{\boldsymbol{n}}^{1,-} \cdot f^*(u_0^1, u_N^K) + \hat{\boldsymbol{n}}^{K+1,+} \cdot f^*(u_N^K, u_0^1)$. We use the notation $\boldsymbol{n}^{1,-}$ to denote the normal on the element D^1 at x^1, which points to the left, i.e., $\hat{\boldsymbol{n}}^{1,-} = -1$. Likewise, $\hat{\boldsymbol{n}}^{K+1,+} = 1$ is the right normal on D^K.
As we have seen in Chapter 1, we construct numerical fluxes such that $f^*(u_0, u_N^{k-1}) = f^*(u_N^{k-1}, u_0^k)$ (single-valued numerical fluxes). Consider for example the Lax–Friedrichs flux:

$$\begin{aligned} f^*(u_h^-, u_h^+) &= \{\{f_h(u_h)\}\} + \frac{C}{2}[\![u_h]\!] = \frac{f(u_h^-) + f(u_h^+)}{2} + \frac{C}{2}\hat{\boldsymbol{n}}^-(u_h^- - u_h^+) \\ &= \frac{f(u_h^+) + f(u_h^-)}{2} + \frac{C}{2}\hat{\boldsymbol{n}}^+(u_h^+ - u_h^-) = f^*(u_h^+, u_h^-), \end{aligned}$$

where we used $\hat{\boldsymbol{n}}^+ = -\hat{\boldsymbol{n}}^-$.
Thus, the term on the right hand side above is zero and mass is conserved.

This derivation shows why single-values numerical fluxes with $f^*(u^-, u^+) = f^*(u^+, u^-)$ are a construction principle: Otherwise, mass would be changed on element boundaries, violating the basic physical intuition.

Energy estimates

The DG-FEM scheme is non-linearly stable for monotone fluxes, i.e.,

$$\frac{1}{2}\frac{d}{dt}\|u_h\|_\Omega \leq 0.$$

Sketch for why this is the case:

- Multiply semi-discrete equation (3.2) by vector $\boldsymbol{u}^h$, which gives

$$\frac{1}{2}\frac{d}{dt}\|u_h^k\|_{\mathsf{D}^k}^2 - \int_{\mathsf{D}^k} f_h^k \frac{\partial u_h^k}{\partial x} dx + \left[u_h^k(x) f^*\right]_{x^k}^{x^{k+1}} = 0$$

- Using the definition $F(u) = \int_u f' v dv$, we obtain after integration by parts

$$\frac{1}{2}\frac{d}{dt}\|u_h^k\|_{\mathsf{D}^k}^2 + \left[F(u_h^k)\right]_{x^k}^{x^{k+1}} + \left[u_h^k(x)(f^* - f_h^k)\right]_{x^k}^{x^{k+1}} = 0$$

- To ensure non-increasing energy, we need to ensure the following relation on each interface x^k:

$$F(u_h^-) - F(u_h^+) - u_h^-(f_h^- - f^*) + u_h^+(f_h^+ - f^*) \geq 0.$$

We apply the mean value theorem (see [Hesthaven & Warburton, 2008, pages 121–122] for details) to the first two terms,

$$F(u_h^-) - F(u_h^+) = u_h^- f(u_h^-) - u_h^+ f(u_h^+) + f(\xi)(u_h^+ - u_h^-),$$

where $\xi \in [u_h^-, u_h^+]$. This gives the condition

$$(f(\xi) - f^*)(u_h^+ - u_h^-) \geq 0,$$

which is nothing else than requiring a **monotone flux** (E-flux).

3.3 Aliasing and filter stabilization

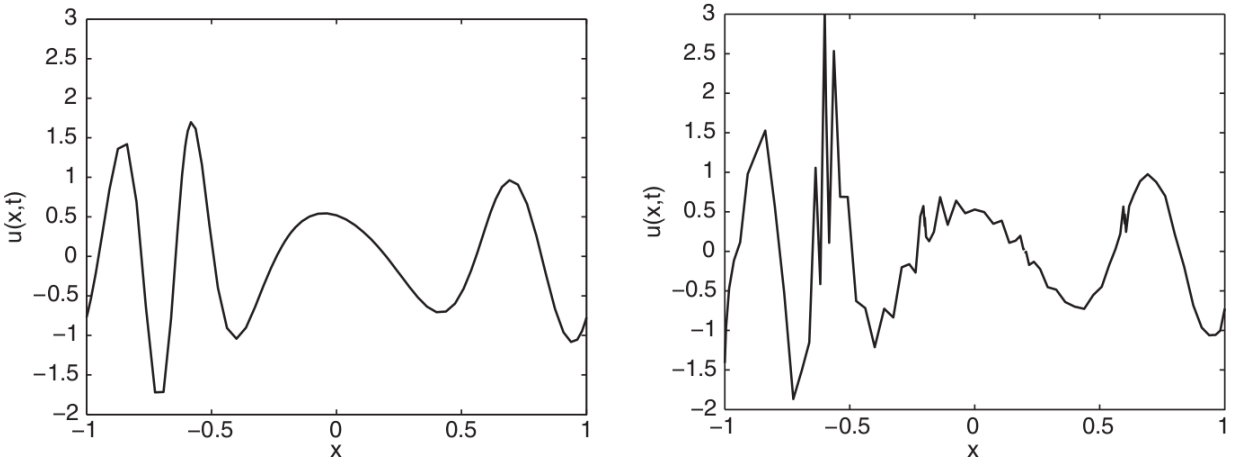
In the previous section, we have not detailed the computation of $\boldsymbol{f}_h^k$ out of $\boldsymbol{u}_h^k$ in the matrix form (3.2). There are two possibilities:

- We find $\boldsymbol{f}_h^k$ by projection of the quantities $f(\boldsymbol{u}_h^k)$ onto polynomials of degree N. This requires the computation of
$$\int_{\mathsf{D}^k} \ell_i f(\boldsymbol{u}_h^k) dx \qquad \text{for } i = 0, \ldots, N,$$
and subsequent multiplication by the inverse mass matrix $\left(\mathcal{M}^k\right)^{-1}$. The evaluation of the integral needs to be done by Gaussian quadrature. For general f, the accurate evaluation can be quite expensive.
- We use the node values of $\boldsymbol{u}_h^k$ to simply set $\boldsymbol{f}_h^k = f(\boldsymbol{u}_h^k) = \left[f(u_0^k), \ldots, f(u_N^k)\right]^T$.

The second definition is the straight-forward to implement. However, there is a subtle problem because $f_h(u_h)$ is in general a polynomial of order higher than N (or not a polynomial) if f is nonlinear or if f involves variable coefficients $a(x)$, so interpolation on $N+1$ points leaves out essential information. This small difference gives rise to problems.
In the figure below (Fig. 5.1 in [Hesthaven & Warburton, 2008]), a variable-coefficient problem with

$$f(u) = a(x)u(x,t), \quad a(x,t) = (1-x^2)^5 + 1 \qquad u(x,0) = \sin(4\pi x)$$

on the domain $\Omega = (-1, 1)$ is considered. For $K = 5$ elements and polynomial degree $N = 16$, the two approaches above give:



The error introduced by interpolation and the possible instability is called **aliasing**. In order to keep the implementation simple, the nodally interpolated version is modified in order to produce stable results. An energy estimate including the aliasing error can be given as

$$\frac{1}{2}\frac{d}{dt}\|u_h^k\|_\Omega^2 \leq C_1\|u_h\|_\Omega^2 + C_2(h,a)N^{1-p}|u|_{\Omega,p}^2$$

The last term on the right hand side can grow large and thus, the energy in the system can grow and cause an instability. It is linked to the difference in

$$\left\|\mathcal{I}_N\frac{dv}{dx} - \frac{d}{dx}\mathcal{I}_N v\right\|,$$

the difference between the derivative of an interpolation $\mathcal{I}_N$ and the interpolation of a derivative. The estimate reveals that

- for u smooth but under-resolved, the problem can be cured by adding resolution
- for non-smooth solutions or if one cannot afford the increased resolution, we can cure the problem by adding artificial dissipation to the equation. More precisely, we want to obtain an error formula of the form
$$\frac{1}{2}\frac{d}{dt}\|u_h^k\|_\Omega^2 \leq C_1\|u_h\|_\Omega^2 + C_2(h,a)N^{1-p}|u|_{\Omega,p}^2 - C_3\varepsilon\|u_h\|_{\Omega,s}^2,$$
where we want to choose $\varepsilon \propto N$ and for some derivative order s. This can be realized by so-called filtering.

Filtering

Filters add dissipation in a particular way through the modified equation

$$\frac{\partial u_h}{\partial t} + \frac{\partial}{\partial x}\mathcal{I}_N f(u_h) = \varepsilon(-1)^{\tilde{s}}\left[\frac{\partial}{\partial x}(1-x^2)\frac{\partial}{\partial x}\right]^{\tilde{s}} u_h$$

The weighting function $(1-x)^2$ is used to avoid additional boundary conditions at the boundary $\partial\Omega = \{\pm 1\}$.
For implementing this method, we want to first advance the equation

$$\frac{\partial u_h}{\partial t} + \frac{\partial}{\partial x}\mathcal{I}_N f(u_h) = 0$$

by one time step and next do one time step of

$$\frac{\partial u_h}{\partial t} = \varepsilon(-1)^{\tilde{s}}\left[\frac{\partial}{\partial x}(1-x^2)\frac{\partial}{\partial x}\right]^{\tilde{s}} u_h.$$

In this form, we would only obtain $\mathcal{O}(\Delta t)$ accuracy. Remember that the dissipative term is only added for stability reasons and it should not affect the convergence order. Therefore, we introduce an additional approximation: The high order derivative will mostly affect the high frequencies in the local solution expansion. In terms of the *modal* expansion from Chapter 2, we want to construct the following filtering:

$$u_h^*(x,t) \approx \sum_{i=0}^{N} \sigma\left(\frac{i}{N}\right)\hat{u}_i(t)\psi_i(x),$$

where $\hat{u}_i(t)$ are the modal coefficients and ψ_i the normalized Legendre polynomials. We want the following properties for the *damping function*:

$$\sigma(\eta) = \begin{cases} = 1, & \eta = 0, \\ \leq 1, & 0 < \eta < 1 \quad \eta = \frac{i}{N}, \\ \geq 0, & \eta = 1. \end{cases}$$

Furthermore, the damping function is assumed monotone, i.e., lower frequencies should not get damped more than higher ones. This choice is a low-pass filter that keeps the low polynomial orders unchanged, whereas high orders get reduced.
Typical choices for σ are:

- $\sigma(\eta) = 1 - \alpha\eta^{2\tilde{s}}$
- $\sigma(\eta) = \exp(-\alpha\eta^{2\tilde{s}})$ (exponential filter) with coefficient $\alpha \approx 36$ (natural logarithm of machine precision $\rightarrow \sigma(1)$ is numerically 0)
- $\sigma(\eta) = \begin{cases} = 1 & 0 \leq \eta < r \\ = 0 & r \leq \eta \leq 1 \end{cases}$ (box filter), where r is between 0 and 1 (e.g. $\frac{2}{3}$).

In terms of the node values $\boldsymbol{u}_h^k$, the filtering operation is implemented by the operation:

$$\boldsymbol{u}_h^{k,*} = \mathcal{V}\Lambda\mathcal{V}^{-1}\boldsymbol{u}_h^k,$$

where the diagonal matrix Λ has the entries

$$\Lambda_{ii} = \sigma\left(\frac{i-1}{N}\right), \quad i = 1, \ldots, N+1.$$

In general, one tries to construct filters such that at least the lower half of the polynomial degrees remains unaffected.

Aliasing due to non-exact quadrature

A related aliasing problem appears when the integrals in the weak form (3.1) are directly approximated by numerical quadrature using Gauss rules. For linear coefficients, one often uses Gauss rules using $N+1$ points to approximate weak forms of polynomials up to degree N because this ensures that products of two polynomials, involving up polynomial degrees up to order $2N$, are integrated exactly. For efficiency reasons, one is often tempted to also evaluate non-linear or variable-coefficient terms that combined with the basis functions give degrees larger than $2N+1$ with Gauss rules of $N+1$ points. Often, such integration errors can be neglected, for example in case there is enough diffusion in the system. However, in case of problems with little diffusivity and stronger shocks, we again need to take aliasing effects into account. For the term $f(u) = u^2$ of Burgers equation, appearing also in many equations of fluid dynamics (Navier–Stokes equations, Euler equations), one typically selects one of two strategies:

- Integrate the convective term with $\frac{3N}{2} + 1$ (integer division rounded up) Gauss points to ensure exact integration of terms $\int_{\mathsf{D}^k} \frac{dv}{dx} u^2 dx$ (so-called $\frac{3}{2}$-rules)
- Filter away the highest third of the frequencies, because then all terms in $\int_{\mathsf{D}^k} \frac{dv}{dx} u^2 dx$ have maximal degree $\frac{2}{3}N$, thus giving a term with degree of at most $2N$ (so-called $\frac{2}{3}$-filtering)

Both strategies involve a **numerical cost**:

- More accurate integration increases the work to compute the integrals (in 2D and 3D both for element and face integrals)
- Filtering out the highest third of the frequencies introduces "unused" coefficients, resulting in a cost overhead as for the more accurate integration.

3.3.1 Example: Effect of filters as the time step varies

Consider the modified exponential filter

$$\sigma(\eta) = \begin{cases} 1, & 0 \le \eta \le \eta_c = \frac{1}{2}, \\ \exp(-\alpha((\eta-\eta_c)/(1-\eta_c))^s), & \frac{1}{2} < \eta \le 1, \end{cases}$$

for s even. This implements the exponential filter on the upper half of the frequencies. This filter is applied in the following numerical algorithm:

- Burgers' equation in 1D is discretized.
- We use a fourth-order Runge–Kutta method for time integration.
- In each stage of the Runge–Kutta method, a right hand side is evaluated from the advection and flux terms and by multiplication with the inverse mass matrix. After computing a vector of these quantities, the filter is applied by
$$\boldsymbol{u}_h^{k,*} = \mathcal{V}\Lambda\mathcal{V}^{-1}\boldsymbol{u}_h^k,$$
where the diagonal matrix Λ has the entries
$$\Lambda_{ii} = \sigma\left(\frac{i-1}{N}\right), \quad i = 1, \dots, N+1.$$

Task: *What happens with the strength of the filter if the time step size is reduced? Does it damp more, the same, or less of the solution over a fixed time span?*
Let us first consider the discretization of the 1D transport equation on the interval $[0, 2\pi]$ with constant transport velocity $a = 2\pi$ and boundary condition $u(0,t) = -\sin(2\pi t)$ in order to understand how the filter works.
The time step size is chosen according to the formula

$$dt = \frac{\mathrm{CFL}}{2\pi} x_{\min},$$

where $x_{\min} = (r_2 - r_1)h^{\min}$ is the minimum distance between two Gauss–Lobatto points on the whole mesh, with $r_2 - r_1$ denoting the minimum distance between two Gauss–Lobatto points $r_1 = -1$ and r_2 for a given polynomial degree.
We set CFL = 0.375. Consider first the **maximum pointwise error** at final time $T = 1$ for a few polynomial degrees without any filtering (note that the time step error dominates for larger polynomial degrees):

K	$N = 2$	$N = 5$	$N = 8$
5	$1.0 \cdot 10^{-1}$	$3.5 \cdot 10^{-5}$	$1.1 \cdot 10^{-7}$
10	$2.5 \cdot 10^{-2}$	$1.2 \cdot 10^{-6}$	$1.4 \cdot 10^{-8}$
20	$6.6 \cdot 10^{-3}$	$5.9 \cdot 10^{-8}$	$1.8 \cdot 10^{-9}$
40	$1.6 \cdot 10^{-3}$	$4.5 \cdot 10^{-9}$	$2.3 \cdot 10^{-10}$

Let us include the exponential filter

$$\sigma(\eta) = \begin{cases} 1, & 0 \le \eta \le \eta_c = \frac{1}{2}, \\ \exp(-\alpha((\eta - \eta_c)/(1 - \eta_c))^s), & \frac{1}{2} < \eta \le 1, \end{cases}$$

using $s = 16$ in the computations, involving the same time step size:

K	$N = 2$	$N = 5$	$N = 8$
5	$5.5 \cdot 10^{-1}$	$2.4 \cdot 10^{-3}$	$1.4 \cdot 10^{-6}$
10	$3.4 \cdot 10^{-1}$	$3.1 \cdot 10^{-4}$	$2.6 \cdot 10^{-8}$
20	$1.9 \cdot 10^{-1}$	$3.8 \cdot 10^{-5}$	$8.7 \cdot 10^{-10}$
40	$1.0 \cdot 10^{-1}$	$4.8 \cdot 10^{-6}$	$8.7 \cdot 10^{-11}$
80	$5.1 \cdot 10^{-2}$	$6.0 \cdot 10^{-7}$	$4.9 \cdot 10^{-11}$

This filter is implemented in the file `Codes1D/Advec1D.m` (in the MATLAB files accompanying the course book) by adding the following code to each step of the Runge–Kutta method:

```
F = Filter1D(N,floor(N/2),16,V,invV);
resu = F*resu;
u = F*u;
```

We observe considerably larger errors with the filter for $N = 2$ and $N = 5$. For $N = 2$, the convergence is degraded to first order. Note that the filter completely removes the quadratic components of the solution. Also for $N = 5$, we also observe considerably worse convergence rates (approximately third order instead of more than fourth).
Let us see what happens if we set CFL $= 0.1$, i.e., we use almost four times as many time steps and we would expect an increase in accuracy due to smaller time errors:

K	$N = 2$	$N = 5$	$N = 8$
5	$5.5 \cdot 10^{-1}$	$5.1 \cdot 10^{-3}$	$1.8 \cdot 10^{-6}$
10	$3.4 \cdot 10^{-1}$	$6.7 \cdot 10^{-4}$	$3.4 \cdot 10^{-8}$
20	$1.9 \cdot 10^{-1}$	$8.4 \cdot 10^{-5}$	$8.1 \cdot 10^{-10}$
40	$1.0 \cdot 10^{-1}$	$1.0 \cdot 10^{-5}$	$4.3 \cdot 10^{-11}$
80	$5.1 \cdot 10^{-2}$	$1.3 \cdot 10^{-6}$	$1.5 \cdot 10^{-10}$

We observe that the errors for $N = 5$ and $N = 8$ increase, despite using a smaller time step. On the other hand, the error for $N = 2$ is unaffected.

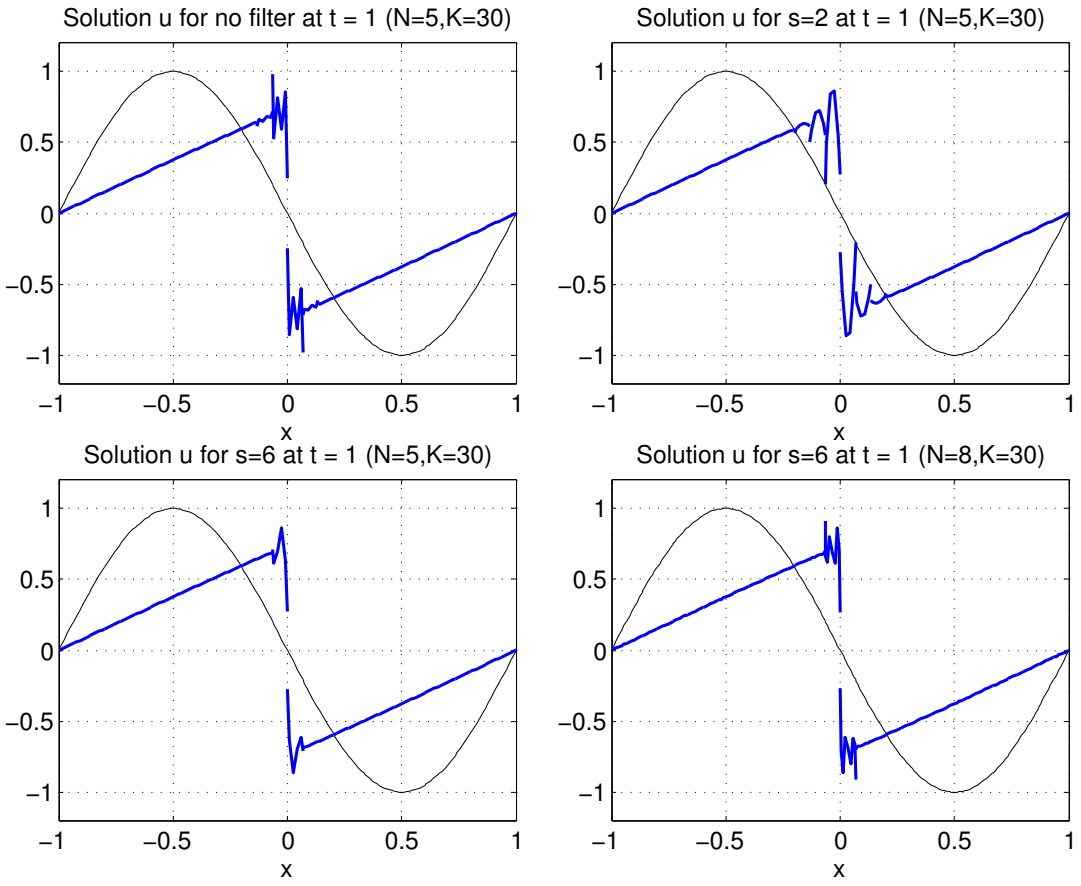
Explanation

- The filter is applied in every stage of the Runge–Kutta method. If the time step becomes smaller, the filter is applied more often when trying to reach a given time. Since the exponential filter decreases the amplitude of the higher frequencies by a factor smaller than one, more filter applications correspond to stronger damping in high frequencies.
- This behavior is linked to the fact of applying a filter to the same function twice: In the second application, it will continue to decrease the amplitudes of the higher modes.
- For $N = 2$, nothing happens because the filter is constructed such that the highest frequency, polynomial degree 2, is completely damped out in one step. In other words, it already acts like a box (sharp cutoff) filter.

Task: *The sharp cutoff filter with $\sigma(\eta) = 0$ for $\eta > \frac{1}{2}$ works independent of the time step size. Why?*

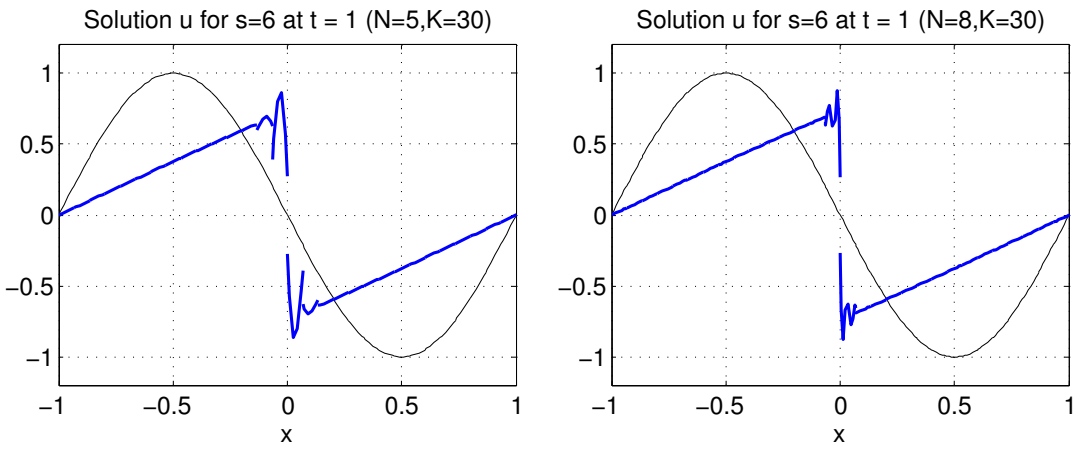
- A box filter has the particular property that any application on a function beyond the initial one does not change the results. Such a filter is called idempotent. It can be shown that such a filter is time-consistent in the sense that it acts the same for all sizes of the time step.
- An exponential filter can be made time-consistent by making the strength of the filter depend on the CFL number. We put more strength when the CFL number is larger.

Let us finally look at the situation with Burgers' equation with $K = 30$, starting with the sine solution. We start with CFL $= 0.25$ and look at solutions with various strengths s of the filter for $N = 5$ (first three figures) and $N = 8$ (fourth figure):

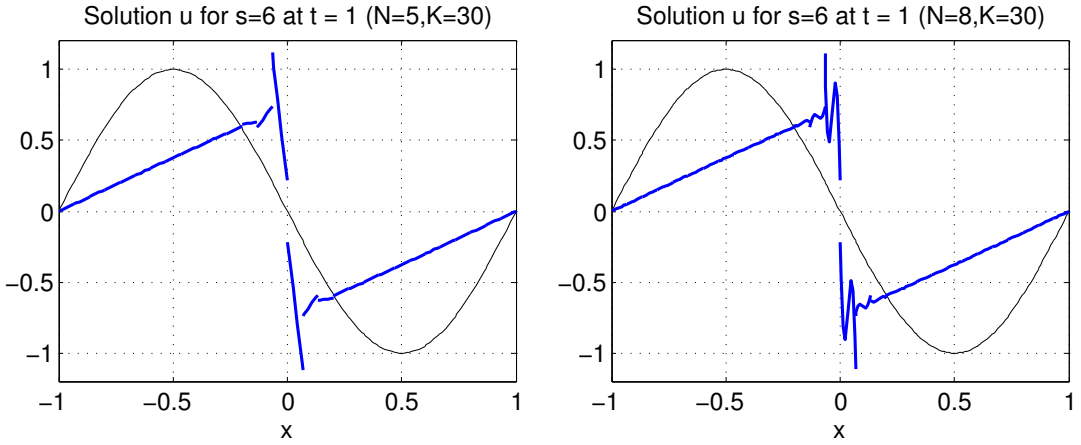


The figures show that the frequency of oscillations is considerably damped, but oscillations are not removed altogether.
For the case with $N = 8$, the original method without filtering is not stable and produces a solution that explodes (due to both aliasing problems and the discontinuity).

Let us look at the final two pictures again for CFL $= 0.05$:



In particular for $N = 8$, we observe smaller oscillations. Finally, we choose a box filter at CFL $= 0.25$ that completely annihilates all polynomial orders of two and higher for $N = 5$ as well as four and higher for $N = 8$:



This filter acts more strongly. In particular for $N = 5$, it has annihilated all components except the constant and linear ones. Note that we choose the cutoff at less $\frac{1}{2}$ in this case to illustrate the effect on the solution. For cutoff at $\frac{1}{2}$, the solution looks very similar to the case with CFL $= 0.05$ (as expected).

3.4 DG-FEM with shocks

As we have seen in the introduction to this chapter (and for some experiments in Chapter 1), non-linear problems often give rise to shocks. To cope with shocks, two strategies are often applied, **filtering** and **limiting**.

Filtering

As we have seen above, this strategy reduces the effect of large oscillations and can make computations stable. The main ingredient to filtering is to sacrifice some of the accuracy by reducing amplitudes in high polynomial degrees. It turns out that many problems allow for applying filters only sporadically, for which the loss in accuracy is usually not very prominent. In terms of the solution, the filtering smears out discontinuities to some extent. If the smearing is not too strong, this results in a considerable improvement of the situation, even if the accuracy is reduced. Of course, excessive smearing is undesired.

Solutions with shocks resolved by high-order discretizations contain oscillations due to Gibb's phenomenon. This looks as if **the best convergence** we can get is **first order**.

Fortunately, it turns out that the highly oscillatory solution contains information to reconstruct very accurate solutions:

- Using special filtering techniques
- Initial conditions with shocks must be read in carefully: Typically, need to project to the solution space, rather than interpolating at nodes directly
- Known for Burgers' equation
- See course book for more information

Limiting

While filtering allows to reduce the amount of oscillations and can make simulations stable, sometimes even small oscillations are detrimental. Since only very strong and low-accuracy filters affecting all but the element-wise constants can guarantee this, an alternative is necessary. It is given by so-called *slope limiters*. Problems with strong jumps in the solution values such as the Euler equation of fluid dynamics need this.

- The compressible Euler equations posed in conservation form

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) &= 0, \\ \frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p \boldsymbol{I}) &= \rho \boldsymbol{g}, \\ \frac{\partial E}{\partial t} + \nabla \cdot (\boldsymbol{u}(E + p)) &= 0, \end{aligned}$$

  formulated in density ρ, velocity $\boldsymbol{u}$ and energy E are often solved for the variables

$$\boldsymbol{y} = \begin{bmatrix} \rho \\ \rho u \\ E \end{bmatrix}.$$

  For computing the actual velocity $\boldsymbol{u}$ from the stored quantity $\rho \boldsymbol{u}$ as well as computing the pressure p through an equation of state (e.g. $p = (\gamma - 1)(E - \frac{1}{2}\rho \boldsymbol{u} \cdot \boldsymbol{u})$ for an ideal gas), we must make sure that the density remains positive.

- This can be difficult if there are strong jumps, e.g. between 1000 and 1, because oscillations as small as 0.1% lead to undershoots into the negative range.

To ensure that the solution stays within bounds, so-called **limiting** can be used. The idea of limiting is to modify the solution after each time step or Runge–Kutta stage specifically with the aim to reduce oscillations to as large an extent as possible. Ideally, one would like the numerical solution to satisfy the inequality

$$\frac{d}{dt} \left\| \frac{\partial u_h}{\partial x} \right\|_{\mathrm{L}^1} \leq 0,$$

i.e., the derivative of the solution u should be non-increasing. This is a requirement on a bound of the total variation of the solution and one calls schemes that satisfy such a relation **total variation diminishing (TVD).**
This requirement can be realized by a slope limiter according to the three properties:

- It should not violate conservation.

- The mean value of the solution on each element should be uniformly bounded, which can be expressed in terms of conditions on the right hand side in the evaluation of the differential operator (see pages 147–148 in [Hesthaven & Warburton, 2008]).
- It should not change the formal accuracy of the method.

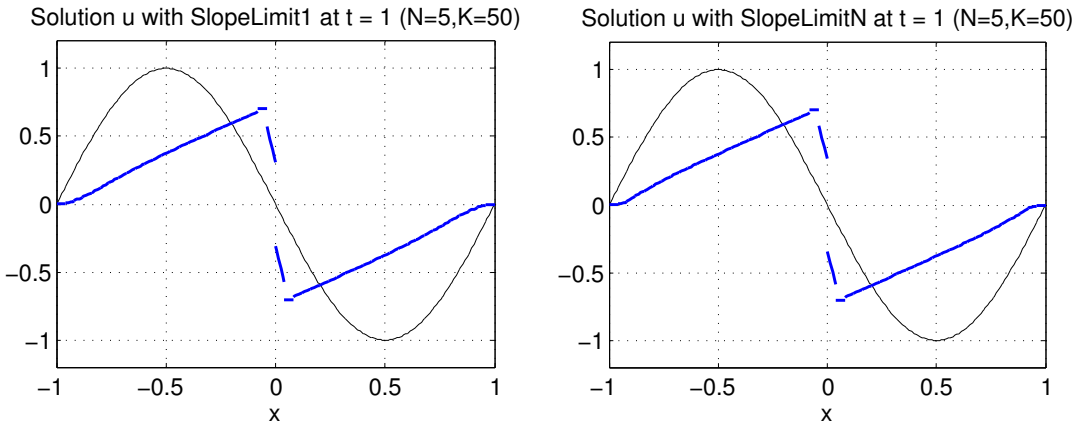
However, such strong requirements lead to severe smearing and destroy the good properties in smooth parts of the solution. In particular, slope limiters typically get enabled close to local extrema of the solution, reducing the formal accuracy to first order.
We do not go into the details of slope limiters here. The course book [Hesthaven & Warburton, 2008], pages 145–157, discusses these issues in more details. The basic idea of the so-called minmod limiter (called Π^1) is to

- check the slopes of the numerical solution u_x (obtained by computing the derivative of the DG-FEM solution functions in space) and
- to set the slope to zero in case the slope changes inside the element, or
- to set the slope to the minimum inside of the element in case the sign is the same in all nodes

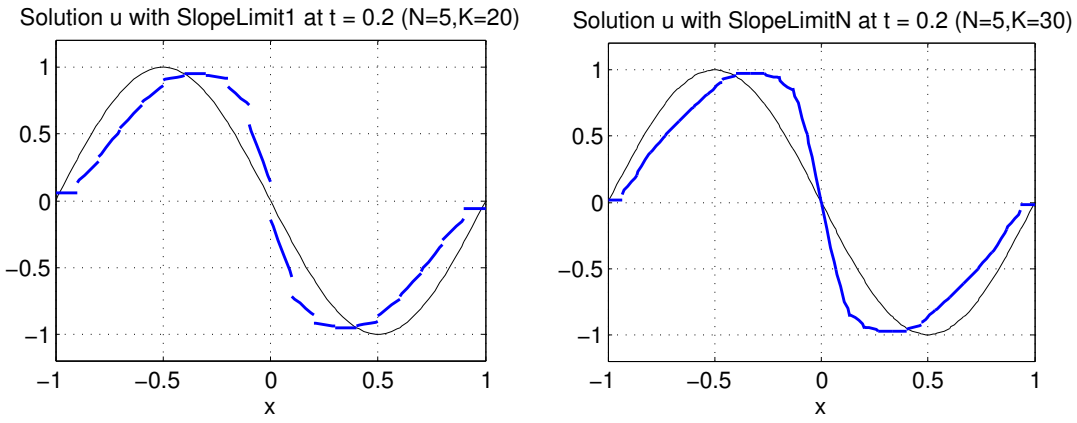
There are also other options such as MUSCL (Monotone Upstream-centered Scheme for Conservation Laws). Most of the methods have their origin in finite volume methods. One possible extension to DG-FEM is the generalized slope limiter Π^N which works as follows:

- check cell averages of the solution and if they need to be limited (like in finite volumes),
- if no limitation necessary, leave the solution as is, else
- choose linear interpolation between the finite-volume limit values inside the elements.

In the following figure, we show the action of a slope limiter on the shock formation in Burgers' equation with initial condition $u_0(x) = -\sin(\pi x)$, for both the minmod limiter Π^1 and the more advanced slope limiter Π^N:



We observe a much smoother solution now as compared to the filters. However, we also see a degradation of accuracy at the end points of the interval (the derivative becomes zero) and overall a solution that looks like a linear interpolation despite the use of polynomial degree $N = 5$.
To appreciate the difference between Π^1 and Π^N, let us look at the same problem at $t = 0.2$ and with fewer elements before the shock forms:



We clearly see that the high-order character of the solution is lost for the Π^1 operator as the interpolation appears linear (reconstructed between the finite volume limits from neighboring cells). For Π^N, the situation is generally better. However, the method still considerably changes the solution near the maximum.

One can further improve the situation by giving up on the strict TVD property and instead require the variation to be bounded (TVB), where one allows some increase and decrease. The limiter `CFD1D/minmodB.m` of the codes is one such example.

Other approaches to limiting discussed in the literature of high-order DG methods are to represent the high-order solution inside an element by several subcells with finite volume discretizations, apply finite-volume limiters on a subgrid, and transfer the limited representation back to polynomials.

Note: Oscillation-diminishing techniques must be combined with suitable Runge–Kutta methods to not destroy the accuracy, so-called strong stability-preserving Runge–Kutta methods. These have coefficients such that the boundedness in one stage carries over to the full time step, i.e., adding contributions in the Runge–Kutta sense does not destroy the result of limiting.

3.5 Check yourself

- Two effects appear for the nonlinear inviscid Burgers equation that are not present in the linear case. Explain these and how an ideal numerical method should cope with these challenges.
- Explain how the DG-FEM matrices (mass, advection, flux) derived for linear transport can be used for the nonlinear Burgers equation.
- State the Lax–Friedrichs flux for Burgers' equation in 1D in terms of u_h^- and u_h^+.
- What is aliasing? Give two examples of discretizations that are susceptible to aliasing.
- What methods are applied for counteracting aliasing?
- Which techniques are used when shocks form in the numerical solution? Give a short explanation of how these methods work.
- How to approach a problem where shock is only local and the remainder is smooth?

4 Formulations for higher dimensions

Reading instructions: More details on the content of this chapter can be found in chapter 6 of the course book [Hesthaven and Warburton, 2008] (the book concentrates on algorithms for triangles and tetrahedra).

4.1 Derivation of DG-FEM discretization

We consider a scalar conservation law in dimensions $d = 2, 3$ of the form

$$\begin{aligned} &\frac{\partial u(\boldsymbol{x},t)}{\partial t} + \nabla \cdot \boldsymbol{f}(u(\boldsymbol{x},t),\boldsymbol{x},t) = 0, && \boldsymbol{x} \in \Omega \subset \mathbb{R}^d, \\ &u(\boldsymbol{x},t) = g(\boldsymbol{x},t), && \boldsymbol{x} \in \partial\Omega_i, \\ &u(\boldsymbol{x},0) = u_0(\boldsymbol{x}), \end{aligned}$$

where $\boldsymbol{f}$ denotes the vector-valued flux function. Boundary conditions are set on all inflow boundaries $\partial\Omega_i$ where the flux Jacobian satisfies

$$\hat{\boldsymbol{n}} \cdot \frac{\partial \boldsymbol{f}}{\partial u} < 0.$$

We assume that a triangulation

$$\Omega_h = \bigcup_{k=1}^{K} \mathsf{D}^k$$

covers the domain Ω.

We assume the elements D^k to be triangles or quadrilaterals (2D) and tetrahedra or hexahedra (3D). For most discussions, we assume the triangles to be straight-sided, such that the boundary is covered by a linear polygon. The extension to curvilinear sides is straight-forward by including polynomial transformations of the element shape to the boundary shape in complete analogy to finite elements. For further details, we refer to the course book, chapter 9 [Hesthaven & Warburton, 2008] (pages 373–406).

As in 1D, we assume the local solution u_h^k to be a weighted sum of polynomials on the element D^k,

$$u_h^k(\boldsymbol{x},t) = \sum_{i=1}^{N_p} u_h^k(\boldsymbol{x}_i^k,t)\ell_i^k(\boldsymbol{x}), \tag{4.1}$$

where $u_h^k(\boldsymbol{x}_i^k,t)$ denotes the value of u_h in node $\boldsymbol{x}_i^k$ and $\ell_i^k(\boldsymbol{x})$ is a Lagrange polynomial of degree N which evaluates to one in the node $\boldsymbol{x}_i^k$ and to zero in all other nodes of the element $\boldsymbol{x}_j^k, j \neq i$. Globally, the element-wise solutions are combined into the solution u_h.

The local statement of the DG-FEM is analogous to 1D:

$$\int_{\mathsf{D}^k} \left[\frac{\partial u_h^k}{\partial t}\ell_j^k(\boldsymbol{x}) - \boldsymbol{f}_h^k \cdot \nabla \ell_j^k(\boldsymbol{x}) \right] d\boldsymbol{x} = -\int_{\partial \mathsf{D}^k} \hat{\boldsymbol{n}} \cdot \boldsymbol{f}^* \ell_j^k(\boldsymbol{x}) d\boldsymbol{x} \tag{4.2}$$

for the weak form and

$$\int_{\mathsf{D}^k} \left[\frac{\partial u_h^k}{\partial t} + \nabla \cdot \boldsymbol{f}_h^k \right] \ell_j^k(\boldsymbol{x}) d\boldsymbol{x} = \int_{\partial \mathsf{D}^k} \hat{\boldsymbol{n}} \cdot \left[\boldsymbol{f}_h^k - \boldsymbol{f}^* \right] \ell_j^k(\boldsymbol{x}) d\boldsymbol{x}$$

for the strong form of the nodal discontinuous Galerkin method.
The numerical flux $\boldsymbol{f}^*$ can be computed in various ways. As before, we will mostly focus on the local Lax–Friedrichs flux

$$\boldsymbol{f}^*(u_h^-, u_h^+) = \frac{\boldsymbol{f}(u_h^-) + \boldsymbol{f}(u_h^+)}{2} + \frac{C}{2} \hat{\boldsymbol{n}} (u_h^- - u_h^+),$$

where u_h^- and u_h^+ are the interior and exterior solution values (taken pointwise in the same physical location of the evaluation point on the face) and C is the local maximum of the flux Jacobian,

$$C = \max_{u \in [u_h^-, u_h^+]} \left| \hat{n}_x \frac{\partial f_1}{\partial u} + \hat{n}_y \frac{\partial f_2}{\partial u} \right| = \max_{u \in [u_h^-, u_h^+]} \left| \hat{\boldsymbol{n}} \cdot \frac{\partial \boldsymbol{f}}{\partial u} \right|$$

for $\boldsymbol{f} = (f_1, f_2)$.
Like for continuous finite elements, two main sets of element shapes are widely used in the context of DG-FEM:

Triangular/tetrahedral elements. The main advantage of triangular elements is the ease of mesh generation. Off-the-shelf mesh generators are often sufficient to create high-quality meshes with good statistics (good aspect ratios, no distorted element shapes). Moreover, nodal triangular elements have the advantage that "linear" basis functions with $N = 1$ only involve the monomials $1, x, y$ up to linear and no mixed-quadratic terms. This reduces the number of unknowns to reach a certain polynomial degree, in particular in 3D: A hexahedral basis with $N = 4$ consists of $5^3 = 125$ basis functions (up to tensor degree 4), whereas the polynomials up to degree 4 only involve $5 \cdot 6 \cdot 7/6 = 35$ terms. Conversely, 120 basis functions per element on tetrahedra offer polynomial degree $N = 7$ (and the associated higher convergence rates).

Quadrilateral/hexahedral elements. For the same number of degrees of freedom and given sufficient mesh quality, hexahedral elements are typically more accurate. Despite more degrees of freedom per element, the considerably larger volume of hexahedra usually more than compensates for the increased number of degrees of freedom. In addition, hexahedral meshes are easily generated for boundary layers (e.g. extruding a mesh from a surface), even though this is often manual labor. Finally, evaluation of tensor-product hexahedral shape functions is faster than for tetrahedra when using tensorial evaluation, i.e., the work per degree of freedom is typically less.

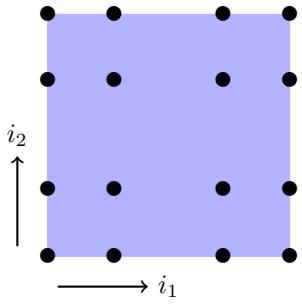
Besides these elements, mixed tetrahedral/hexahedral meshes (with wedges and pyramids in the transition region) or even meshes with other topology (e.g. six-sided elements in 2D) are easily possible with DG as no continuity requirements over faces, edges, and vertices must be satisfied.

4.2 DG-FEM on quadrilaterals

For quadrilaterals, we construct the shape functions $\ell_i^k(\boldsymbol{x}) = \ell_i^k(x, y)$ by a **tensor product** of 1D functions $\ell_{i_1}^k(x)$ and $\ell_{i_2}^k(y)$ as follows:

$$\ell_i^k(x, y) = \ell_{i_1 + (N+1) i_2}^k(x, y) = \ell_{i_1}^k(x) \ell_{i_2}^k(y),$$

for the index $i = i_1 + (N + 1) i_2$ of the shape functions.
In this formula, we assume a *lexicographic* numbering of degrees of freedom within the elements, where i_1 goes from 0 to N and i_2 goes from 0 to N. Thus, the index i for the degrees of freedom on element D^k goes from 0 to $N_p = (N + 1)^2$ (exclusive), as represented here for $N = 3$:



For the one-dimensional node distributions, we use the Gauss–Lobatto node points from 1D in each of the coordinate directions.
We insert these polynomial functions into the representation (4.1) and the integrals for DG-FEM. As opposed to the 1D case where all the matrices $\mathcal{M}$, $\mathcal{S}$ and $\mathcal{F}$ have been built, we now only build the mass matrix $\mathcal{M}$ and evaluate the integrals underlying $\mathcal{S}$ and $\mathcal{F}$ as we compute the right hand side of the equation. This form of evaluation is called matrix-free evaluation.

Computation of element mass matrix

For the mass matrix, we need to compute the following integral:

$$\mathcal{M}_{ij}^k = \int_{\mathsf{D}^k} \ell_i^k(\boldsymbol{x}) \ell_j^k(\boldsymbol{x})\, d\boldsymbol{x}.$$

As in finite elements, the integral is transformed to the reference element $[-1,1]^2$ with coordinates r, s. We denote the Jacobian matrix of the transformation from the reference to the real element by $\boldsymbol{J}^k$. The 2×2 matrix $\boldsymbol{J}^k$ is given by $\boldsymbol{J}^k = \frac{d\boldsymbol{F}^k}{d\boldsymbol{r}} = \begin{bmatrix} \frac{\partial F_1^k}{\partial r} & \frac{\partial F_1^k}{\partial s} \\ \frac{\partial F_2^k}{\partial r} & \frac{\partial F_2^k}{\partial s} \end{bmatrix}$, where $\boldsymbol{F}^k = (F_1^k, F_2^k)^T$ is a bi-linear (or higher order) transformation from the unit cell nodes to the nodes in real coordinate. We denote the Lagrange polynomials on the unit element by $\ell_i(r,s)$.
The integral for the mass matrix is thus given by the following formula:

$$\mathcal{M}_{ij}^k = \int_{-1}^{1}\int_{-1}^{1} \ell_i(r,s)\ell_j(r,s)|\boldsymbol{J}^k(r,s)|\, dr ds$$

The integral is approximated by quadrature. Here, we want to consider the case of Gaussian quadrature based on the Gauss–Legendre quadrature nodes ξ_a and Gauss weights w_a. From a one-dimensional Gauss rule, a multi-dimensional rule is constructed by a tensor product, i.e., we select

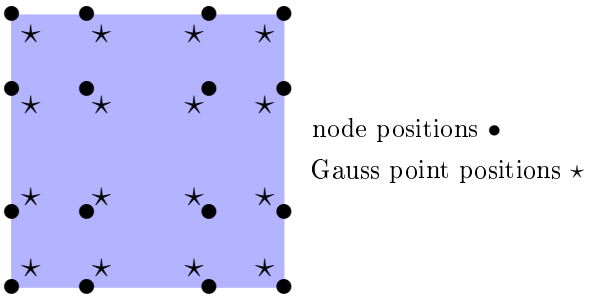
- the Gauss nodes $\boldsymbol{r}_q = (r_{q_1}, s_{q_2}) = (\xi_{q_1}, \xi_{q_2})$
- with quadrature weight $w_q = w_{q_1} w_{q_2}$.

This gives the integral approximation

$$\mathcal{M}_{ij}^k \approx \sum_{q=1}^{N_c} \ell_i(\boldsymbol{r}_q)\ell_j(\boldsymbol{r}_q)|\boldsymbol{J}^k(\boldsymbol{r}_q)|w_q = \sum_{q_1=1}^{N_{c,1\mathrm{D}}}\sum_{q_2=1}^{N_{c,1\mathrm{D}}} \underbrace{\ell_{i_1}(r_{q_1})\ell_{i_2}(s_{q_2})}_{\ell_i(\boldsymbol{r}_q)} \underbrace{\ell_{j_1}(r_{q_1})\ell_{j_2}(r_{q_2})}_{\ell_j(\boldsymbol{r}_q)} |\boldsymbol{J}^k(r_{q_1}, s_{q_2})| w_{q_1} w_{q_2}.$$

We usually choose the number of Gauss points per dimension $N_{c,1\mathrm{D}}$ to equal the number of basis functions per direction, i.e., equal to $N+1$. Since Gaussian quadrature on $N+1$ evaluates polynomials up to degree $2N+1$ exactly and the highest polynomial degree is $2N$, this ensures exact quadrature. However, for non-affine geometries, i.e., geometries where the transformation between unit and real cell

is not linear, the determinant of the Jacobian matrix $|\boldsymbol{J}^k(r_{q_1}, s_{q_2})|$ is also a polynomial. If a bi-linear mapping from reference to real cell is used, its degree is at most 2 (because $\boldsymbol{J}^k$ is at most linear in r and s), and the degree is at most 3 in 3D. However, if a polynomial mapping of degree l is used, it can contain polynomials up to degree $2l$ (or $3l$ in 3D). This increases the number of quadrature points necessary for exact integration. However, one often does not use more than $N+1$ points per dimension even in this case because the quadrature error appears to be of order $2N+2$ as compared to $N+1$ for the polynomial interpolation. (The actual situation is more involved and under-integration of geometry can give rise to similar "aliasing" problems as inexactly captured nonlinear terms. Often, aliasing errors from nonlinear terms are the more important source of error.)
For $N_{c,1\mathrm{D}} = N+1 = 4$, the evaluation of the integrals involves the following nodes:



For computing the integrals, we need to evaluate all the Lagrangian polynomials in the quadrature points marked by $\star$. Let us collect the evaluation of the basis functions in the quadrature points in a matrix $\mathcal{N} \in \mathbb{R}^{N_p \times N_c}$:

$$\mathcal{N}_{iq} = \ell_i(\boldsymbol{r}_q).$$

Illustration for bilinear functions $N = 1$ in 2D:

$$\mathcal{N} = \begin{bmatrix} 0.62 & 0.17 & 0.17 & 0.045 \\ 0.17 & 0.62 & 0.045 & 0.17 \\ 0.17 & 0.045 & 0.62 & 0.17 \\ 0.045 & 0.17 & 0.17 & 0.62 \end{bmatrix}$$

With this matrix, the mass matrix is given by the following product of matrices:

$$\mathcal{M}^k = \mathcal{N}\mathcal{W}^k\mathcal{N}^T,$$

where the matrix

$$\mathcal{W}^k = \begin{bmatrix} |\boldsymbol{J}^k(\boldsymbol{r}_1)|w_1 & 0 & \dots \\ 0 & |\boldsymbol{J}^k(\boldsymbol{r}_2)|w_2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

collects the quadrature weights and Jacobian determinants on all quadrature points on the diagonal. Due to the DG construction, the DG mass matrices can be inverted element by element and applied to the right hand side terms.

Evaluation of advection term

For the advection term, we do not want to compute the matrix explicitly. Rather, we want to directly evaluate the integral for a given function $u_h^k(\boldsymbol{x}, t)$. Going through all test functions, we obtain a vector $\boldsymbol{s} = (s_1, s_2, \dots)^T$ with the following entries:

$$s_j = \int_{\mathsf{D}^k} \nabla \ell_j^k(\boldsymbol{x}) \cdot \boldsymbol{f}(u_h^k)\, d\boldsymbol{x}$$

Again, we transform the integral to the reference element. The gradient of ℓ_j^k is given by the product of the transpose of the inverse Jacobian matrix, $\left(\boldsymbol{J}^k\right)^{-T}$, and the gradient on the reference element. This gives the following integral:

$$s_j = \int_{-1}^{1}\int_{-1}^{1}\left(\left(\boldsymbol{J}^k\right)^{-T}(\boldsymbol{r})\nabla_{\boldsymbol{r}}\ell_j(\boldsymbol{r})\right)^T \boldsymbol{f}(u_h^k(\boldsymbol{r}))|\boldsymbol{J}^k|\,drds.$$

As before, we approximate the integral by quadrature. This results in the formula

$$s_j \approx \sum_{q=1}^{N_c}\left(\left(\boldsymbol{J}^k\right)^{-T}(\boldsymbol{r}_q)\nabla_{\boldsymbol{r}}\ell_j(\boldsymbol{r}_q)\right)^T \boldsymbol{f}(u_h^k(\boldsymbol{r}_q))|\boldsymbol{J}^k|w_q.$$

For the evaluation of $\boldsymbol{f}(u_h^k)$, we first evaluate u_h^k in all quadrature points $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots$. In vector notation, this is given by the operation:

$$\boldsymbol{u}_q^k = \begin{bmatrix} u_h^k(\boldsymbol{r}_1) \\ u_h^k(\boldsymbol{r}_2) \\ u_h^k(\boldsymbol{r}_3) \\ \vdots \end{bmatrix} = \mathcal{N}^T\boldsymbol{u}^k,$$

where $\boldsymbol{u}^k$ denotes the values of the DG-FEM solution u_h^k in the element nodes. In the quadrature points, we can then evaluate the flux $\boldsymbol{f}$ pointwise. Since the flux $\boldsymbol{f} = (f_1, f_2)$ maps a scalar value u onto a vector of two components, we will denote by $\boldsymbol{f}^k$ the vector of length $2N_c$ containing both components of $\boldsymbol{f}$, with the first N_c entries belonging to the first component f_1 and the second N_c entries to the second component f_2.
Next, we represent the gradient operation in terms of a matrix. To this end, we denote by $\mathcal{D}$ the derivative of the basis functions ℓ_j on the reference element with respect to r and s:

$$\mathcal{D} = \begin{bmatrix}\mathcal{D}_r & \mathcal{D}_s\end{bmatrix} \in \mathbb{R}^{N_p\times 2N_c}$$

with the matrices of partial derivatives defined by

$$\mathcal{D}_{r,iq} = \frac{\partial \ell_i(r_q, s_q)}{\partial r}$$

and

$$\mathcal{D}_{s,iq} = \frac{\partial \ell_i(r_q, s_q)}{\partial s}$$

Taking all the above steps together, we obtain the following formula for computing the evaluation of the elemental advection vector:

$$\boldsymbol{s}^k = \mathcal{D}\mathcal{X}^k\boldsymbol{f}^k,$$

where the matrix $\mathcal{X}^k$ contains the product of the inverse Jacobian matrix of the transformation $(\boldsymbol{J}^k)^{-1}$, the determinant of the Jacobian matrix $|\boldsymbol{J}^k(\boldsymbol{r}_q)|$, and the quadrature weight. Due to the particular way we order the derivative matrix $\mathcal{D}$ and the flux $\boldsymbol{f}^k$, the structure of $\mathcal{X}^k$ is a block matrix of diagonal matrices,

$$\mathcal{X}^k = \begin{bmatrix}\mathcal{X}_{11}^k & \mathcal{X}_{12}^k \\ \mathcal{X}_{21}^k & \mathcal{X}_{22}^k\end{bmatrix} = \begin{bmatrix}\operatorname{diag}\left(\left(J^k\right)_{11}^{-1} w_q|\boldsymbol{J}^k|\right)_{q=1:N_c} & \operatorname{diag}\left(\left(J^k\right)_{12}^{-1} w_q|\boldsymbol{J}^k|\right)_{q=1:N_c} \\ \operatorname{diag}\left(\left(J^k\right)_{21}^{-1} w_q|\boldsymbol{J}^k|\right)_{q=1:N_c} & \operatorname{diag}\left(\left(J^k\right)_{22}^{-1} w_q|\boldsymbol{J}^k|\right)_{q=1:N_c}\end{bmatrix},$$

where each matrix X_{ab}^k is of size $N_c \times N_c$ and diagonal with the respective quadrature point data as entry.

Evaluation of face integrals

For the face computations, we need to evaluate the integrals over products of ℓ_j^k and the DG-FEM solution u_h^k from both sides. We only discuss the weak form here; the strong form is derived similarly. We proceed in analogy to the cell term by transforming the integral to the reference element and extracting the values on the faces. Since face terms involve in general data from both sides of the face, we do not proceed element by element but rather pass through all the faces of the computational mesh. For interior faces, we proceed according to the following steps:

Loop over all interior faces e:

1. Identify the indices k^- and k^+ of the two elements adjacent to face e
2. Compute the Jacobian transformation $\boldsymbol{J}^{k^-}$ and $\boldsymbol{J}^{k^+}$ on the face quadrature points (in 2D: the line over the face) for the left and right element. The local face numbers are denoted by $\hat{e}^-$ and $\hat{e}^+$ (both being a number between 1 and 4).
3. Evaluate $u_h^{k^-}$ and $u_h^{k^+}$ on the face quadrature points by multiplication with appropriate matrices $\mathcal{N}_{e^-}^T$ and $\mathcal{N}_{e^+}^T$ that contain the values of the N_p shape functions evaluated in the quadrature points of the respective faces.
4. On each quadrature point:
   a) Compute the normal vector $\hat{\boldsymbol{n}}^-$ on element k^- by the following procedure:
      - Transform unit tangential vector $\boldsymbol{t}(\hat{e}^-)$ to real space:
      $$\boldsymbol{t}^- = \begin{bmatrix} t_x^- \\ t_y^- \end{bmatrix} = \boldsymbol{J}^{k^-} \boldsymbol{t}(\hat{e}^-)$$
      - Compute direction of normal vector by the vector orthogonal to $\boldsymbol{t}^-$,
      $$\tilde{\boldsymbol{n}}^- = \begin{bmatrix} -t_y^- \\ t_x^- \end{bmatrix}$$
      - Normalize:
      $$\hat{\boldsymbol{n}}^- = \frac{\tilde{\boldsymbol{n}}^-}{|\tilde{\boldsymbol{n}}^-|}$$

      In 3D, a similar procedure is applied but one transforms the two tangential vectors from the unit surface and then takes the cross product for finding $\tilde{\boldsymbol{n}}^- = \boldsymbol{t}^- \times \boldsymbol{t}_2^-$.
   b) Evaluate the numerical flux $\boldsymbol{f}^*$ from u^- and u^+ on the quadrature point $\boldsymbol{r}_q$.
   c) Multiply $\boldsymbol{f}^*$ by the normal vector $\hat{\boldsymbol{n}}^-$. Write this result into a vector a^- for the element k^-. The vector on the outer side, a^+ for the element k^+, is given by $-a^-$ (negative sign because the normal vector points into the opposite direction for k^+).
   d) Multiply by the quadrature weight and the determinant of the Jacobian matrix for the face, i.e., the norm of the transformed tangential vector $|\boldsymbol{t}^-|$.
5. Multiply the resulting vectors $a^\pm$ by $\mathcal{N}_{e^-}$ and $\mathcal{N}_{e^+}$ and subtract the results from the value for the right hand side in index k^- and k^+, respectively. The negative sign is because the face term has negative sign in eq. (4.2). The multiplication by the matrices $\mathcal{N}_{e^-}$ and $\mathcal{N}_{e^+}$, respectively, performs the summation over all quadrature points and tests by all test functions on the elements k^- and k^+.

A similar procedure is applied for the boundary faces:

Loop over all boundary faces e:

1. Identify the index k^- elements adjacent to face e
2. Compute the Jacobian transformation $\boldsymbol{J}^{k^-}$ on the face quadrature points (in 2D: the line over the face). The local face number is denoted by $\hat{e}^-$ (a number between 1 and 4 in 2D; in 3D, there are 6 faces).
3. Evaluate $u_h^{k^-}$ on the face quadrature points by multiplication with the matrix $\mathcal{N}_{e^-}^T$.
4. On each quadrature point:
   a) Compute the normal vector $\hat{\boldsymbol{n}}^-$ on element k^- by the following procedure:
      - Transform unit tangential vector $t(\hat{e}^-)$ to real space:
      $$\boldsymbol{t}^- = \begin{bmatrix} t_x^- \\ t_y^- \end{bmatrix} = \boldsymbol{J}^{k^-} \boldsymbol{t}(\hat{e}^-)$$
      - Compute direction of normal vector by the vector orthogonal to $\boldsymbol{t}^-$,
      $$\tilde{\boldsymbol{n}}^- = \begin{bmatrix} -t_y^- \\ t_x^- \end{bmatrix}$$
      - Normalize:
      $$\hat{\boldsymbol{n}}^- = \frac{\tilde{\boldsymbol{n}}^-}{|\tilde{\boldsymbol{n}}^-|}$$
   b) Evaluate the numerical flux $\boldsymbol{f}^*$ from u^- and possible boundary values g.
   c) Multiply $\boldsymbol{f}^*$ by the normal vector $\hat{\boldsymbol{n}}^-$. Write this result into the vector a^- for the element k^-.
   d) Multiply by the quadrature weight and the determinant of the Jacobian matrix for the face $|\boldsymbol{t}^-|$.
5. Multiply the resulting vectors by $\mathcal{N}_{e^-}$ and subtract the results from the value for the right hand side in index k^-. The negative sign is because the face term has negative sign in eq. (4.2).

4.2.1 Fast evaluation of integrals

Both the evaluation of the advection term and the face terms involve essentially a sequence of three operations:

- Evaluation of shape functions or their derivatives in all quadrature points (reference cell derivatives)
- Operations on quadrature points, including flux evaluation, application of geometry, etc.
- Multiplication for all test functions (on reference cell) and summation over quadrature points

When considering the numerical cost, we realize that the first and third operation both have leading order cost $N_p N_c$, which for the case $N_p = N_c$ simply evaluates to N_p^2. The operations on quadrature points are much cheaper in comparison with costs $\mathcal{O}(N_c)$ (but the constant is usually a bit larger). To see why this is a problem, consider the work for evaluating the shape functions on all quadrature points in 2D and 3D:

	2D		3D	
N	N_p	N_p^2	N_p	N_p^2
1	4	16	8	64
2	9	81	27	729
3	16	256	64	4096
4	25	625	125	15625
5	36	1296	216	46656

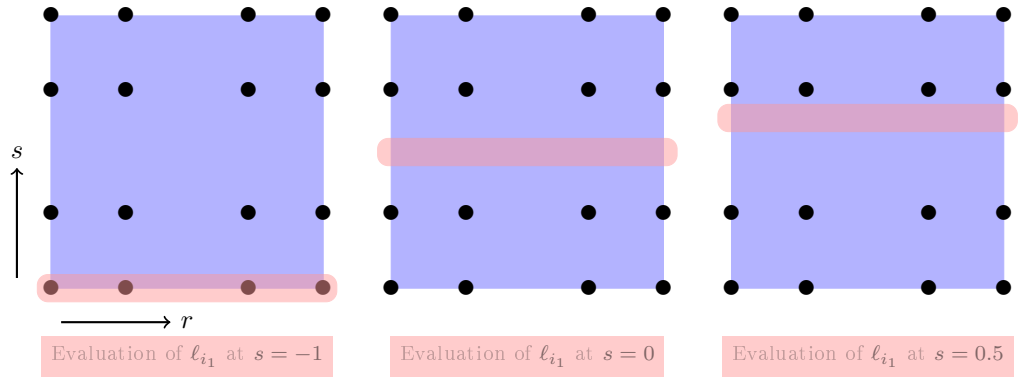
In order to make the method competitive, we need to improve on the evaluation of the shape functions on quadrature points, in particular for 3D.
The idea to make this operation faster is so-called tensorial evaluation. The idea is to exploit that both the basis functions $\ell_i(\boldsymbol{r}_q) = \ell_{i_1}(r_{q_1})\ell_{i_2}(s_{q_2})$ form a tensor product and we want to evaluate the shape functions on a tensor product quadrature.
Let us consider the fast evaluation of basis functions in the quadrature points, i.e., the operation

$$\mathcal{N}^T \boldsymbol{u}^k.$$

Matrix $\mathcal{N}$ is constructed as follows:

$$\mathcal{N}_{iq} = \ell_i^{\mathrm{2D}}(\boldsymbol{r}_q) = \ell_{i_1}^{\mathrm{1D}}(r_{q_1})\ell_{i_2}^{\mathrm{1D}}(s_{q_2}).$$

By construction of the basis functions and the quadrature formula, the evaluation along the r-direction is the same in all layers of the y-direction: No matter what the coordinate s_{q_2} is, the first part $\ell_i^{\mathrm{1D}}(r_{q_1})$ is always the same.



In matrix notation, this construction manifests itself in form of a *tensor product matrix*,

$$\mathcal{N} = \mathcal{N}_2^{\mathrm{1D}} \otimes \mathcal{N}_1^{\mathrm{1D}},$$

where the matrices $\mathcal{N}_1^{\mathrm{1D}}$ and $\mathcal{N}_2^{\mathrm{1D}}$ collect the evaluation of the 1D basis functions in r and s directions, respectively:

$$\mathcal{N}_1^{\mathrm{1D}} = \ell_{i_1}^{\mathrm{1D}}(r_{q_1}),$$
$$\mathcal{N}_2^{\mathrm{1D}} = \ell_{i_2}^{\mathrm{1D}}(s_{q_2}).$$

The symbol $\otimes$ denotes the tensor product of matrices, also called Kronecker product. Since we have the same basis functions in r and s direction and the same quadrature points, the two matrices are the same,

$$\mathcal{N}_1^{\mathrm{1D}} = \mathcal{N}_2^{\mathrm{1D}}.$$

In the above example for $\mathcal{Q}_3$ basis functions (cubic polynomials on a tensor grid), the 1D matrices are 4×4. The tensor product $\mathcal{N}^{1D} \otimes \mathcal{N}^{1D}$ is a 16×16 matrix. In a tensor product, the dimension of the final matrix is the product of the dimensions in the two involved matrices.
In three space dimensions, the shape matrix $\mathcal{N}$ is constructed by a tensor product of three matrices,

$$\mathcal{N}^{3D} = \mathcal{N}^{1D} \otimes \mathcal{N}^{1D} \otimes \mathcal{N}^{1D},$$

and the final dimension for polynomial degree N is $(N+1)^3 \times (N+1)^3$ (e.g. 64×64 for $\mathcal{Q}_3$ elements).

Matrix-vector product with Kronecker matrices.

Let us consider the Kronecker product of two matrices $\mathcal{A} \in \mathbb{R}^{m \times n}$ and $\mathcal{B} \in \mathbb{R}^{p \times q}$ in a more abstract way. We associate matrix $\mathcal{A}$ with values in the first index direction i_1 (r direction) and matrix $\mathcal{B}$ with shape values in the second index direction i_2 (s direction).
The Kronecker product is written as

$$\mathcal{B} \otimes \mathcal{A} = \begin{bmatrix} b_{11}\mathcal{A} & \cdots & b_{1q}\mathcal{A} \\ \vdots & \ddots & \vdots \\ b_{p1}\mathcal{A} & \cdots & b_{pq}\mathcal{A} \end{bmatrix} \tag{4.3}$$

or, more explicitly,

$$\mathcal{B} \otimes \mathcal{A} = \begin{bmatrix} b_{11}a_{11} & b_{11}a_{12} & \cdots & b_{11}a_{1n} & \cdots & \cdots & b_{1q}a_{11} & b_{1q}a_{12} & \cdots & b_{1q}a_{1n} \\ b_{11}a_{21} & b_{11}a_{22} & \cdots & b_{11}a_{2n} & \cdots & \cdots & b_{1q}a_{21} & b_{1q}a_{22} & \cdots & b_{1q}a_{2n} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ b_{11}a_{m1} & b_{11}a_{m2} & \cdots & b_{11}a_{mn} & \cdots & \cdots & b_{1q}a_{m1} & b_{1q}a_{m2} & \cdots & b_{1q}a_{mn} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ b_{p1}a_{11} & b_{p1}a_{12} & \cdots & b_{p1}a_{1n} & \cdots & \cdots & b_{pq}a_{11} & b_{pq}a_{12} & \cdots & b_{pq}a_{1n} \\ b_{p1}a_{21} & b_{p1}a_{22} & \cdots & b_{p1}a_{2n} & \cdots & \cdots & b_{pq}a_{21} & b_{pq}a_{22} & \cdots & b_{pq}a_{2n} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ b_{p1}a_{m1} & b_{p1}a_{m2} & \cdots & b_{p1}a_{mn} & \cdots & \cdots & b_{pq}a_{m1} & b_{pq}a_{m2} & \cdots & b_{pq}a_{mn} \end{bmatrix}$$

Now we compute the product $\boldsymbol{z} \in \mathbb{R}^{mp}$ of $\mathcal{B} \otimes A$ with a vector $\boldsymbol{y} \in \mathbb{R}^{nq}$.

$$\boldsymbol{z} = (\mathcal{B} \otimes \mathcal{A})\boldsymbol{y},$$

Assume that $\boldsymbol{y}$ is sorted with the indices i_1 associated to matrix $\mathcal{A}$ running fastest, i.e.,

$$\boldsymbol{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_n & \cdots & \cdots & y_{(q-1)n+1} & y_{(q-1)n+2} & \cdots & y_{qn} \end{bmatrix}^T.$$

By looking at the matrix in (4.3), we observe a repeating structure. In the first column of the block matrix, the matrix $\mathcal{A}$ is repeated p times for each of the coefficients b_{11} until b_{p1} of the matrix $\mathcal{B}$. The goal of the following is to transform the multiplication by $\mathcal{B} \otimes \mathcal{A}$ into a series of multiplications by $\mathcal{A}$ and $\mathcal{B}$.

Multiplication by $\mathcal{A}$. To use the repeated appearance of $\mathcal{A}$, we factor out the common coefficient b_{11} from the first matrix block and form a product with $\mathcal{A}$ on the entries $\boldsymbol{y}^{(1)} = [y_1, y_2, \ldots, y_n]^T$.

We write

$$\boldsymbol{w}^{(1)} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \mathcal{A}\boldsymbol{y}^{(1)}$$

Similarly, we can form a vector $\boldsymbol{y}^{(2)} = [y_{n+1}, y_{n+2}, \dots y_{2n}]^T$ and form the product

$$\boldsymbol{w}^{(2)} = \mathcal{A}\boldsymbol{y}^{(2)}.$$

This way, we proceed with until $\boldsymbol{w}^{(q)} = \mathcal{A}\boldsymbol{y}^{(q)}$.

By this algorithm, we have formed a vector

$$\boldsymbol{w} = \left[\left(\boldsymbol{w}^{(1)}\right)^T, \left(\boldsymbol{w}^{(2)}\right)^T, \dots, \left(\boldsymbol{w}^{(q)}\right)^T \right]^T \in \mathbb{R}^{mq},$$

which involved q multiplications by the matrix $\mathcal{A}$.

Multiplication by $\mathcal{B}$. In the second step, we need to perform similar operations using the matrix $\mathcal{B}$ that had been factored out. In terms of the matrix (4.3), we have multiplied and summed within the matrix $\mathcal{A}$. To perform multiplication by the entries of $\mathcal{B}$ and the final summation, consider the first entry of the result vector, z_1:

$$z_1 = \sum_{i=1}^{q} b_{1i} w_1^{(i)}.$$

Similarly, we obtain

$$z_2 = \sum_{i=1}^{q} b_{1i} w_2^{(i)},$$

and so on until

$$z_m = \sum_{i=1}^{q} b_{1i} w_m^{(i)}.$$

For the next entry z_{m+1}, the sum is again similar as for z_1 but using the next series of coefficients b_{2i}. Taken together, we obtain the vector $\boldsymbol{z}$ when going through the rows one by one.

This operation corresponds to m multiplications by the matrix $\mathcal{B}$ on the m vectors

$$\begin{aligned} &[w_1^{(1)}, w_1^{(2)}, \dots, w_1^{(q)}]^T, \\ &[w_2^{(1)}, w_2^{(2)}, \dots, w_2^{(q)}]^T, \\ &\vdots \\ &[w_m^{(1)}, w_m^{(2)}, \dots, w_m^{(q)}]^T. \end{aligned}$$

In the second step representing multiplication by $\mathcal{B}$, the summation does not run over direct neighbors in the temporary vector $\boldsymbol{w}$ but rather jumps over m entries. If we collect $\boldsymbol{w}$ in a matrix $\mathcal{W}$ by putting q columns consisting of m entries next to each other,

$$\mathcal{W} = \begin{bmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & w_1^{(q)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & w_2^{(q)} \\ \vdots & \vdots & \ddots & \vdots \\ w_m^{(1)} & w_m^{(2)} & \cdots & w_m^{(q)} \end{bmatrix}$$

then the second step is a matrix-matrix product

$$\mathcal{B}\mathcal{W}^T.$$

Likewise, the first step to compute $\boldsymbol{w}$ from $\boldsymbol{y}$ is a matrix-matrix product, with the matrix $\mathcal{Y}$ formed by filling the entries of $\boldsymbol{y}$ into q columns of length n:

$$\mathcal{W} = \mathcal{A}\mathcal{Y}.$$

Taken together, the computation of $\boldsymbol{z}$ from $\boldsymbol{y}$ is represented by the product of the three matrices

$$\mathcal{Z} = \mathcal{A}\mathcal{Y}\mathcal{B}^T, \tag{4.4}$$

where multiplication by $\mathcal{B}^T$ from the right is nothing else than computing the product $\mathcal{B}\mathcal{W}^T$. The columns of matrix $\mathcal{Z}$ are the respective entries $z_{(i-1)m+1}$ to z_{im} (column index i) of the result vector $\boldsymbol{z}$.

Benefits of the tensorized matrix-vector product.

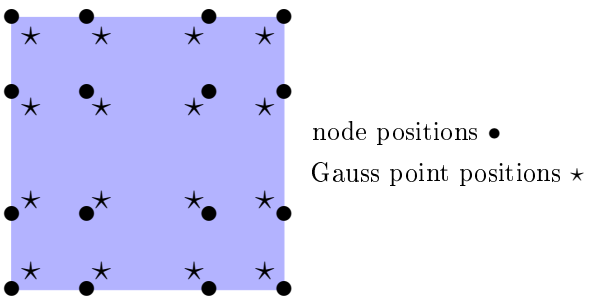
Let us look at the computational cost for the computation of $\mathcal{A}\mathcal{Y}\mathcal{B}^T$ instead of $(\mathcal{B}\otimes\mathcal{A})\boldsymbol{y}$. For simplicity, assume that the matrix dimensions are all the same, $m = n = p = q$. The original matrix-vector costs m^4 multiplications and m^4 additions, the cost of multiplying a matrix of size $m^2 \times m^2$ by a vector. The cost for the tensorized version is less:

- Multiplication $\mathcal{A}\mathcal{Y}$: Matrix-matrix product of matrices of size m, i.e., m^3 multiplications and m^3 additions
- Multiplication $\mathcal{W}\mathcal{B}^T$: m^3 multiplications and m^3 additions

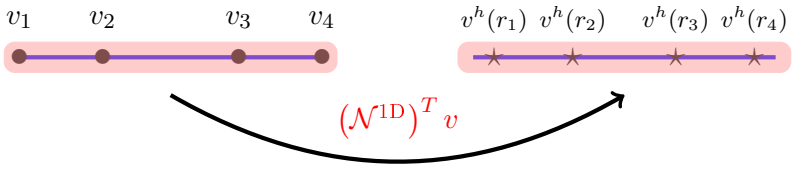
This algorithm reduces the cost from $2m^4$ arithmetic operations to $4m^3$ operations. If m is large (i.e., higher polynomial degree), this results in a considerable saving. This evaluation concept is also called *sum factorization* because it takes out common factors of the summation in the matrix-vector products.

Visualization.

Let us now find a graphical representation of the mechanisms of the tensorized product on the finite element nodes for the $\mathcal{Q}_3$ elements. As we have seen above, the goal of the product $\mathcal{N}^T\boldsymbol{u}^k$ is to evaluate the finite element function u^h in all quadrature points by a sum of the basis functions times the nodal values.



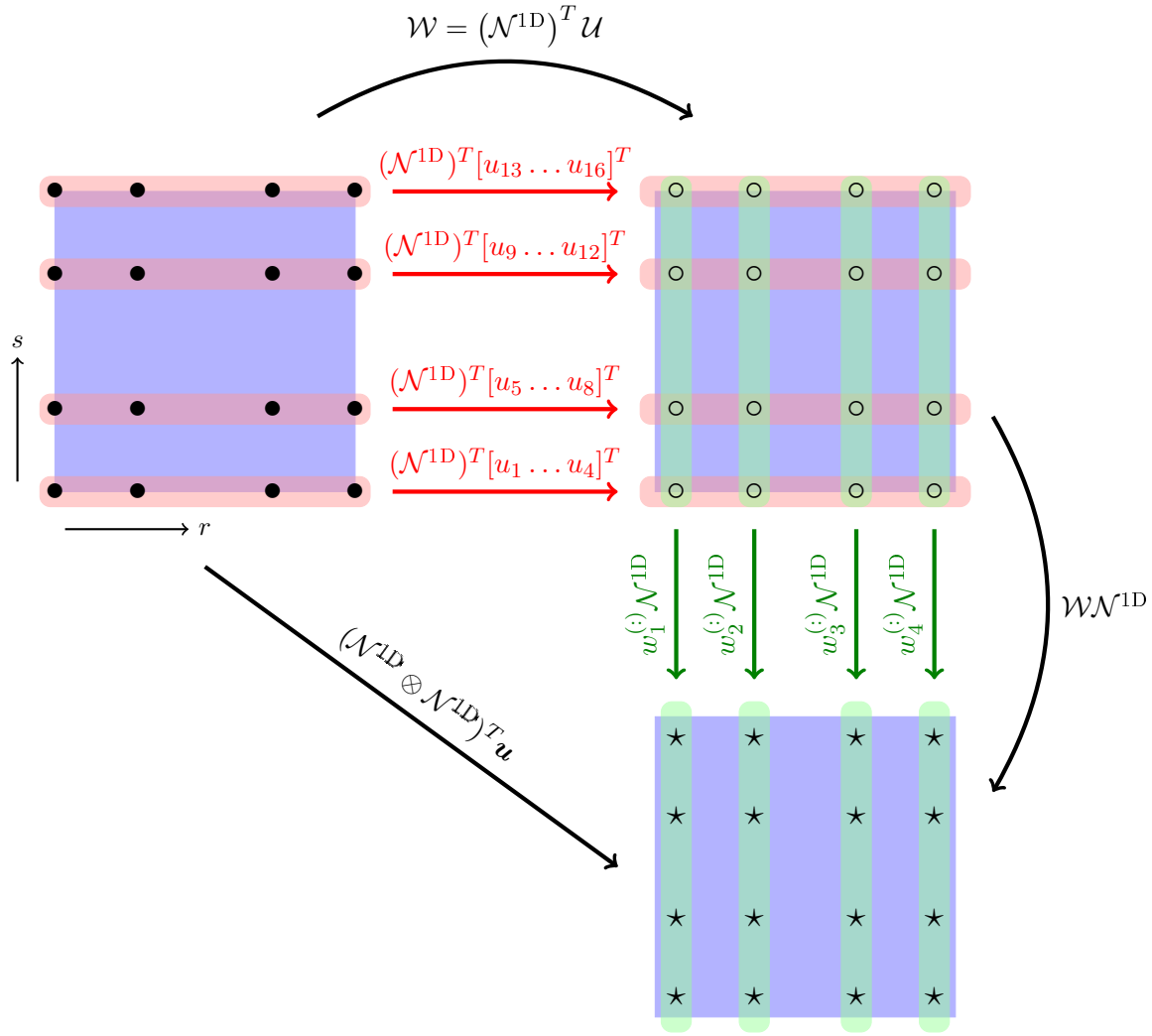
The first step is the multiplication $\mathcal{W} = \left(\mathcal{N}^{1\mathrm{D}}\right)^T\mathcal{U}$. The multiplication $\left(\mathcal{N}^{1\mathrm{D}}\right)^T\boldsymbol{v}$ represents the evaluation of a one-dimensional function with node values v on the 1D quadrature nodes. We depict this as follows:



For the two-dimensional case, we operate in two steps. The first step is to compute the temporary matrix $\mathcal{W}$ (i.e., the vectors $\boldsymbol{w}^{(i)}$). The order of summation in the product $\left(\mathcal{N}^{1\mathrm{D}}\right)^T \mathcal{U}$ is such that we only perform operations in the r-direction. The operations performed by the matrix-matrix product are visualized in red color in the plot below.
The second step in the tensorized matrix-vector product performs the multiplication along the second dimension,

$$\mathcal{W}\mathcal{N}^{1\mathrm{D}}$$

and is visualized in green color in the plot.



This graph illustrates that the algorithm changes the full coupling in the evaluation of the sum of all node values distributed onto the quadrature points into two one-dimensional operations. The one-dimensional operations only involve $N + 1$ operations at polynomial degree N and are thus cheaper than coupling between all $(N + 1)^2$ degrees of freedom and quadrature points.

Extension to three space dimensions.

The ideas shown in the graph above extend straight-forwardly to three spatial dimensions. The Kronecker product $\mathcal{N}^{1\mathrm{D}} \otimes \mathcal{N}^{1\mathrm{D}} \otimes \mathcal{N}^{1\mathrm{D}}$ is not explicitly formed to give a matrix of size $(N+1)^3 \times (N+1)^3$, but we rather apply a series of one-dimensional operations along the unit directions r, s, and t. A one-dimensional operation involves $(N+1)^2$ operations, and we have to do this for a whole plane of $(N+1)^2$ layers. For instance, the operation in r direction, has to be done for all node values (or quadrature points) in all values of s and t.
The savings in computational costs are more significant in 3D as compared to 2D: Executing the product $\left(\mathcal{N}^{1\mathrm{D}} \otimes \mathcal{N}^{1\mathrm{D}} \otimes \mathcal{N}^{1\mathrm{D}}\right) \boldsymbol{u}^k$ involves $2(N+1)^6$ operations for polynomial degree N, whereas the tensorized evaluation involves three computations with costs

$$\underbrace{(N+1)^2}_{\#\text{layers}} \quad \underbrace{2(N+1)^2}_{\text{operations in 1D}}$$

each. Thus, the total cost is

$$6(N+1)^4.$$

Using sum factorization for gradient operations and integration.

In the above demonstration, we considered the evaluation of the DG-FEM function u^h in the quadrature points, given the node values $\boldsymbol{u}^k$. Of course, this concept can also be applied for evaluating the first derivative with respect to the reference coordinates $\boldsymbol{r} = (r, s, t)$, i.e. $\nabla_{\boldsymbol{r}} u^h$. In matrix form, we represent this by the matrix

$$\mathcal{D}^T \boldsymbol{u}^k \quad \text{with} \quad \mathcal{D} = \begin{bmatrix} \mathcal{D}_r & \mathcal{D}_s \end{bmatrix}.$$

Each of the two matrices $\mathcal{D}_r$ and $\mathcal{D}_s$ is of similar shape as $\mathcal{N}$. If we take the partial derivatives with respect to r, this corresponds to derivatives with respect to the 1D basis function along r-direction but keeping the values of the 1D basis function along the s-direction, i.e.,

$$\begin{aligned} \mathcal{D}_r &= \mathcal{N}^{1\mathrm{D}} \otimes \mathcal{D}^{1\mathrm{D}}, \\ \mathcal{D}_s &= \mathcal{D}^{1\mathrm{D}} \otimes \mathcal{N}^{1\mathrm{D}}. \end{aligned}$$

Thus, we can again apply the idea of the tensor framework for the multiplication of $\boldsymbol{u}^k$ with $\mathcal{D}_r^T$ and $\mathcal{D}_s^T$, respectively.
Likewise, we can perform the integration step using sum factorization:

- Testing an equation by the test function values ℓ_i and performing summation over quadrature points corresponds to multiplication of a vector $\boldsymbol{v}$ in quadrature points with the matrix $\mathcal{N}$. This is realized by the same one-dimensional operations as above but with transposed matrices $\mathcal{N}^{1D}$ instead of $\left(\mathcal{N}^{1D}\right)^T$.
- An equation involving the gradient of test functions, $\nabla \ell_i$, is represented by the matrix-vector product $\mathcal{D}(\mathcal{X}^k \boldsymbol{f}^k)$, where $\mathcal{X}^k$ collects the factored-out Jacobian, Jacobian determinant, and quadrature weight, and $\boldsymbol{f}^k$ is the integrand (e.g. flux on quadrature points). This is treated by applying sum factorization separately on the r and s components of $\mathcal{X}^k \boldsymbol{f}^k$ and finally adding the two contributions.

For face integrals, the same techniques can be applied. For nodal basis functions, the evaluation of functions on the faces needs to first select the correct indices out of the nodal vector (e.g. the left or right end point in 1D). Secondly, the node values on the face need to be evaluated in the face quadrature

points. This operation is again of tensor form, with the dimension reduced by one as compared to cell integrals. Therefore, the cost for face integrals is $\mathcal{O}((N+1)^2)$ in 2D and $\mathcal{O}((N+1)^3)$ in 3D. If shape functions are not nodal (or if the node points are not on the element boundary), one needs to perform interpolation to the boundary by a weighted sum in the direction normal to the face using an evaluation point on the face. This is a 1D operation similar to the ones considered above.

Details of computational costs.

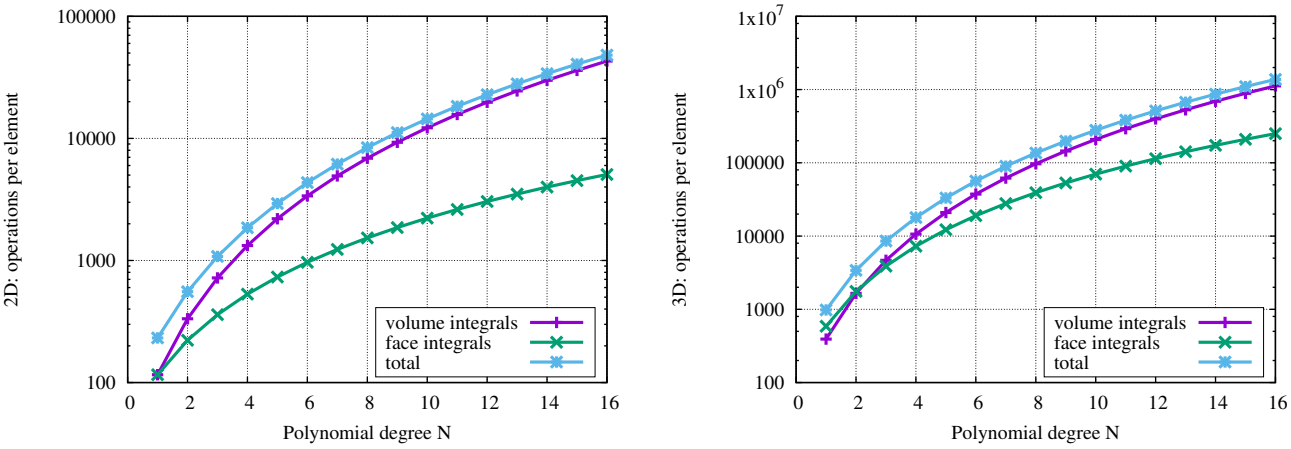
As we have seen above, the sum factorization step considerably reduces the computational effort to change from node values to values in quadrature points and the other way around. Let us re-consider the table from the beginning of this subsection and list the number of arithmetic operations for the operation $\mathcal{N}^T \boldsymbol{u}^k$ for the sum factorization evaluation as compared to the naive implementation:

	2D			3D		
N	N_p	naive	sum fact.	N_p	naive	sum fact.
1	4	32	32	8	128	96
2	9	162	108	27	1458	486
3	16	512	256	64	8192	1536
4	25	1250	500	125	31 250	3750
5	36	2592	864	216	93 312	7776

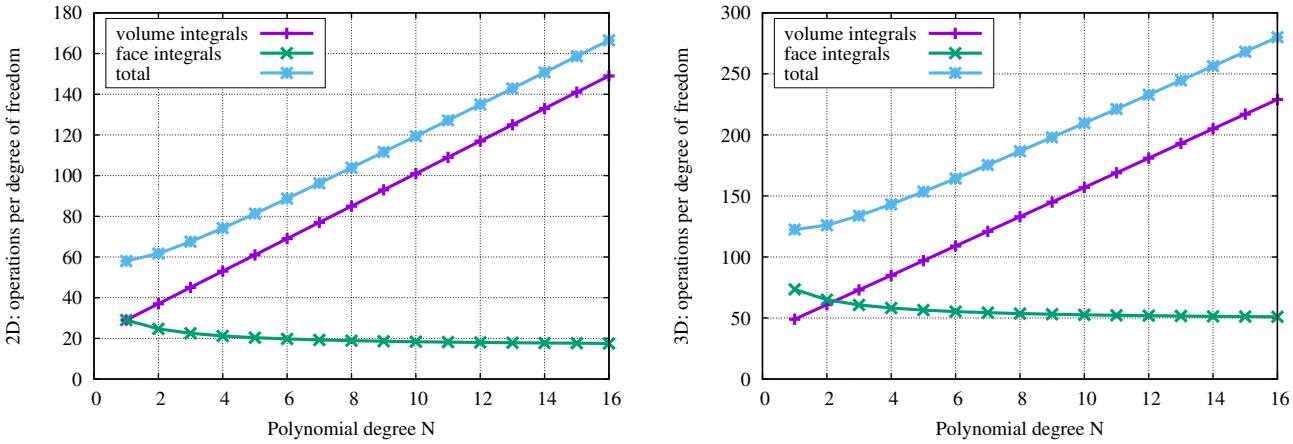
We see that the improvement in operations is about one order of magnitude for $N = 4$ in 3D.
Next, we analyze the cost per degree of freedom for the evaluation routines based on sum factorization for the advection equation with local integrals

$$\int_{\mathsf{D}^k} \left[\frac{\partial u_h^k}{\partial t} \ell_j^k(\boldsymbol{x}) - \boldsymbol{f}_h^k \cdot \nabla \ell_j^k(\boldsymbol{x}) \right] d\boldsymbol{x} = - \int_{\partial \mathsf{D}^k} \hat{\boldsymbol{n}} \cdot \boldsymbol{f}^* \ell_j^k(\boldsymbol{x}) d\boldsymbol{x}.$$

We display the work done per element for the 2D case (quadrilateral elements) on the left and the 3D case (hexahedral elements) on the right. We separately display the work for the volume integral and the face integrals. In order to make the work comparable, we assign the complete work on two faces (from the $-$ and the $+$ side) to one element and count 2 faces per element in 2D and 3 faces per element in 3D. This setup disregards the boundaries of the computational domain.

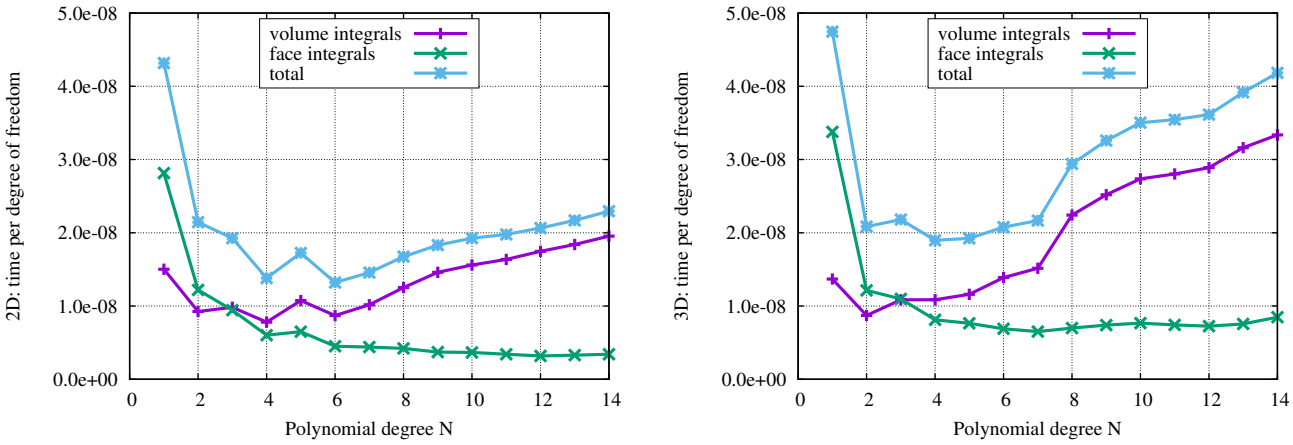


We observe that in 3D, the computation of the face integrals is more expensive than the volume integrals for polynomial degrees one and two. This effect can be more easily seen when listing the work per degree of freedom rather than per element:



Since the surface-to-volume ratio decreases for higher order elements, the work for face integration decreases as the polynomial degree increases. Thus, the operations for a quadratic element appear similar as for a linear one, computed relative to the number of degrees of freedom. For very large polynomial degrees, the asymptotic behavior with work proportional to the polynomial degree becomes apparent.

Finally, we want to show the computational time per degree of freedom on a high-performance implementation of the sum-factorization techniques (measured by using all six cores of an Intel Xeon E1650 (Sandy Bridge) CPU):



Conclusions

- The time per degree of freedom is almost constant over a wide range of polynomial degrees
- Can select as high polynomial degree as possible (mesh restrictions, variable coefficients, maybe stabilization character of face jumps)
- Face integration dominates at low degree, element integration at high degree

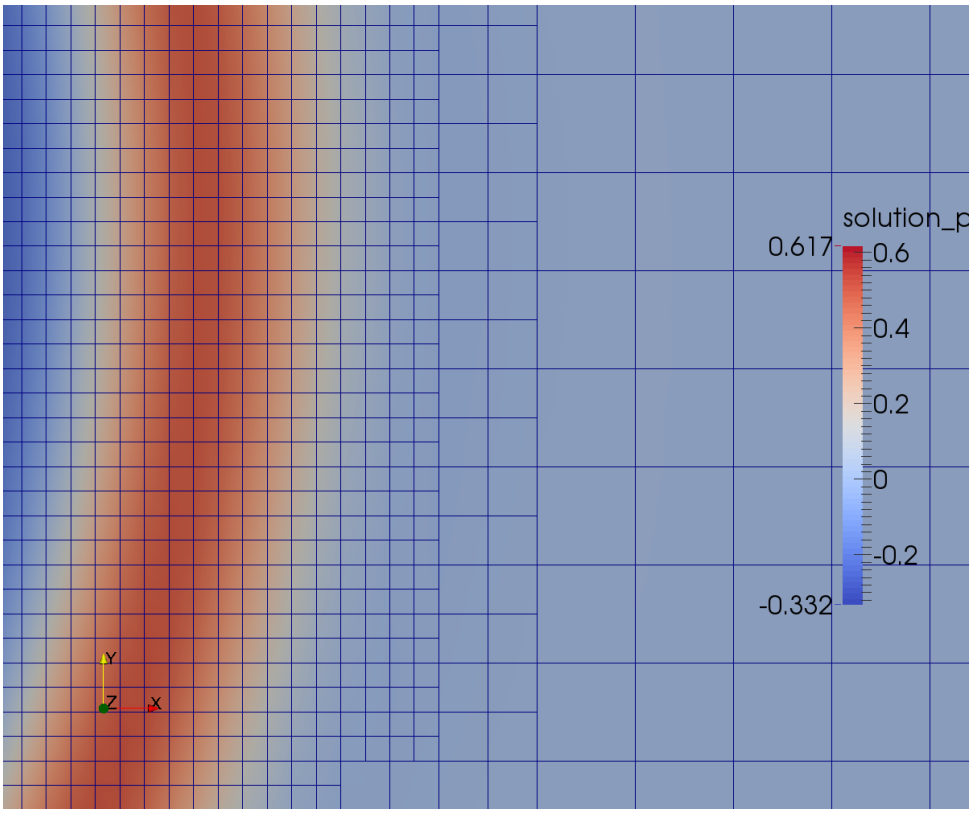
4.2.2 Non-conforming elements and adativity

A desirable feature of approximations to partial differential equations is the ability to concentrate the work, i.e., a fine mesh, to the most interesting regions. In mesh-based methods, this can be achieved by *adaptive mesh refinement*.

On quadrilateral elements, the most common strategy for mesh refinement is *structured refinement by subdivision of elements*. Structured refinement is typically organized by a mesh stored in a tree

concept. Each element represents a leaf of the tree. If an element is refined, four children (or eight in 3D) are created independently of the other elements. This way, some elements can be more refined than others. The number of refinement steps from the base mesh is called the level of the cell.
A suitable criterion for refinement can be the following:

- Elements with large gradients in the solution
- Elements with large jumps of the solution over faces
- A posteriori error estimation: Define a functional of the error (e.g. drag coefficient defined as an integral of the solution over some surface), solve an adjoint problem, weight residuals of equation (dual weighted residual techniques, DWR)



As before, we work with an algorithm that performs four blocks of operations:

- Volume (element) integrals – as usual
- Boundary integrals, including imposition of boundary conditions – as usual
- Integrals on faces between elements
  - Face integrals between elements of the same level of refinement (no hanging nodes) – as usual
  - Face integrals between elements of different refinement level – adjust!
- Apply inverse mass matrix – as usual

For the face integration on faces between different refinement level, we apply the following strategy on a mesh where we have a 2:1 mesh ratio, i.e., the difference between refinement levels is at most 1:

- Perform integration from the refined side, where u^- denotes the solution on the finer element and u^+ the solution on the coarser element
  - u^- is evaluated on the quadrature points of the face as usual with $\mathcal{N}_{e^-}^T$
  - For u^+, we need to evaluate on a "subface" relative to the coarse element, i.e., either $[-1,0]$ or $[0,1]$ in terms of unit coordinates:

    Case 1 $[-1,0]$: Evaluate on reference points $\tilde{r}_q = \frac{1}{2}r_q - \frac{1}{2}$ with matrix $(\mathcal{N}_{e^+}^{\mathrm{l}})^T$

    Case 2 $[0,1]$: Evaluate on reference points $\tilde{r}_q = \frac{1}{2}r_q + \frac{1}{2}$ with matrix $(\mathcal{N}_{e^+}^{\mathrm{r}})^T$
  - From both sides, the selection of the boundary points involved is the same as in the uniform case (sketch)
  - Go through quadrature points and evaluate numerical flux from u^- and u^+ in the quadrature points of the refined edge as usual
  - Multiply by $\mathcal{N}_{e^-}$ for testing on e^-, by $\mathcal{N}_{e^+}^{\mathrm{l/r}}$

In 3D, the algorithm proceeds similarly. There are four different cases to be treated. In terms of the tensorized evaluation, it suffices to combine the one-dimensional matrices $\mathcal{N}_{e^+}^{\mathrm{l/r}}$ along the r and s directions on the face.

DG-FEM is extensible to other non-conforming cases than the 2:1 mesh balance considered above, including non-matching grids where the vertices can be placed in arbitrary locations. This is easier than for continuous finite elements:

- In continuous finite elements, the 2:1 mesh balance allows to perform consistent computations relatively easily (need to "constrain" indices from the refined side such that only polynomials from the coarse side are possible – algebraic operation).
- For non-matching meshes, finding restrictions on the solution spaces must be done through additional variables with so-called mortar methods (idea: perform integration on one side with values from the other side, which introduces coupling matrices).

The only restriction for the basic variant of DG-FEM are *consistent interfaces*:
There must be a unique interface between the two meshes without overlapping regions or gaps in order to integrate on the same domain. Otherwise:

- Ignore gaps/overlaps in face integrals (inconsistent!)
- Use high order boundary mappings (approximate boundary by high order basis functions) to reduce gaps
- Evaluate shape functions of one element exactly on the surface of the neighbor by local interpolation/extrapolation to the real locations (expensive to find the unit coordinates)

4.3 DG-FEM on triangles

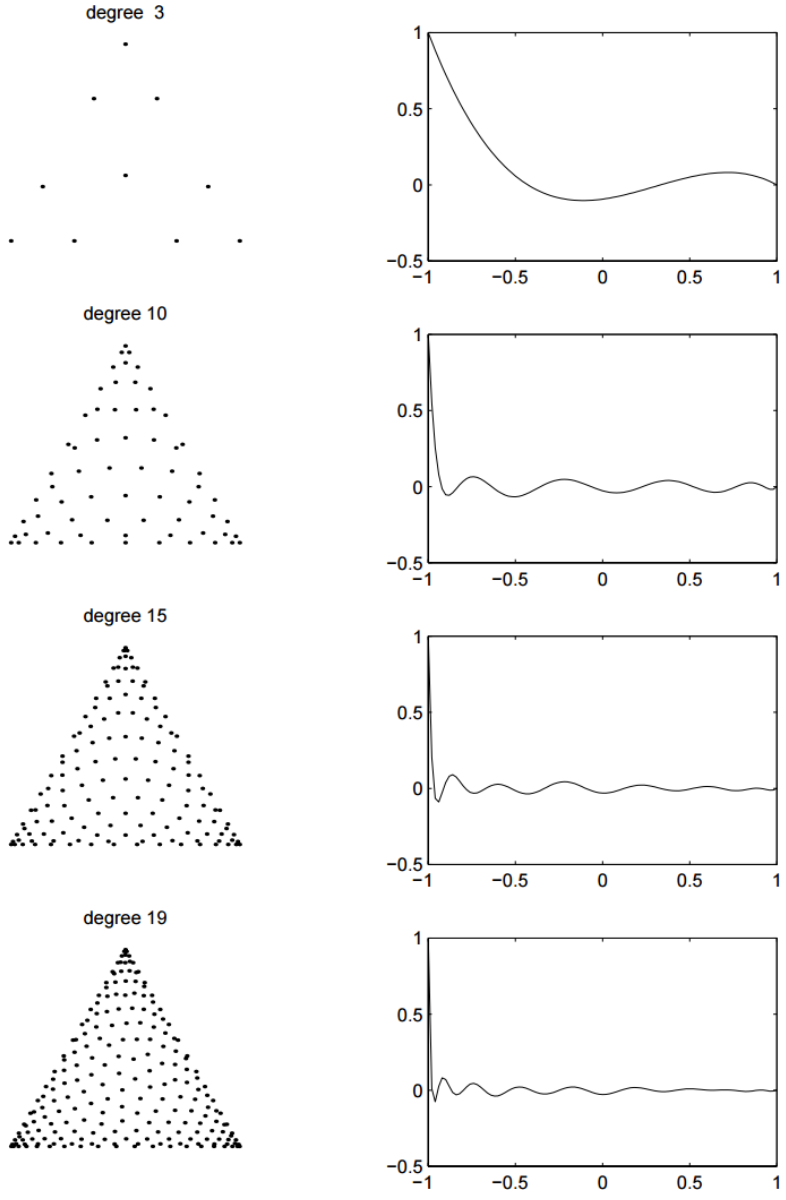
The procedure for DG-FEM on triangles and tetrahedra is in many aspects similar to quadrilaterals and hexahedra. In this section, we discuss the three main aspects that are different in the triangular case.

4.3.1 Definition of reference element nodes

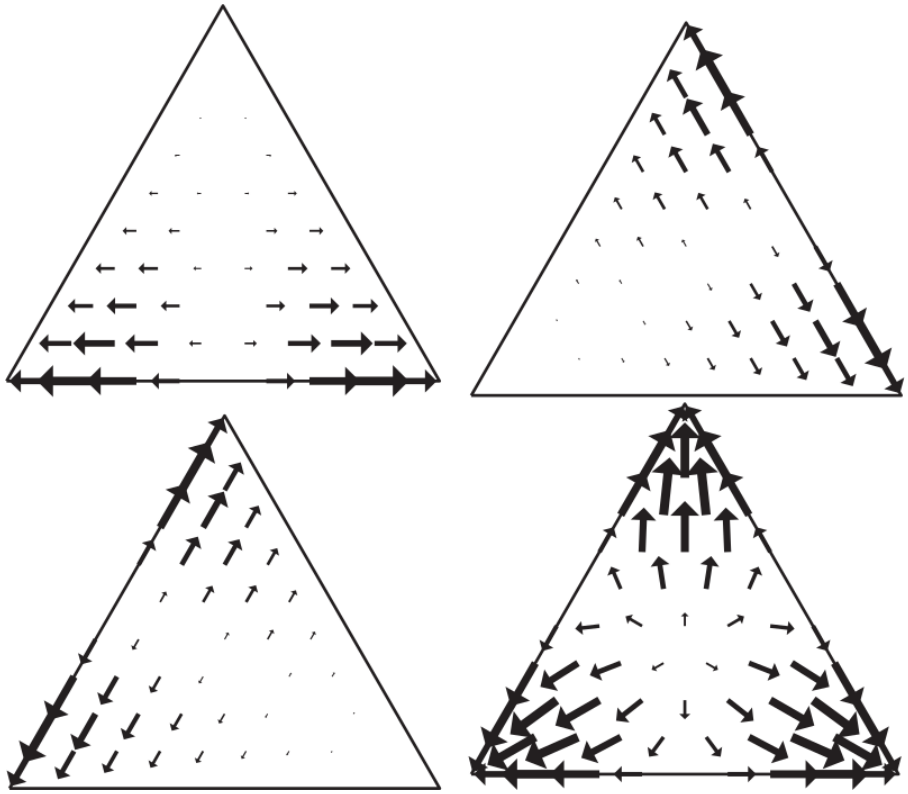
As opposed to the quadrilateral case where we can simply take the tensor product of the Gauss–Lobatto points in 1D, we have to work harder to derive suitable node distributions on triangles.
There are two ways to derive these points:

- Expensive algorithms which compute the node positions in 2D/3D ahead of time and tabulate them (similar to quadrature formulas)
- On-the-fly computation of the nodes from optimized 1D distribution (e.g. Gauss–Lobatto) by moving equidistant points into more favorable positions (blend-and-warp)

An example for the former method are so-called Fekete points. These points are a generalization of the Gauss–Lobatto points to triangles based on maximization of the determinant of the Vandermonde matrix of some orthogonal polynomials.
The resulting distribution on a equilateral triangle, together with the basis function located in the lower left corner, is as follows:



The latter procedure is described in detail in the course book, Section 6.1 [Hesthaven & Warburton, 2008]. The main idea is to define the shift function of equidistant points on the boundary of the triangles to the Gauss–Lobatto points and *warp* it into the elements. In order to combine the results from the three edges into the triangle, the three distributions are *blended*.
The shift of the equidistant points to the final end points is shown in the following figure (from [Hesthaven & Warburton, 2008]), first for the three edges separately and then the combined shifts:



The resulting node distributions are very close to the Fekete points for lower polynomial degrees. For higher degrees, the points are somewhat worse in terms of the determinant of the Vandermonde matrix but there are some variants of warp-and-blend that include a factor for improved conditioning [Hesthaven & Warburton, 2008].

How many node points are there on the unit element?

- 2D: $(N+1)(N+2)/2$ points (e.g. 15 for $N=4$)
- 3D: $(N+1)(N+2)(N+3)/6$ points (e.g. 35 for $N=4$)

4.3.2 Evaluation of Lagrange basis functions in terms of reference points

Having defined the node positions, we need to find a strategy to evaluate Lagrange polynomials based on these points. As opposed to the 1D (or tensor product) case where we can write down an analytic expression in terms of the Lagrange interpolation, we need another strategy on triangles.
The most common approach for evaluating in a quadrature node $\boldsymbol{r}_q$ is to perform the following three steps

- Definition of orthogonal polynomials ψ_i on the unit triangle: Use Jacobi polynomials, a generalization of Legendre polynomials

- Evaluation of the Vandermonde matrix $\mathcal{V}$ in the given nodes $\boldsymbol{r}_j$:

$$\mathcal{V}_{ij} = \psi_i(\boldsymbol{r}_j)$$

- Evaluation of Lagrange polynomial ℓ_j in quadrature node $\boldsymbol{r}_q$:
  - Evaluate ψ_i in $\boldsymbol{r}_q \to \boldsymbol{p}$
  - Compute $\boldsymbol{l} = \mathcal{V}^{-T}\boldsymbol{p}$, where $\boldsymbol{l}$ holds the values of the Lagrange polynomials in point $\boldsymbol{r}_q$.

The integrals over the elements and faces can then be computed similar to the quadrilateral case by transformation to the reference element, again involving the Jacobian matrix $\boldsymbol{J}$ of the transformation between reference and real element in front of derivatives.

4.3.3 Elementwise operations

In principle, the quadrature technique introduced for quadrilaterals can be used on triangles as well. We approximate the integral by a sum of the integrand evaluated in quadrature points, multiplied by the quadrature weights. Again, the definition of quadrature/cubature formulas for the triangles is more involved than on quadrilaterals because the Gauss points defined in 1D cannot be directly extended. The most common strategy is to use tabulated point locations and weights. These are based on complicated expressions and usually solved by computer algebra tools. An important observation is that the number of quadrature points for integrating polynomials of degree $2N$ is usually larger than the number of nodes $N_p = (N+1)(N+2)/2$ due to the more complicated shape. In fact, most known formulas are relatively close to the naive approach of using a Gaussian quadrature with $(N+1)^2$ points that are compressed towards the upper end of the reference triangle.
Integration is the method of choice for curved boundaries where the mass and derivative matrices need to evaluated for each element separately. However, there exist optimizations for straight-sided elements.

Optimized matrices for affine transformations.

On straight-sided elements, the transformation is an affine function (i.e., a linear function with constant offset) in the unit variables,

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix} = -\frac{r+s}{2}\boldsymbol{v}^1 + \frac{r+1}{2}\boldsymbol{v}^2 + \frac{s+1}{2}\boldsymbol{v}^3,$$

for the three vertex locations $\boldsymbol{v}^i$ of the element $\mathcal{D}^k$. (In this text, we define the unit triangles as the three points $(-1,-1),(1,-1),(-1,1)$.)
For a linear transformation, the Jacobian matrix is constant throughout the element and depends only on the vertex locations,

$$\boldsymbol{J}^k = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{r}} = \frac{1}{2}\begin{bmatrix} v_1^2 - v_1^1 & v_1^3 - v_1^1 \\ v_2^2 - v_2^1 & v_2^3 - v_2^1 \end{bmatrix}.$$

Recall that for quadrilateral elements, this is only the case if the element shape is a parallelogram. For trapezoidal or more general shapes, the Jacobian depends on r and s due to the mixed-quadratic term rs.
This gives the following simplification of the *mass matrix*:

$$\mathcal{M}_{ij}^k = \int_{\mathsf{D}^k} \ell_i^k(\boldsymbol{x})\ell_j^k(\boldsymbol{x})\,d\boldsymbol{x} = \int_{\mathsf{I}} \ell_i(\boldsymbol{r})\ell_j(\boldsymbol{r})|\boldsymbol{J}^k|\,d\boldsymbol{r} = \left|\boldsymbol{J}^k\right| \int_{\mathsf{I}} \ell_i(\boldsymbol{r})\ell_j(\boldsymbol{r})\,d\boldsymbol{r},$$

where the geometric factor $\det(\boldsymbol{J}^k) = |\boldsymbol{J}^k|$ can be factored out. For multiplication with the inverse mass matrix $\mathcal{M}$, we can thus invert the unit mass matrix $\mathcal{M}^{\mathsf{I}}$ and account for the geometry by an additional factor $|\boldsymbol{J}^k|^{-1}$.
For the derivative matrix applied to the convective term, the following formula can be chosen:

- Compute derivative matrices for the reference element:

$$\left(\mathcal{S}_r^{\mathsf{I}}\right)_{ij} = \int_I \ell_i(\boldsymbol{r}) \frac{\partial \ell_j(\boldsymbol{r})}{\partial r}\, d\boldsymbol{r},$$
$$\left(\mathcal{S}_s^{\mathsf{I}}\right)_{ij} = \int_I \ell_i(\boldsymbol{r}) \frac{\partial \ell_j(\boldsymbol{r})}{\partial s}\, d\boldsymbol{r},$$

- For multiplication by $\mathcal{S}$ for the strong form, we multiply use the following matrices:

$$\mathcal{S}_x^k = |\boldsymbol{J}^k|(J_{11}^{-1}\mathcal{S}_r^{\mathsf{I}} + J_{21}^{-1}\mathcal{S}_s^{\mathsf{I}})$$
$$\mathcal{S}_y^k = |\boldsymbol{J}^k|(J_{11}^{-1}\mathcal{S}_r^{\mathsf{I}} + J_{22}^{-1}\mathcal{S}_s^{\mathsf{I}})$$

  For the weak form, multiplication by $\mathcal{S}^T$ is implemented analogously.

- The extension to 3D is straight-forward.

As for the mass matrix, only the reference-element matrices $\mathcal{S}_r^{\mathsf{I}}$ and $\mathcal{S}_s^{\mathsf{I}}$ need to be computed. In the computational algorithm, one simply builds the flux $\boldsymbol{f}(\boldsymbol{u}_h^k)$ of the nodal solution values and applies the above derivative formulas.
For the evaluation of face terms, $(d-1)$-dimensional matrices can be built where we may again factor out the geometry.
From a computational point of view, the above approach has two advantages:

- No integrals must be computed during the evaluation of the right hand side. In literature, such methods are referred to as quadrature-free methods.

- The algorithm multiplies by the same matrices $\mathcal{S}_r^{\mathsf{I}}$, $\mathcal{S}_s^{\mathsf{I}}$, and $(\mathcal{M}^{\mathsf{I}})^{-1}$ for all elements, which allows them to remain in caches of processors; thus, good utilization of modern processors can be obtained where often the memory access is the limiting factor.

4.4 Check yourself

- Compare the local statement of DG-FEM in 1D and in higher dimensions.

- Explain the computation of the local Lax–Friedrichs flux in two space dimensions.

- DG-FEM on quadrilaterals is usually based on tensor-product shape functions. How are the nodes distributed?

- How are nodes distributed on triangles?

- DG-FEM methods for quadrilaterals are predominantly implemented with quadrature, whereas DG-FEM for triangles and tetrahedra often use pre-computed matrices on the reference element and factored-out coefficients and geometry. Explain the reason for this difference.

- Give an outline of the computation of the advection and face terms in DG-FEM with quadrature.

- What is the idea of tensorial techniques for fast evaluation of integrals on quadrilaterals? Explain the main steps in the algorithm for evaluating the spatial derivative $\nabla_{\boldsymbol{r}}$ of shape functions.

- In which case is the advantage of the tensorial evaluation over the naive evaluation larger, for $N = 1$ or for $N = 4$? Give an estimation of the difference in cost.
- Explain the procedure for computing face integrals on meshes with hanging nodes. What modifications compared to the conforming case (grid without hanging nodes) are necessary?

5 Applications

5.1 Acoustic wave equation

The acoustic wave equation describes the propagation of waves in ideal materials. It represents a simplification of the linearized compressible Navier–Stokes equations by neglecting the viscous term (inviscid flow, i.e., Euler equations), linearizing around a background density level and assuming small deviations from this state, and inserting a linear relation between pressure and density, $p = C\rho$. This gives the following system of differential equations

$$\begin{aligned} \rho \frac{\partial \boldsymbol{v}}{\partial t} + \nabla p &= 0, \\ \frac{1}{c^2}\frac{\partial p}{\partial t} + \rho \nabla \cdot \boldsymbol{v} &= 0, \end{aligned} \tag{5.1}$$

where $\boldsymbol{v} = (v_1, v_2, \ldots, v_d)$ denotes the vector-valued velocity and p the pressure, in this context called the acoustic pressure. The constant c denotes the speed of sound and ρ the density.
In the above equation, the velocity can be eliminated from the second equation by taking its time derivative and replacing $\frac{\partial \boldsymbol{v}}{\partial t}$ by $-\frac{1}{\rho}\nabla p$ through the first equation, which gives the second-order form of the wave equation in acoustic pressure only,

$$\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} - \rho \nabla \cdot \left(\frac{1}{\rho} \nabla p \right) = 0.$$

For the discretization with the discontinuous Galerkin method, one usually considers the wave equation as a system (5.1) in terms of $\boldsymbol{v}$ and p.
The wave equation is closed by initial conditions for $\boldsymbol{v}(\boldsymbol{x}, 0)$ and $p(\boldsymbol{x}, 0)$ and boundary conditions. Exemplary boundary conditions are:

$$\begin{aligned} &p(\boldsymbol{x}, t) = p_D(\boldsymbol{x}, t) \quad \text{for } \boldsymbol{x} \in \partial\Omega_D \quad \text{(Dirichlet boundary conditions)}, \\ &\rho \boldsymbol{v} \cdot \hat{\boldsymbol{n}} = h_N(\boldsymbol{x}, t) \quad \text{for } \boldsymbol{x} \in \partial\Omega_N \quad \text{(Neumann (traction) boundary conditions)}, \\ &\rho \boldsymbol{v}(\boldsymbol{x}, t) \cdot \hat{\boldsymbol{n}} - \frac{1}{c} p(\boldsymbol{x}, t) = 0 \quad \text{for } \boldsymbol{x} \in \partial\Omega_A \quad \text{(absorbing boundary conditions)}. \end{aligned}$$

The absorbing boundary condition is used to model "open" boundaries on bounded computational domains: The waves should leave the computational domain Ω without any non-physical reflections. The stated condition is a very simple approach (a so-called first order condition) and reflects back up to a few percent of the wave in 2D and 3D. More sophisticated techniques are higher order absorbing boundary conditions or perfectly matched layers (PML).
The idea of a perfectly matched layers is to introduce a sponge layer outside of the region of interest where the equations are modified such that any contribution is dissipated. At the boundary of the PML layer, arbitrary boundary conditions can be set, e.g. Dirichlet conditions, since we expect most information to have vanished inside of the layer. The notion of "perfect matching" relates to the

requirement that the interface between the physical domain and the sponge layer should not give rise to any reflections by itself. To achieve this matching and introduce dissipation, auxiliary variables are introduced in the sponge layer (or one shifts into complex variables in the PML layer and imposes damping). By adjusting the thickness of the PML, one can control how much information will be reflected back into the physical domain. [See, e.g., A. Modave, E. Delhez, C. Geuzaine, Optimizing Perfectly Matched Layers in Discrete Contexts, *Int. J. Numer. Meth. Eng.* 99(6): 410–437, 2014.]
For the DG-FEM discretization of the discretized acoustic wave equation, the equations are multiplied by $\frac{1}{\rho}$ and c^2, respectively. On element D^k, this gives the following weak form:

$$\left(\frac{\partial \boldsymbol{v}_h}{\partial t}, \boldsymbol{\ell}_i^k\right)_{\mathsf{D}^k} - \left(p_h, \frac{1}{\rho}\nabla\cdot\boldsymbol{\ell}_i^k\right)_{\mathsf{D}^k} + \left\langle \left(\frac{p}{\rho}\right)^* \hat{\boldsymbol{n}}, \boldsymbol{\ell}_i^k\right\rangle_{\partial\mathsf{D}^k} = 0,$$
$$\left(\frac{\partial p}{\partial t}, \ell_i^k\right)_{\mathsf{D}^k} - \left(c^2\rho\boldsymbol{v}_h, \nabla\ell_i^k\right)_{\mathsf{D}^k} + \left\langle (c^2\rho\boldsymbol{v})^*\cdot\hat{\boldsymbol{n}}, \ell_i^k\right\rangle_{\partial\mathsf{D}^k} = 0.$$

In this equation, we need two numerical fluxes, one for defining $\left(\frac{p}{\rho}\right)^*$ in the first equation and the second for defining $(c^2\rho\boldsymbol{v})^*$ in the second equation. As usual $\hat{\boldsymbol{n}}$ denotes the outer unit normal vector on the boundary $\partial\mathsf{D}^k$.
Before deriving the numerical flux, we restrict the problem to a common application class in practice where density ρ and speed of sound c are constant within elements but may vary over element boundaries. This gives rise to scattering of waves along material boundaries which are of high technical relevance, as wave scattering can be used to detect material boundaries through inverse analysis for example. In order to distinguish the material parameters on two sides of an element, we write c^- and c^+ in analogy to the pressure and velocity variables.
We consider two versions of the numerical flux:

- **Local Lax–Friedrichs flux**:

$$\boldsymbol{f}^*(\boldsymbol{u}^-, \boldsymbol{u}^+) = \frac{1}{2}\left(\boldsymbol{f}(\boldsymbol{u}^-) + \boldsymbol{f}(\boldsymbol{u}^+)\right) + \frac{C}{2}\hat{\boldsymbol{n}}^-(\boldsymbol{u}^- - \boldsymbol{u}^+)$$

  where $\boldsymbol{u} = (\boldsymbol{v}, p)$ and $\boldsymbol{f}(\boldsymbol{u}) = \begin{bmatrix} p/\rho & 0 \\ 0 & p/\rho \\ \rho c^2 v_1 & \rho c^2 v_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_x(\boldsymbol{u}) & \boldsymbol{f}_y(\boldsymbol{u}) \end{bmatrix}$ (for 2D). In this equation, the multiplication $\hat{\boldsymbol{n}}^-(\boldsymbol{u}^- - \boldsymbol{u}^+)$ produces a tensor of rank two and size 3×2, compatible with $\boldsymbol{f}(\boldsymbol{u})$. For the computation of the constant C in the Lax–Friedrichs flux, we need to form partial derivatives of $\boldsymbol{f}$ with respect to $\boldsymbol{u}$ and multiply by the normal vector $\hat{\boldsymbol{n}}$ on the surface of the face. We get the following two components:

$$\frac{\partial \boldsymbol{f}_x}{\partial \boldsymbol{u}} = \begin{bmatrix} 0 & 0 & \frac{1}{\rho} \\ 0 & 0 & 0 \\ \rho c^2 & 0 & 0 \end{bmatrix}, \qquad \frac{\partial \boldsymbol{f}_y}{\partial \boldsymbol{u}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\rho} \\ 0 & \rho c^2 & 0 \end{bmatrix}.$$

  We assume multiplication by the normal vector $\hat{\boldsymbol{n}} = (\hat{n}_x, \hat{n}_y)^T$. This means that we need to find the eigenvalues of the matrix

$$\begin{bmatrix} 0 & 0 & \hat{n}_x\frac{1}{\rho} \\ 0 & 0 & \hat{n}_y\frac{1}{\rho} \\ \hat{n}_x\rho c^2 & \hat{n}_y\rho c^2 & 0 \end{bmatrix}, \quad \text{characteristic polynomial } \lambda(-\lambda^2 + \hat{n}_x^2 c^2 + \hat{n}_y c^2) = \lambda(-\lambda^2 + c^2).$$

  The largest eigenvalue is $\pm c$, which can be inserted into the above form.

Together with the normal vector in the face terms, the following terms are used for the Lax–Friedrichs flux:

$$\begin{aligned} \left(\frac{p}{\rho}\right)^* \boldsymbol{n}^- &= \frac{1}{2}\left(\left(\frac{p}{\rho}\right)^- + \left(\frac{p}{\rho}\right)^+\right)\boldsymbol{n}^- + \frac{\max(c^-, c^+)}{2}(\boldsymbol{v}^- - \boldsymbol{v}^+), \\ (\rho c^2 \boldsymbol{v})^* \cdot \hat{\boldsymbol{n}}^- &= \frac{(\rho c^2 \boldsymbol{v})^- + (\rho c^2 \boldsymbol{v})^+}{2} \cdot \hat{\boldsymbol{n}}^- + \frac{\max(c^-, c^+)}{2}(p^- - p^+). \end{aligned} \tag{5.2}$$

For imposing boundary conditions, several options exist. A simple approach for Dirichlet boundaries is to set $p^+ = p_D$ and symmetric conditions on the velocity, $v^- = v^+$. The imposed value for the velocity corresponds to a do-nothing condition and resembles the way outflow boundaries are handled for advection problems (recall that mathematically, no condition should be set at the outflow). Note that a slight variation often gives more accurate results, namely to define the "outer" pressure by a mirroring around the boundary value,

$$p^+ = -p^- + 2p_D.$$

For Neumann boundary conditions, the situation is reversed and we set $p^+ = p^-$ and $v^+ = -v^- + 2h_N/\rho$. Finally, for the case of absorbing boundary conditions, the extension is

$$p^+ = -p^- + 2c\rho v^- \cdot \hat{\boldsymbol{n}}^-.$$

- **Hybridizable discontinuous Galerkin (HDG) flux** [see N. C. Nguyen, J. Peraire, B. Cockburn, High-order implicit hybridizable discontinuous Galerkin methods for acoustics and elastodynamics, *J. Comput. Phys.* 230:3695–3718, 2011]: For this flux, an auxiliary variable λ_h on the faces is defined through

$$\lambda_h = \begin{cases} \frac{1}{2\tau}\left((\rho\boldsymbol{v})^- - (\rho\boldsymbol{v})^+\right)\cdot\hat{\boldsymbol{n}}^- + \frac{1}{2}\left(p^- + p^+\right), & \text{for interior faces,} \\ p_D, & \text{on Dirichlet boundaries,} \\ \frac{1}{\tau}\rho\boldsymbol{v}^-\cdot\hat{\boldsymbol{n}}^- + p^-, & \text{on Neumann boundaries,} \\ \frac{\tau}{\tau+\frac{1}{c}}\rho\boldsymbol{v}^-\cdot\hat{\boldsymbol{n}}^- + \frac{1}{2}p^-, & \text{on absorbing boundaries.} \end{cases} \tag{5.3}$$

With this new variable, the following realization of the numerical flux is chosen:

$$\begin{aligned} p^* &= \lambda_h. \\ (\rho\boldsymbol{v})^*\cdot\hat{\boldsymbol{n}} &= (\rho\boldsymbol{v})^-\cdot\hat{\boldsymbol{n}} + \tau\left(p^- - \lambda_h\right). \end{aligned} \tag{5.4}$$

The quantity λ_h can be interpreted as a pressure value on the faces between elements and is usually referred to as the *trace* of the pressure. The parameter τ represents a stabilization parameter that can be adjusted to the problem at hand. Large parameters τ force the difference between the pressure values to be small, $p^- \approx p^+$. A usual choice is to set $\tau = \frac{1}{\max(c^-, c^+)}$.

Note that the formulation of the flux if coefficients jump is not unique. Several different choices exist, depending on whether continuity on the primal variable p is imposed or rather $\frac{p}{\rho}$. Usually, the former is used as in the formulation of the HDG flux. On the other hand, the Lax–Friedrichs flux given above imposes the latter condition.
For constant coefficients ρ and c, the HDG flux is closely related to the Lax–Friedrichs flux if

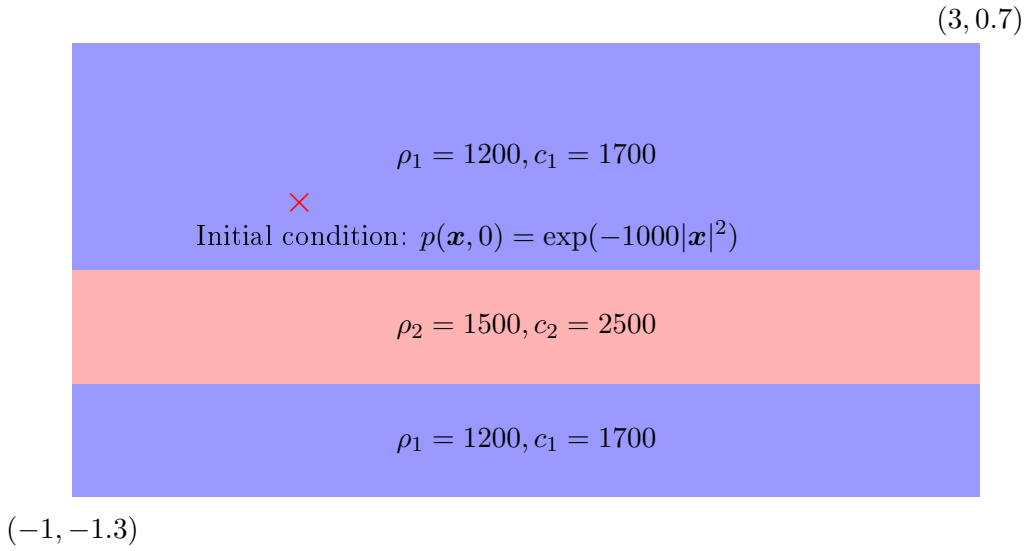
$$\tau = c\rho$$

is selected. Inserting the equation for λ_h into Equation (5.4) gives the following expression:

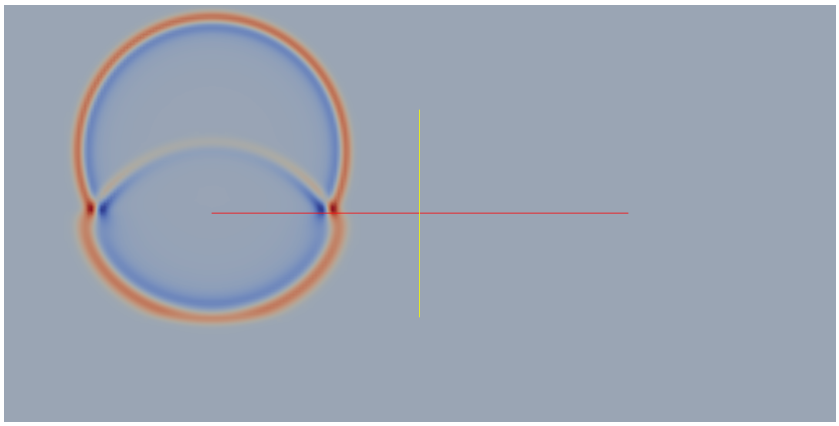
$$\frac{1}{\rho}p^*\hat{\boldsymbol{n}}^- = \frac{c}{2}\left((\boldsymbol{v}^- - \boldsymbol{v}^+)\cdot\hat{\boldsymbol{n}}\right)\hat{\boldsymbol{n}}^- + \frac{1}{2\rho}(p^- + p^+)\hat{\boldsymbol{n}}^-,$$
$$\rho c^2 \boldsymbol{v}^*\cdot\hat{\boldsymbol{n}} = \rho c^2 \frac{\boldsymbol{v}^- + \boldsymbol{v}^+}{2}\cdot\hat{\boldsymbol{n}}^- + \frac{c}{2}(p^- - p^+).$$

Comparing to (5.2) with constant coefficients, we see that the only difference is that the velocity contribution in the flux for p^* appears as $\boldsymbol{v}^- - \boldsymbol{v}^+$ in the Lax–Friedrichs flux, whereas the respective expression is $\left((\boldsymbol{v}^- - \boldsymbol{v}^+)\cdot\hat{\boldsymbol{n}}^-\right)\hat{\boldsymbol{n}}^-$ in the HDG flux. In one space dimension, this yields a completely equivalent method to the Lax–Friedrichs method with the mirrored boundary conditions. In higher dimensions, however, the HDG flux can give more accurate results than the Lax–Friedrichs flux if $\tau \approx \frac{1}{c}$, as shown by Nguyen, Peraire, and Cockburn.

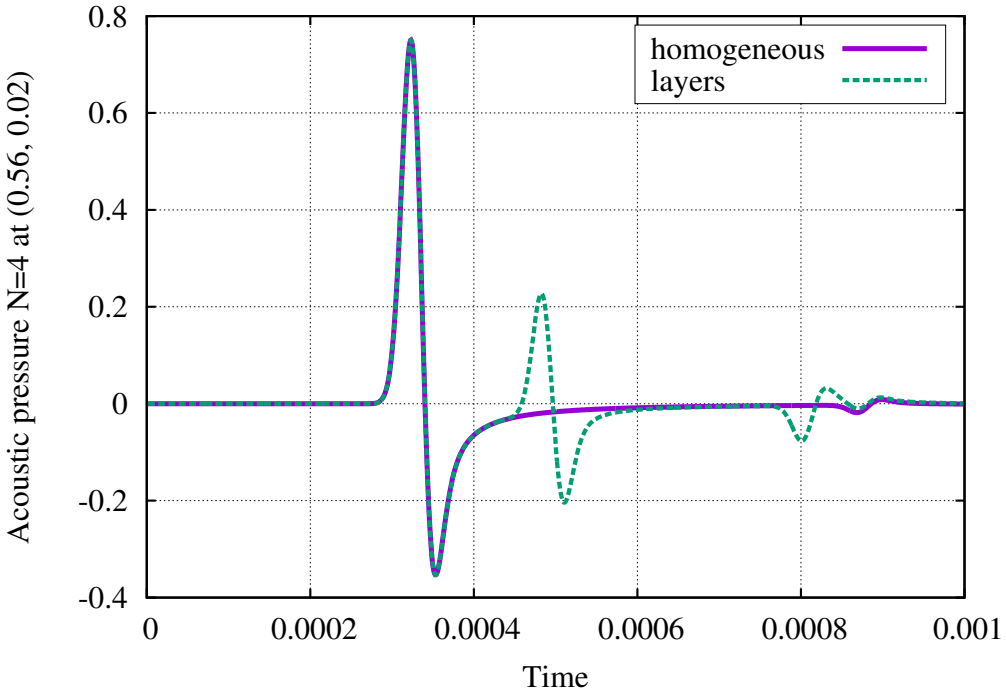
Computational example



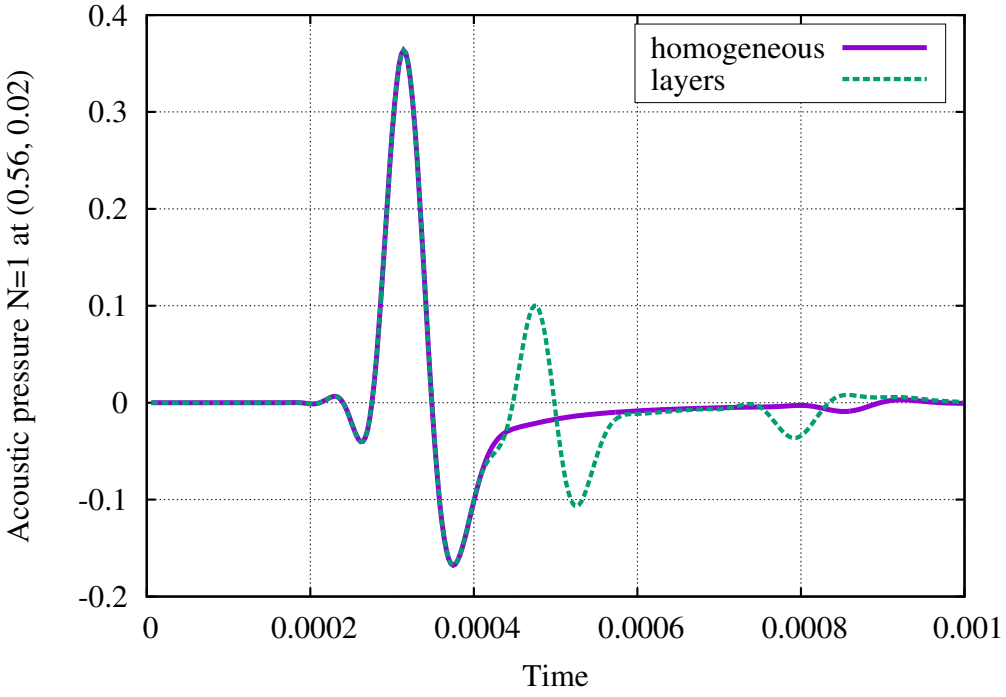
At time $t = 3.7 \cdot 10^{-4}$, we observe the following solution for the pressure:



We see the reflection of the wave from the material boundary in the center of domain (red line).
If we monitor the acoustic pressure in the point $(0.56, 0.02)$, we see the following picture if we compare the layered material with a situation where material 1 fills the whole domain:



We clearly see the initial peak as the original wave passes through the control point in both cases, and the reflection that distinguishes the layered case. We see a second reflection around time 0.0008 and finally a third one shortly thereafter. The last peak also appears in the homogeneous case and is due to the non-perfect nature of the non-reflective boundary condition.
Finally, let us compare with a solution at $N = 1$ instead of $N = 4$:



To summarize, the discontinuous Galerkin method is beneficial for the wave equation for several reasons:

- The solutions are often (piecewise) smooth. To resolve oscillations of high frequencies typical for wave propagation, high-order methods with considerably fewer degrees of freedom than low-order methods are appropriate (see discussion in Sec. 1).

- Material boundaries located at mesh boundaries, i.e., those exactly captured by the element boundaries, do not spoil the high order convergence. The flexible meshing in DG-FEM allows for taking such issues into account.
- If the wave propagation is coupled to other physics, the generality of the DG method allows for a natural coupling.
- There is a wide range of problems subject to the wave equation and related equations such as the elastic wave equation and Maxwell's equations, for example in seismics, material reconstruction, electromagnetics, etc.

For wave propagation on homogeneous materials over large domains, possibly even unbounded ones, driven by effects on the boundary, other numerical methods are preferred over DG-FEM. These include methods relying on fundamental solutions (Green's functions) that reduce the problem in a homogeneous domain onto a problem on the boundary. The leads to so-called boundary element methods (BEM). The solution on the boundary is more expensive per degree of freedom than solving the wave equation in the whole domain: Matrices are typically densely coupled because they include integrals over two variables with double integrals; fast solution techniques use e.g. hierarchical matrices and ideas akin to the fast multipole method. For large domains with high volume-to-area ratio, the lower dimensionality more than compensates for the more difficult equations. From the solution on the boundary, the interior solution is reconstructed through the fundamental solution.

5.2 Euler equations

Reading instructions: More details on the content of this lecture can be found in sections 5.9 (pp. 161–165) and 6.6 (pp. 206–236) of the course book [Hesthaven and Warburton, 2008].

The Euler equations are a simplification of the compressible Navier–Stokes equations in the inviscid limit. This means that the velocity of the fluids is so large that viscous forces are negligible compared to the inertial forces and are dropped from the equation. In terms of the Reynolds number that characterizes the Navier–Stokes equations, the limit $\mathrm{Re} \to \infty$ is represented by the Euler equations. Typical flow situations where this is justified is air flow around aircrafts in the supersonic case ($\mathrm{Ma} > 1$) or other flow situations with large velocities.
In conservative form, the compressible Euler equations are given by

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) &= 0, \\ \frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I}) &= \rho \boldsymbol{g}, \\ \frac{\partial E}{\partial t} + \nabla \cdot (\boldsymbol{u}(E+p)) &= 0, \end{aligned}$$

where ρ denotes the fluid density, $\boldsymbol{u} = (u, v)$ denotes the fluid velocity, and E the internal energy. The fluid pressure p is related to the other quantities by an equation of state. For an ideal gas, the relation is

$$p = (\gamma - 1)\left(E - \frac{1}{2}\rho \boldsymbol{u} \cdot \boldsymbol{u}\right),$$

where $\gamma = 1.4$ denotes the gas constant for a monoatomic gas.
The rank-2 tensor $\boldsymbol{u} \otimes \boldsymbol{u}$ is a so-called diadic tensor and given by the following expression in 2D ($\boldsymbol{u} = [u, v]$):

$$\boldsymbol{u} \otimes \boldsymbol{u} = \begin{bmatrix} uu & uv \\ vu & vv \end{bmatrix}, \quad \text{sometimes also written as } \boldsymbol{u}\boldsymbol{u}^T.$$

The origin of this term is the transport of the velocity field by itself (Reynolds transport theorem) in a fixed (Eulerian) frame of reference.
In component form and in two space dimensions, the Euler equations take the following form

$$\begin{aligned}
&\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} = 0,\\
&\frac{\partial \rho u}{\partial t} + \frac{\partial (\rho u^2 + p)}{\partial x} + \frac{\partial \rho uv}{\partial y} = \rho g_x,\\
&\frac{\partial \rho v}{\partial t} + \frac{\partial \rho uv}{\partial x} + \frac{\partial (\rho v^2 + p)}{\partial y} = \rho g_y,\\
&\frac{\partial E}{\partial t} + \frac{\partial u(E+p)}{\partial x} + \frac{\partial v(E+p)}{\partial y} = 0.
\end{aligned}$$

For the development of numerical schemes, we collect the solution variables and fluxes in vectors,

$$\begin{aligned}
\boldsymbol{q} &= [\rho, \rho u, \rho v, E]^T,\\
\boldsymbol{F} &= [\rho u, \rho u^2 + p, \rho uv, u(E+p)]^T,\\
\boldsymbol{G} &= [\rho v, \rho uv, \rho v^2 + p, v(E+p)]^T,
\end{aligned}$$

which allows us to write the Euler equations in concise vector form

$$\frac{\partial \boldsymbol{q}}{\partial t} + \frac{\partial \boldsymbol{F}}{\partial x} + \frac{\partial \boldsymbol{G}}{\partial y} = 0.$$

The implementation of the vector flux functions $\boldsymbol{F}$ and $\boldsymbol{G}$ involves two steps:

- Extract the primitive variables $[\rho, u, v, p]$ from the conserved variables $\boldsymbol{q} = [\rho, \rho u, \rho v, E]$.
- Compute $\boldsymbol{F}$ and $\boldsymbol{G}$ in the DG-FEM nodes.

(If working with quadrature, we first interpolate the conserved variables onto the quadrature points, extract the primitive variables, and evaluate the fluxes.)
The DG-FEM statement for the Euler equations on each element reads

$$\int_{\mathsf{D}^k} \left(\frac{\partial \boldsymbol{q}_h}{\partial t} \boldsymbol{\phi}_h - \boldsymbol{F}_h \frac{\partial \boldsymbol{\phi}_h}{\partial x} - \boldsymbol{G}_h \frac{\partial \boldsymbol{\phi}_h}{\partial y} \right) d\boldsymbol{x} + \int_{\partial \mathsf{D}^k} (\hat{n}_x \boldsymbol{F}_h + \hat{n}_y \boldsymbol{G}_h)^* \boldsymbol{\phi}_h \, d\boldsymbol{x} = 0.$$

5.2.1 Numerical fluxes

Due to the close relation of numerical flux functions in DG-FEM to the implementation of the flux in finite volume methods and the long history of finite volumes for compressible gas dynamics, a variety of fluxes are available for the Euler equations. These fluxes play an important role in the accuracy of the solution and can in some cases alleviate the need for limiters or filters.
The **local Lax–Friedrichs flux** for the Euler equations is

$$(\hat{n}_x \boldsymbol{F}_h + \hat{n}_y \boldsymbol{G}_h)^* = \hat{n}_x \{\{F_h\}\} + \hat{n}_y \{\{G_h\}\} + \frac{\lambda}{2} (\boldsymbol{q}^- - \boldsymbol{q}^-)$$

with the approximate local maximum acoustic wave speed

$$\lambda = \max_{s \in \{q_h^-, q_h^+\}} \left(|\boldsymbol{u}(s)| + \sqrt{\left| \frac{\gamma p(s)}{\rho(s)} \right|} \right).$$

More advanced variants are the following (see the discussion in the course book [Hesthaven & Warburton, 2008], section 6.6.2, pages 217–224):

Roe flux. The Roe flux is an extension of the local Lax–Friedrichs flux based on the concept of Riemann solvers. Riemann solvers are methods for solving conservation equations $\frac{\partial u}{\partial t} + \nabla \cdot f(u) = 0$ with piecewise constant data. This is the context of the numerical flux in finite volume methods that relates the average of solutions in neighboring elements. The derivation of Riemann solver depends on the particular form of the equations. For simple equations, including the Euler equations, exact Riemann solvers can be designed by looking at the eigenvalues of the flux $\frac{\partial f}{\partial u}$. For more complicated systems, approximate Riemann solvers are used.

The Roe flux is defined by the following formula:

$$\boldsymbol{f}^* = \{\{\boldsymbol{f}\}\} + \frac{1}{2}|\hat{\mathcal{A}}|[\![\boldsymbol{u}]\!],$$

where $\hat{\mathcal{A}}$ is a linearization of $\boldsymbol{f}$ along $\boldsymbol{u}$ in the normal direction $\hat{\boldsymbol{n}}$ and $|\hat{\mathcal{A}}| = \mathcal{U}|\Lambda|\mathcal{U}^{-1}$, taking absolute values in the diagonal entries of $|\Lambda|$.
In terms of the computation of $\hat{\mathcal{A}}$, the following linearization formula is assumed for the Roe flux

$$\hat{\mathcal{A}} = \int_0^1 \frac{d\boldsymbol{f}(\boldsymbol{u}(\xi))}{d\boldsymbol{u}} d\xi$$

with $\boldsymbol{u}(\xi) = \boldsymbol{u}^- + (\boldsymbol{u}^+ - \boldsymbol{u}^-)\xi$.

In the realization of the Roe flux, computing an integral for finding $\hat{\mathcal{A}}$ is too expensive. Therefore, some simple choices are

- $\hat{\mathcal{A}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(\{\{u\}\})$
- $\hat{\mathcal{A}} = \{\{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\}\}$

A more elaborate choice to integrates the flux Jacobian is based on the assumption of one strong wave. The idea is to use the linearization

$$\hat{\mathcal{A}}(\boldsymbol{u}^+ - \boldsymbol{u}^-) = s(\boldsymbol{u}^+ - \boldsymbol{u}^-)$$

for the propagation speed s.

In terms of the Euler equation, the realization of $\hat{\mathcal{A}}$ is based on a rotation of the momentum components $(\rho u, \rho v)$ into a coordinate system normal and tangential to a face is used [Hesthaven & Warburton, 2008, p. 220]. An important ingredient in these fluxes is the local speed of sound defined by

$$c^2 = (\gamma - 1)\left(H^* - \frac{(u^*)^2 + (v^*)^2}{2}\right),$$

based on averaged quantities

$$\begin{aligned}
\rho^* &= \sqrt{\rho^- \rho^+}, \\
u^* &= \frac{\sqrt{\rho^-}u^- + \sqrt{\rho^+}u^+}{\sqrt{\rho^-} + \sqrt{\rho^+}}, \\
v^* &= \frac{\sqrt{\rho^-}v^- + \sqrt{\rho^+}v^+}{\sqrt{\rho^-} + \sqrt{\rho^+}}, \\
H^* &= \frac{\sqrt{\rho^-}H^- + \sqrt{\rho^+}H^+}{\sqrt{\rho^-} + \sqrt{\rho^+}},
\end{aligned}$$

where

$$H = \frac{E + p}{\rho} \qquad \text{(enthalpy)}.$$

Harten-Lax-van-Leer (HLL). The Roe flux assumes one strong wave only and can therefore be too simple in complex situations. The HLL flux assumes that the Riemann problem consists of three states, separated by two waves s^- and s^+ with $s^+ > s^-$, which gives the flux

$$\boldsymbol{f}^*(\boldsymbol{q}^-, \boldsymbol{q}^+) = \begin{cases} \boldsymbol{f}(\boldsymbol{q}^-), & s^- \geq 0, \\ \dfrac{s^+\boldsymbol{f}(\boldsymbol{q}^-) - s^-\boldsymbol{f}(\boldsymbol{q}^+) + s^+s^-(\boldsymbol{q}^+ - \boldsymbol{q}^-)}{s^+ - s^-}, & s^- \leq 0 \leq s^+, \\ \boldsymbol{f}(\boldsymbol{q}^+), & s^+ \leq 0. \end{cases}$$

If $s^- \geq 0$, all information propagates to the right and simple upwinding is used. Similarly, for $s^+ \leq 0$ all information propagates to the left and we again use upwinding.

The parameters $s^\pm$ can be determined by the following formula:

$$s^- = \min\left(\lambda(\hat{\mathcal{A}}(\boldsymbol{q}^-)), \lambda(\hat{\mathcal{A}}(\boldsymbol{q}^*))\right),$$
$$s^+ = \min\left(\lambda(\hat{\mathcal{A}}(\boldsymbol{q}^+)), \lambda(\hat{\mathcal{A}}(\boldsymbol{q}^*))\right),$$

where $\boldsymbol{u}^*$ contains the weighted averages from the Roe flux (e.g. $\rho^* = \sqrt{\rho^-\rho^+}$.

The HLL flux can be extended to the case with more than two sound waves s^- and s^+ (contact waves) by the so-called Harten-Lax-van-Leer-Contact flux (HLLC), see the course book [Hesthaven & Warburton, 2008] for details.

5.2.2 Properties of Euler equations

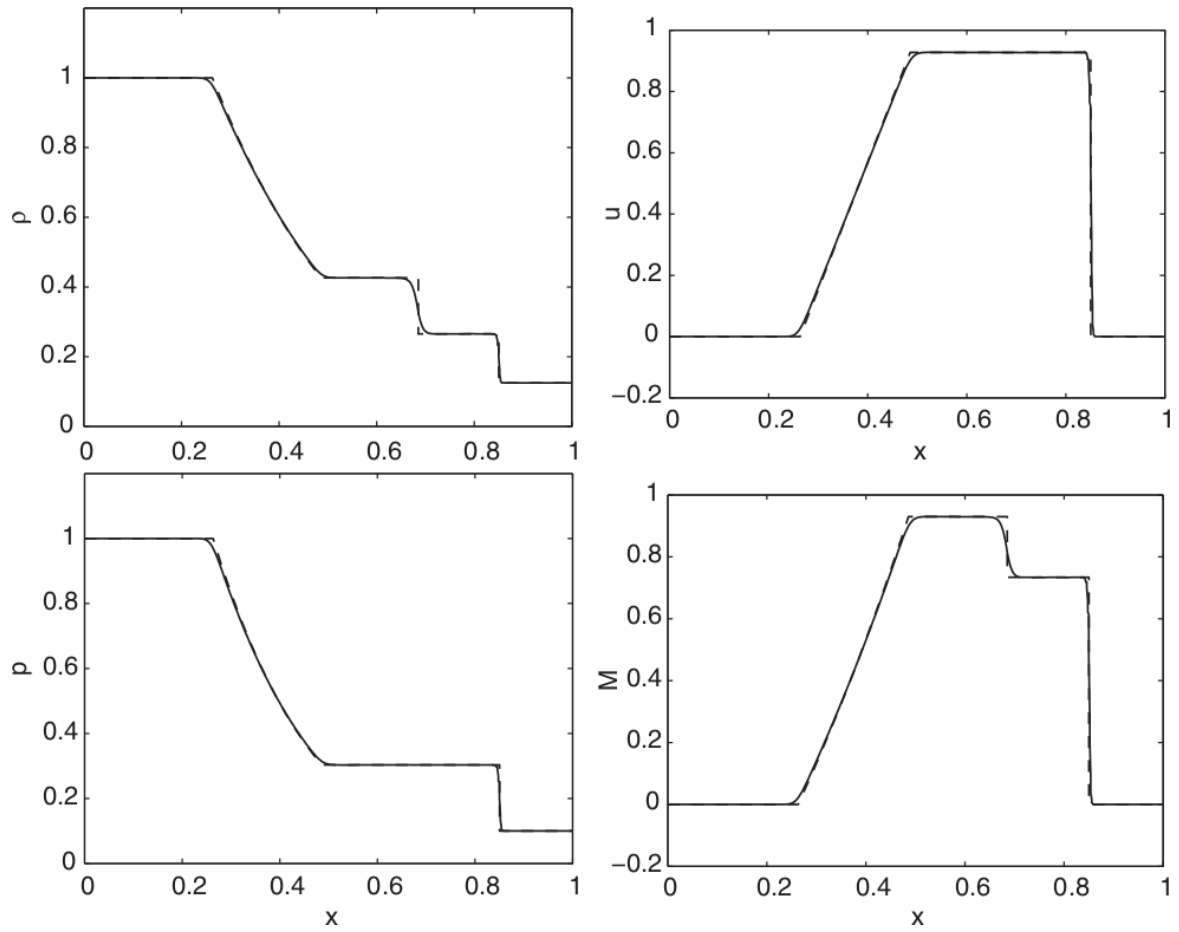
Due to the absence of dissipation, the Euler equations develop shocks where the values of the solution change rapidly. In general, this requires strong tools such as limiters. Filters alone are often not enough for the Euler equations because the they do not prevent oscillations. Thus, zero or negative values for the density appear, which renders the recovery of the primal variables $\boldsymbol{u}$ or p impossible in a physically meaningful way. Due to the nonlinearity, oscillations that appear near shocks (discontinuities) propagate quickly and destroy the solution.
For limiters, the one-dimensional concepts explained in Chapter 3 can be generalized. In general, most limiters belong to the TVBM (total variation bounded methods) class that are combined with simple positivity checks for the density and the pressure. The course book [Hesthaven & Warburton, 2008] describes the construction of a limiter for the Euler equations that keeps the solution gradients bounded. It is applied at the end of each Runge–Kutta stage. It takes the solution on a patch of four elements in 2D, the element and its direct neighbors and limits the jump of solution values (and hence the gradient) over faces. From a limited gradient and the cell average, a new solution with less oscillations is reconstructed.

1D Example: Sod's shock tube

A simple test example is the following one-dimensional problem, so-called Sod's shock tube problem:

$$\rho(x,0) = \begin{cases} 1, & x < 0.5, \\ 0.125, & x \geq 0.5, \end{cases}$$
$$u(x,0) = 0,$$
$$E(x,0) = \frac{1}{\gamma - 1}\begin{cases} 1, & x < 0.5, \\ 0.1, & x \geq 0.5. \end{cases}$$

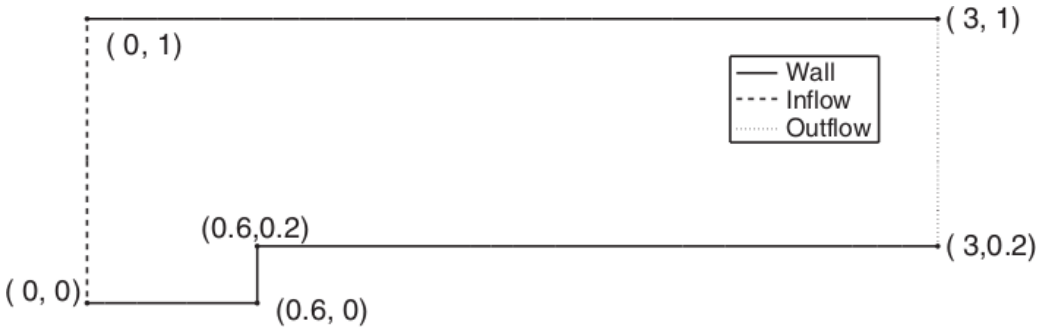
The solution for $K = 250$ elements with $N = 1$ (linear basis functions) and a MUSCL TVBM limiter is shown in the following figure (reprinted from [Hesthaven & Warburton, 2008, Fig. 5.14]):



The figure displays the density ρ, the velocity u, the pressure p, and the local Mach number $M = \frac{u}{c}$ with c the speed of sound.
The contact discontinuity around $x = 0.7$ can be identified by a discontinuity in the density (and energy, not shown) but continuity in the velocity and the pressure.

2D Example: Forward facing step

A second example is given by the flow on a forward facing step also considered in one of the course projects. It is given by the following geometry:

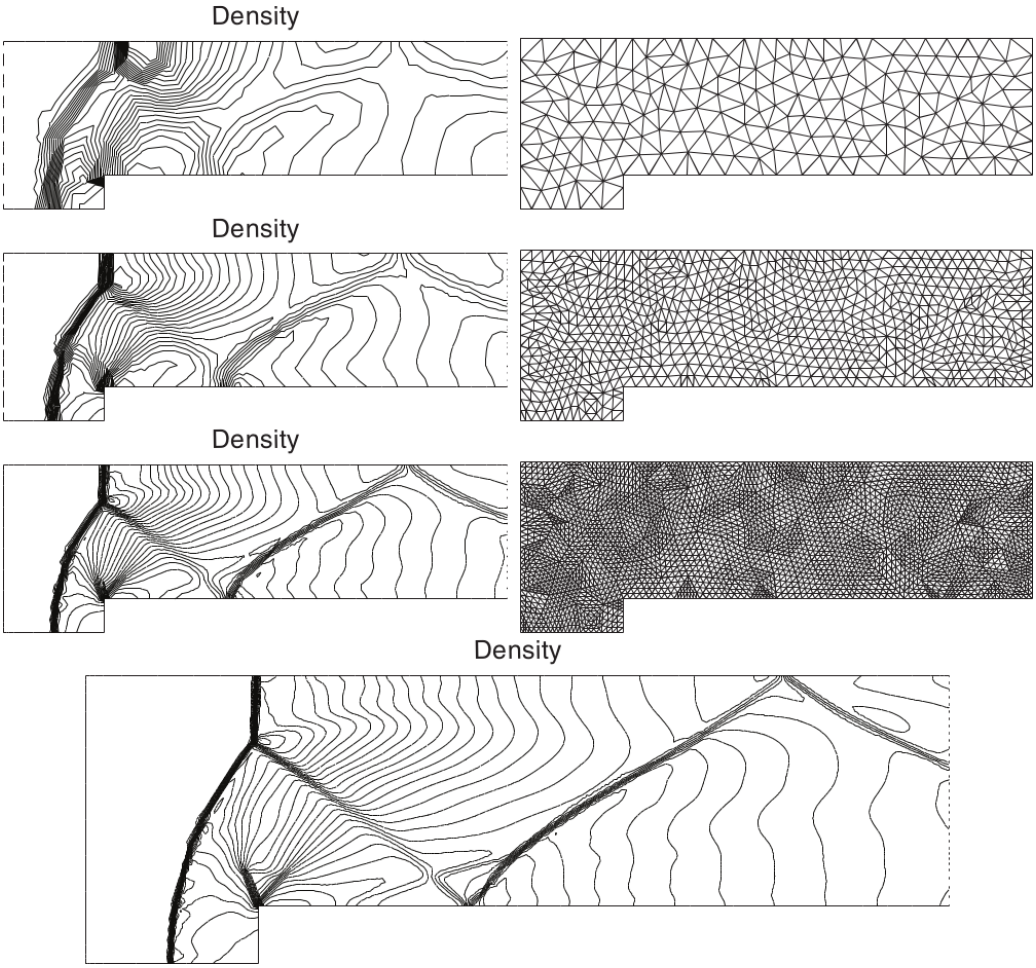


At the inflow, the following values are given:
$\rho = \gamma$, $\rho u = 3\gamma$, $\rho v = 0$, $E = \frac{1}{\gamma - 1} + \frac{9\gamma}{2}$.

For the walls, reflective conditions of the form

$$\begin{aligned}
\rho^+ &= \rho^-, \\
\rho u^+ &= \rho u^- - 2(n_x \rho u + n_y \rho v)^- n_x, \\
\rho v^+ &= \rho v^- - 2(n_x \rho u + n_y \rho v)^- n_y, \\
E^+ &= E^-.
\end{aligned}$$

are applied. The outflow conditions are $\boldsymbol{q}^+ = \boldsymbol{q}^+$.
Computational results are shown in the following figure (reprinted from [Hesthaven & Warburton, 2008, Fig. 6.14]), using the Roe flux and the Euler limiter developed in the course book:



5.3 Check yourself

- How is the Lax–Friedrichs flux derived for the acoustic wave equation?
- What is the typical context in which DG-FEM is one of the preferred solution methods for the acoustic wave equation?
- Give a short overview over numerical fluxes for the Euler equations and the application needs.

6 Equations with Second Derivatives

6.1 Methods for time-dependent problems

The methods presented in the first five chapters deal with DG-FEM methods for partial differential equations with first spatial derivatives. In many applications, second derivatives need to be discretized as well. As opposed to continuous finite elements where the H^1 regularity of the ansatz spaces makes the representation of second derivative operators natural by the weak form with integration by parts, this simple extension does not work for DG-FEM where the ansatz for the numerical approximation is discontinuous (broken) over element boundaries, i.e., it is only of regularity L_2.
We note that DG-FEM methods for second derivatives are most widespread in discretizing problems where standard FEM are insufficient. For elliptic problems with nice coefficients, continuous finite elements are mathematically optimal and usually more efficient than discontinuous approaches (fewer degrees of freedom, no face integrals). Situations where discontinuous approaches are preferred are for example:

- Conservation laws where the first derivative is strong and should be represented accurately with DG-FEM, e.g., in fluid dynamics, or
- Strong jumps in the coefficients that require careful discretization for accuracy and stability reasons, e.g., for flow in porous media.

As a model problem, we consider the time-dependent heat equation,

$$\frac{\partial u}{\partial t} = \nabla \cdot a(\boldsymbol{x}) \nabla u = \sum_{i=1}^{d} \frac{\partial}{\partial x_i} a(\boldsymbol{x}) \frac{\partial u}{\partial x_i},$$

or, in one space dimension,

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} a(x) \frac{\partial u}{\partial x}.$$

We assume the coefficient $a(x) > 0$ to be positive and set boundary conditions on the whole boundary. For the heat equation, possible conditions are

- Dirichlet conditions $u(x,t) = g$ (prescribing a fixed temperature), or
- Neumann conditions $\frac{\partial u}{\partial x} = h$ (prescribing a heat flux).

The Dirichlet and Neumann parts of the boundary do not overlap.

6.1.1 Transformation into first-order system

One way to treat these problems with DG-FEM is to rewrite the equation into an equivalent first-order system of equations. Technically, this approach has been chosen for the acoustic wave equation from

a perspective of the second-order equation $\frac{\partial^2 p}{\partial t^2} - \frac{\partial^2 p}{\partial x^2} = 0$. Splitting into systems of lower derivative equations is a general technique in the discretization of differential equations, think about the case of integrating second or higher order time derivatives with standard methods for first-order time derivatives (e.g. Runge–Kutta methods) by auxiliary variables.
For the heat equation, we introduce the following system

$$\begin{aligned}\frac{\partial u}{\partial t} &= \frac{\partial}{\partial x}\sqrt{a}q\\ q &= \sqrt{a}\frac{\partial u}{\partial x},\end{aligned}$$

which extends to higher dimension by the following vector notation:

$$\begin{aligned}\frac{\partial u}{\partial t} &= \nabla\cdot\sqrt{a}\boldsymbol{q},\\ \boldsymbol{q} &= \sqrt{a}\nabla u,\end{aligned}$$

with the vector $\boldsymbol{q} = [q_1, q_2, \ldots, q_d]$ in d space dimensions.
This equation is now discretized with the techniques for first derivatives. We introduce the ansatz functions for element $k \in \{1, 2, \ldots, K\}$ for both u and q,

$$\begin{aligned}u_h^k(x,t) &= \sum_{i=0}^{N} u_h^k(x_i,t)\ell_i^k(x),\\ q_h^k(x,t) &= \sum_{i=0}^{N} q_h^k(x_i,t)\ell_i^k(x).\end{aligned}$$

where N is the polynomial degree.
This gives the following strong form:

$$\begin{aligned}\mathcal{M}^k\frac{d\boldsymbol{u}_h^k}{dt} &= \tilde{\mathcal{S}}^{\sqrt{a}}\boldsymbol{q}_h^k - \int_{\partial \mathsf{D}^k}\hat{\boldsymbol{n}}\cdot\left((\sqrt{a}q_h^k) - (\sqrt{a}q_h^k)^*\right)\boldsymbol{\ell}^k(x)dx,\\ \mathcal{M}^k\boldsymbol{q}_h^k &= \mathcal{S}^{\sqrt{a}}\boldsymbol{u}_h^k - \int_{\partial \mathsf{D}^k}\hat{\boldsymbol{n}}\cdot\left((\sqrt{a}u_h^k) - (\sqrt{a}u_h^k)^*\right)\boldsymbol{\ell}^k(x)dx,\end{aligned}$$

and the corresponding weak form

$$\begin{aligned}\mathcal{M}^k\frac{d\boldsymbol{u}_h^k}{dt} &= -\left(\mathcal{S}^{\sqrt{a}}\right)^T\boldsymbol{q}_h^k + \int_{\partial \mathsf{D}^k}\hat{\boldsymbol{n}}\cdot(\sqrt{a}q_h^k)^*\boldsymbol{\ell}^k(x)dx,\\ \mathcal{M}^k\boldsymbol{q}_h^k &= -\left(\tilde{\mathcal{S}}^{\sqrt{a}}\right)^T\boldsymbol{u}_h^k + \int_{\partial \mathsf{D}^k}\hat{\boldsymbol{n}}\cdot(\sqrt{a}u_h^k)^*\boldsymbol{\ell}^k(x)dx.\end{aligned}$$

In these equations, we use the special advection operators

$$\tilde{S}_{ij}^{\sqrt{a}} = \int_{\mathsf{D}^k}\ell_i^k(x)\frac{d\sqrt{a(x)}\ell_j^k(x)}{dx}\,dx, \quad S_{ij}^{\sqrt{a}} = \int_{\mathsf{D}^k}\sqrt{a(x)}\ell_i^k(x)\frac{d\ell_j^k(x)}{dx}\,dx.$$

Simplified implementation

In the form above, we assumed modified first derivative operators $\tilde{\mathcal{S}}^{\sqrt{a}}$ and $\mathcal{S}^{\sqrt{a}}$. To simplify, inexact integration according to

$$\begin{aligned}\mathcal{M}^k\frac{d\boldsymbol{u}_h^k}{dt} &= \mathcal{S}^k\mathcal{A}^k\boldsymbol{q}_h^k - \int_{\partial \mathsf{D}^k}\hat{\boldsymbol{n}}\cdot\left((\sqrt{a}q_h^k) - (\sqrt{a}q_h^k)^*\right)\boldsymbol{\ell}^k(x)dx,\\ \mathcal{M}^k\boldsymbol{q}_h^k &= \mathcal{A}^k\mathcal{S}^k\boldsymbol{u}_h^k - \int_{\partial \mathsf{D}^k}\hat{\boldsymbol{n}}\cdot\left((\sqrt{a}u_h^k) - (\sqrt{a}u_h^k)^*\right)\boldsymbol{\ell}^k(x)dx,\end{aligned}$$

can be applied, where $\mathcal{A}_{ii}^k = \sqrt{a(x_i^k)}$ collects the coefficients a in a diagonal matrix. A similar approach can be used for the weak form.
Of course, this gives rise to an aliasing error:

- The situation is less critical for the heat equation than for conservation laws because the inherent dissipation makes solutions smooth and well-resolved.
- If needed, stabilization by filters can be applied analogously to the first derivative case.

Furthermore, the following approximation is used in certain implementations:

$$\mathcal{M}^{-1}\mathcal{A}\mathcal{S} \approx \mathcal{A}\mathcal{D}_r,$$

with the usual differentiation matrix $\mathcal{D}_r = \mathcal{M}^{-1}\mathcal{S}$.
This second simplification interchanges the product $\mathcal{M}^{-1}\mathcal{A}$ by $\mathcal{A}\mathcal{M}^{-1}$. The effect is similar to aliasing as long as the coefficients a does not vary too much. In that case, the accuracy of the method is not affected. The picture changes if a varies strongly inside elements, though. Alternatives are numerical integration through quadrature where these simplifications are not necessary (see Section 4.2). However, the error due to a limited number of quadrature points can have a similar effect.

6.1.2 Numerical fluxes

For the numerical fluxes $(\sqrt{a}q_h)^*$ and $(\sqrt{a}u_h)^*$, connections between the components q_h and u_h are possible (and desired). The most general definition is the following:

$$\begin{aligned}(\sqrt{a}q_h)^* &= f\left((\sqrt{a}q_h)^-, (\sqrt{a}q_h)^+, (\sqrt{a}u_h)^-, (\sqrt{a}u_h)^+\right),\\ (\sqrt{a}u_h)^* &= g\left((\sqrt{a}q_h)^-, (\sqrt{a}q_h)^+, (\sqrt{a}u_h)^-, (\sqrt{a}u_h)^+\right).\end{aligned}$$

These definition express the fact that the flux can involve the left and right values of both q_h and u_h on the face $\partial\mathsf{D}^k$. A disadvantage of this flux is the fact that this introduces a strong coupling between the equations of u_h and q_h over the faces.
A common simplification is to restrict the choice of the flux by the following:

$$\begin{aligned}(\sqrt{a}q_h)^* &= f\left((\sqrt{a}q_h)^-, (\sqrt{a}q_h)^+, (\sqrt{a}u_h)^-, (\sqrt{a}u_h)^+\right),\\ (\sqrt{a}u_h)^* &= g\left((\sqrt{a}u_h)^-, (\sqrt{a}u_h)^+\right).\end{aligned}$$

To understand the effect of the simplification, let us look at the structure of the final matrix system,

$$\begin{bmatrix}\mathcal{M} & 0\\ 0 & \mathcal{M}\end{bmatrix}\begin{bmatrix}\frac{d\boldsymbol{u}_h}{dt}\\ \boldsymbol{q}_h\end{bmatrix} = \begin{bmatrix}\mathcal{E} & \mathcal{B}\\ \mathcal{C} & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{u}_h\\ \boldsymbol{q}_h\end{bmatrix}, \tag{6.1}$$

where

- the matrix $\mathcal{M}$ represents the collection of all element mass matrices (block-diagonal),
- $\mathcal{E}$ contains matrix entries for representing the terms $(\sqrt{a}u_h)^-, (\sqrt{a}u_h)^+$ of the numerical flux $(\sqrt{a}q_h)^*$,
- $\mathcal{B}$ contains the matrix $\tilde{\mathcal{S}}^{\sqrt{a}}$ and contributions of the terms $(\sqrt{a}q_h)^-, (\sqrt{a}q_h)^+$ in the numerical flux $(\sqrt{a}q_h)^*$,
- $\mathcal{C}$ contains the matrix $\mathcal{S}^{\sqrt{a}}$ and contributions of the the terms $(\sqrt{a}u_h)^-, (\sqrt{a}u_h)^+$ in the numerical flux $(\sqrt{a}u_h)^*$.

If we assume an explicit time stepping method, we can compute the value for $\boldsymbol{q}_h$ in terms of the solution $\boldsymbol{u}_h$ by multiplication with the matrix $\mathcal{C}$, i.e., integrals on the cells and faces and multiplication by the inverse mass matrix.

Boundary conditions

For boundary conditions, the following definitions of the "external" values (u_h^+, q_h^+) are widespread:

- Dirichlet boundary conditions

$$u_h^+ = 2g - u_h^-, \quad q_h^+ = q_h^- \quad \Rightarrow \begin{cases} \{\{u_h\}\} = g, & [\![u_h]\!] = 2\hat{\boldsymbol{n}}^-(u_h^- - g), \\ \{\{q_h\}\} = q_h^-, & [\![q_h]\!] = 0. \end{cases}$$

- Neumann boundary conditions

$$u_h^+ = u_h^-, \quad q_h^+ = 2h - q_h^- \quad \Rightarrow \begin{cases} \{\{u_h\}\} = u_h^-, & [\![u_h]\!] = 0, \\ \{\{q_h\}\} = h, & [\![q_h]\!] = 2\hat{\boldsymbol{n}}^-(q_h^- - h). \end{cases}$$

6.1.3 First method: central flux

The central flux for the heat equation reads as follows:

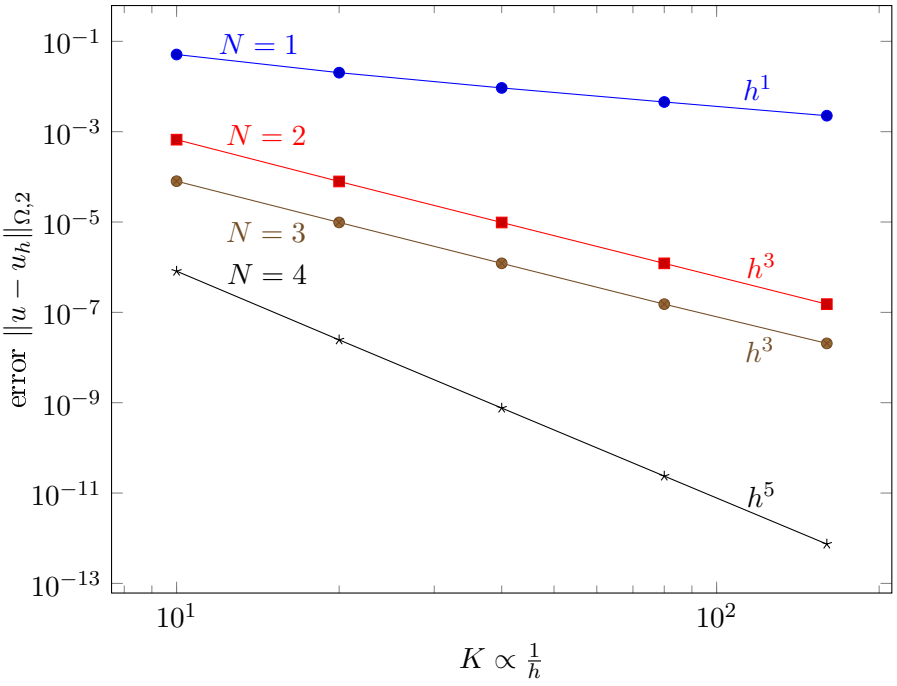
$$(\sqrt{a}q_h)^* = \{\{\sqrt{a}q_h\}\}, \quad (\sqrt{a}u_h)^* = \{\{\sqrt{a}u_h\}\}.$$

Convergence and stability

It is shown in the course book [Hesthaven & Warburton, 2008] that the discretization of the heat equation with central fluxes is **stable**. However, the convergence is suboptimal:
For the heat equation with given initial condition,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad x \in [0, 2\pi], \qquad u(x,0) = \sin(x), \qquad u(0,t) = 0, u(2\pi,t) = 0.$$

simulated until time $t = 0.8$ with analytic solution $u(x,t) = \exp(-t)\sin(x)$, the convergence is only optimal of rate $\mathcal{O}(h^{N+1})$ for N even, whereas it is $\mathcal{O}(h^N)$ for N odd:



This is in line with theoretical results which predict:

$$\|u_h(\cdot,T) - u(\cdot,T)\|_\Omega^2 + \int_0^T \|q_h(\cdot,t) - q(\cdot,t)\|_\Omega^2 dt \leq Ch^{2N},$$

with the constant $C = \mathcal{O}(1)$ for N odd and $C = \mathcal{O}(h^2)$ for N even.

Selection of time step size for explicit time stepping

If the solution is advanced in time using an explicit Runge–Kutta method, the following restriction arises:

$$\Delta t \leq C \min_{i,k} \frac{1}{a(x_i^k)} \left(\Delta x_i^k\right)^2$$

where $a(x_i^k)$ denotes the value of the coefficient in all node points on all elements and Δx_i^k is the distance between two nodes. For constant coefficients a, the relation reduces to the minimum distance between the Gauss–Lobatto nodes on all elements according to $h_{\min}(r_1 - r_0)$.
As compared to the advection equation where the time step limit is Δx_i^k, the second derivatives forces considerably smaller time steps. This result is due to the maximum eigenvalue in the derivative operator that multiplies the value of two first-derivative operators. (Idea for why this is the case: We need to consider the largest eigenvalue in the matrix from Eq. (6.1); this involves products of terms of the form $\mathcal{BC}$, which both contain a first derivative operator.)
For these reasons, explicit time stepping with second derivatives is much less attractive than for conservation laws, unless the coefficients a are very small as compared to other terms in the equation (reaction coefficients, convection). Instead, methods that solve the implicit systems of equations with techniques for elliptic operators (e.g. multigrid methods) are preferred.

6.1.4 Local discontinuous Galerkin methods

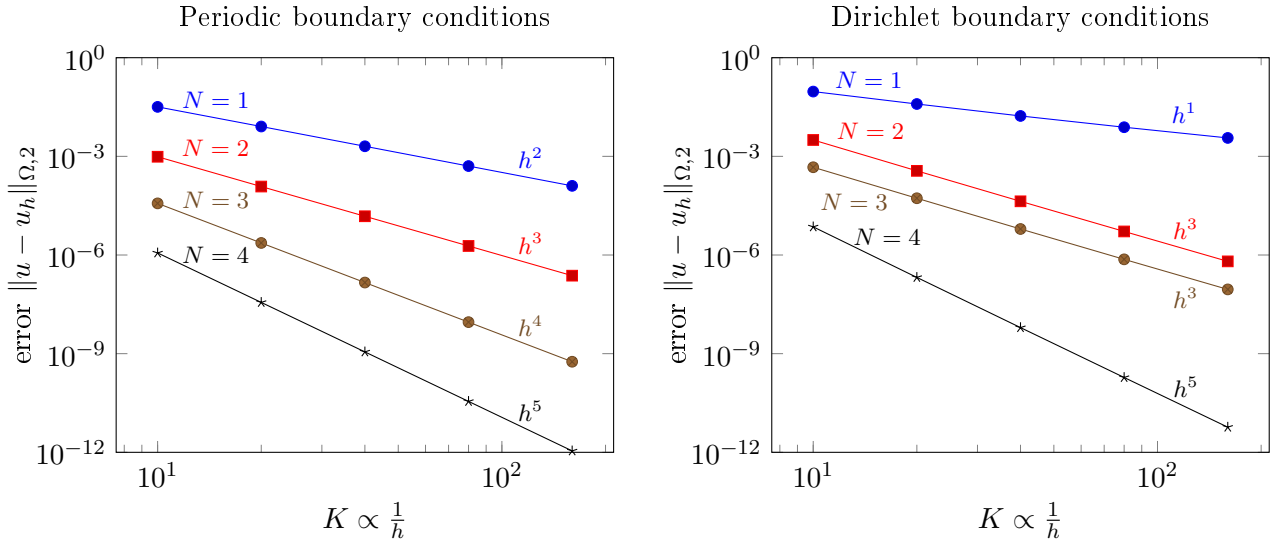
The suboptimal convergence in the simple central flux seen in the previous subsection raises the question whether better convergence can be obtained by modified numerical fluxes. One variant are so-called *local discontinuous Galerkin (LDG) methods* with the following choice:

$$\begin{aligned}(\sqrt{a}q_h)^* &= \{\{\sqrt{a}q_h\}\} + \hat{\boldsymbol{\beta}} \cdot [\![\sqrt{a}q_h]\!], \\ (\sqrt{a}u_h)^* &= \{\{\sqrt{a}u_h\}\} - \hat{\boldsymbol{\beta}} \cdot [\![\sqrt{a}u_h]\!],\end{aligned}$$

where either $\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{n}}$ or $\hat{\boldsymbol{\beta}} = -\hat{\boldsymbol{n}}$.
The LDG methods use upwinding even though the heat equation does not have a preferred direction of information flux. However, the final result is a symmetric operation because the upwinding is done in *opposite* directions for u_h and q_h. The opposite directions are also important for stability.
This gives the following convergence result for periodic boundary conditions:

$$\|u_h(\cdot,T) - u(\cdot,T)\|_\Omega^2 + \int_0^T \|q_h(\cdot,t) - q(\cdot,t)\|_\Omega^2 dt \leq C h^{2N+2}.$$

However, when the method is extended to Dirichlet boundary conditions by the selections made above (inserting u^+ and q^+ into the LDG flux), the optimal convergence is lost and a situation similar to the central flux is recovered. On the left, we report the convergence for the heat equation with periodic boundary conditions, and on the right convergence with homogenous Dirichlet boundary conditions is shown:



6.1.5 Extension to more general problems

The central or the LDG flux developed in this section can also be applied to mixed problems where both first and second order derivatives need to be represented. For the convection–diffusion equation of the form

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f(u) = \frac{\partial}{\partial x} a(x) \frac{\partial u}{\partial x},$$

combined with suitable initial and boundary conditions, the following transformation to a first-order system is applied:

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} \left(f(u) - \sqrt{a} q \right) &= 0, \\ q &= \sqrt{a} \frac{\partial u}{\partial x}. \end{aligned}$$

For discretization, separate fluxes are used for the terms from the diffusive operator and the convective operator. For the former, the methods derived above are used, whereas for Lax–Friedrichs type fluxes can be used f^* (or other, more sophisticated, fluxes). Such a method is implemented in the programs supplementing the course book, for example `BurgersRHS1D.m`.
Likewise, the extension to higher space dimensions is straight-forward using the vector-valued flux $\boldsymbol{q} = \nabla u$.

6.2 Methods for elliptic problems

Let us now assume we solve a stationary equation of the form

$$-\frac{\partial^2 u}{\partial x^2} = \sin(x), \quad x \in [0, 2\pi], \qquad u(0) = u(2\pi) = 0.$$

The exact solution is $u(x) = \sin(x)$. However, the method derived above is not stable. This manifests itself in a singular matrix system when solving for $\boldsymbol{u}_h$,

$$\mathcal{A} \boldsymbol{u}_h = \boldsymbol{f}_h,$$

where the matrix $\mathcal{A}$ is derived from a block system similar to (6.1),

$$\begin{bmatrix} \mathcal{E} & \mathcal{B} \\ \mathcal{C} & -\mathcal{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_h \\ \boldsymbol{q}_h \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_h \\ 0 \end{bmatrix}.$$

By looking at the second equation, we realize that $\boldsymbol{q}_h = \mathcal{M}^{-1}\mathcal{C}\boldsymbol{u}_h$, where $\mathcal{M}$ is the mass matrix. This is because we assumed the flux $(\sqrt{a}u_h)^*$ to only depend on $(\sqrt{a}u_h)^-$ and $(\sqrt{a}u_h)^+$ but not the values of q_h. Using this expression, $\boldsymbol{q}_h$ can be eliminated from the first equation, giving:

$$\mathcal{E}\boldsymbol{u}_h + \mathcal{B}\boldsymbol{q}_h = \mathcal{E}\boldsymbol{u}_h + \mathcal{B}\mathcal{M}^{-1}\mathcal{C}\boldsymbol{u}_h = \boldsymbol{f}_h,$$

which gives the following system matrix:

$$\mathcal{A} = \mathcal{E} + \mathcal{B}\mathcal{M}^{-1}\mathcal{C}.$$

Matrix $\mathcal{A}$ is sparse similar to finite elements because $\mathcal{M}^{-1}$ is block-diagonal.
The problem with the discretization are zero eigenvalues, so-called spurious modes, of the form $\boldsymbol{u}^- = -\boldsymbol{u}^+$, which vanish for the central flux that involves only $\{\{u_h\}\}$. In the time-dependent case, these do not cause problems because they remain unaltered by the time discretization method (zero eigenvalue $\rightarrow$ corresponding modes does not get activated) and typically do not appear in the initial condition.

6.2.1 Stabilization for the central flux

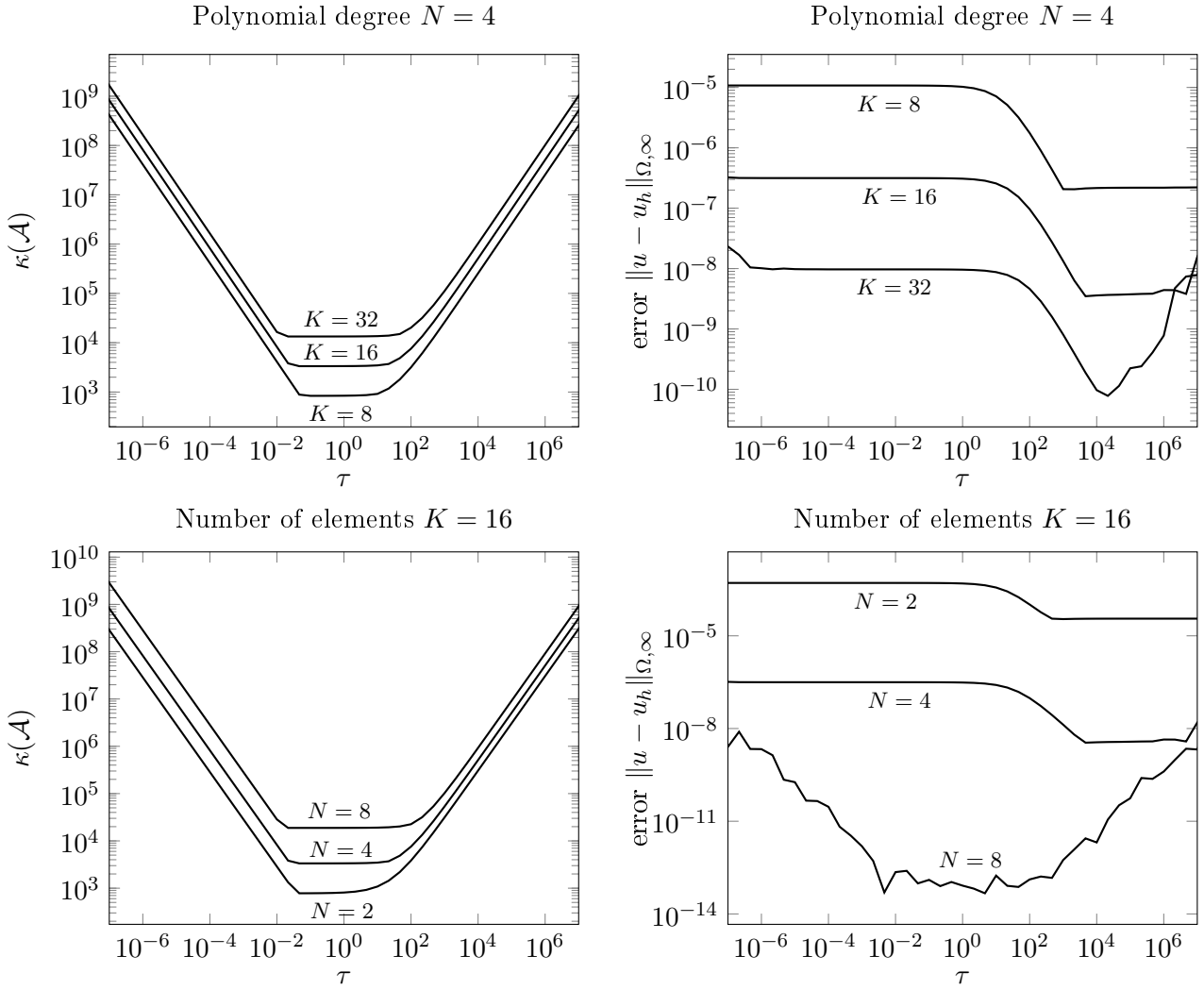
For the stationary problem, we need to prevent jumps of the form $\boldsymbol{u}^- = -\boldsymbol{u}^+$ by modification of the numerical flux.
For the central flux, the choice

$$q^* = \{\{q\}\} - \tau[\![u]\!], \qquad u^* = \{\{u\}\},$$

fixes the problem. This term acts as a penalization of the jump of u_h over the faces. Since the DG-FEM approach will, similarly to finite elements, seek to minimize the "energy" in terms of the discrete form, including a term of the form $\langle [\![u_h]\!], \tau [\![u_h]\!] \rangle_{\Gamma_h}$ prevents this mode from appearing.
The parameter τ controls the properties of the stabilized central flux:

- Choosing τ too small will not add enough penalty to the unphysical term. This is expressed in a matrix $\mathcal{A}$ which is still close to singular and has large condition numbers.
- Choosing τ too large, the penalization will dominate the other physically relevant terms. This is again a matrix with large condition numbers.

Therefore, we expect a range of values of τ which give best results. We check the conditioning $\kappa(\mathcal{A})$ (left panels) and the approximation quality (right panels) for several mesh sizes and polynomial degrees in the following figure:



We observe that the condition number behaves as

$$\kappa(\mathcal{A}) = \begin{cases} \tau^{-1}, & \tau \ll 1, \\ \tau, & \tau \gg 1, \end{cases}$$

with the best conditioning for $\tau = \mathcal{O}(1)$. However, the approximation becomes better if τ gets larger. Regarding the overall convergence behavior of the central flux with $\tau = 1$, we have:

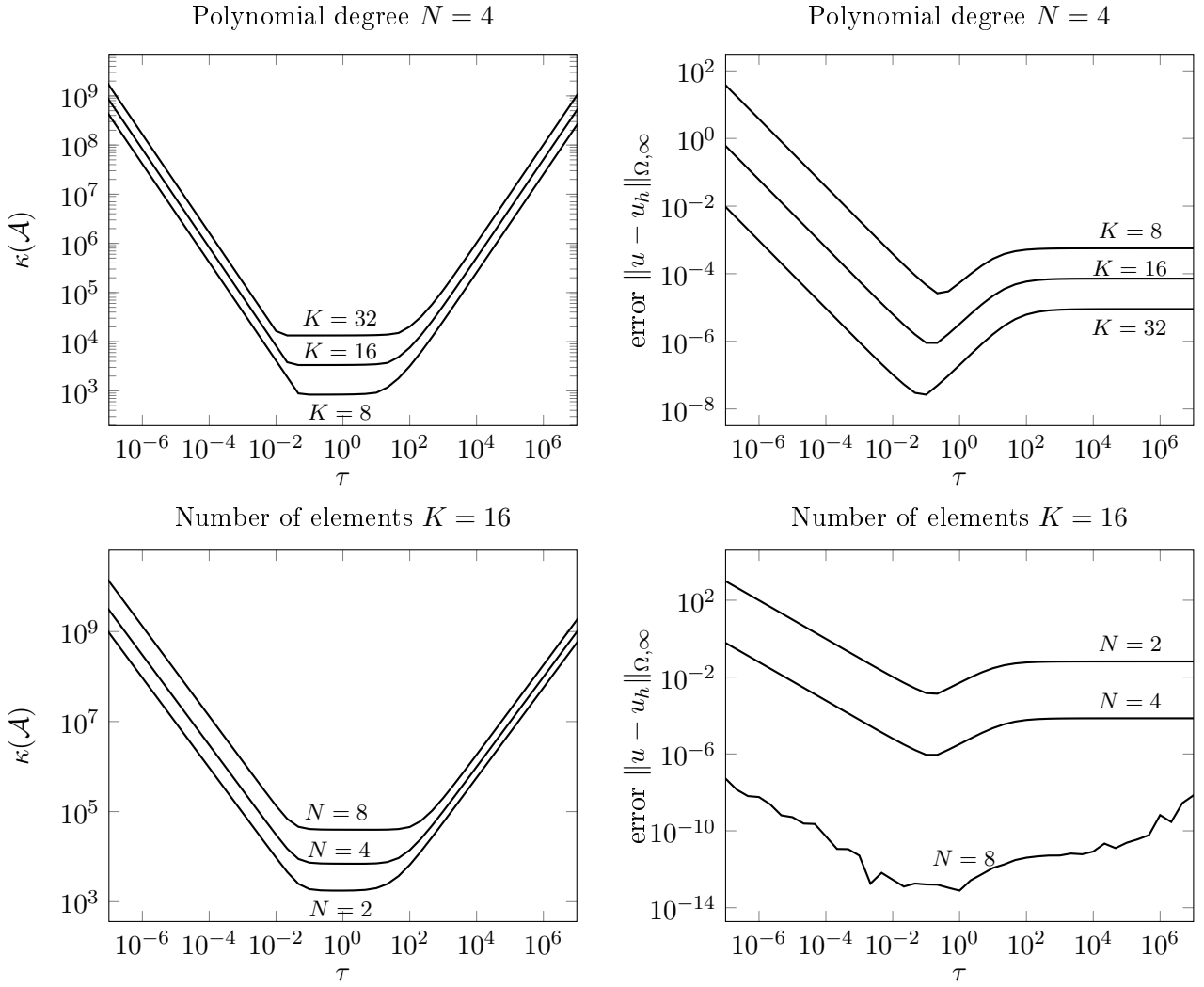
$$\|u - u_h\| = \mathcal{O}(h^{N+1}) \qquad \text{(optimal)}.$$

6.2.2 Stabilized LDG method

Similarly as for the heat equation, a stabilized LDG flux can be formulated for the Poisson equation by the following numerical flux:

$$(q_h)^* = \{\{q_h\}\} + \hat{\boldsymbol{\beta}} \cdot [\![q_h]\!] - \tau [\![u_h]\!],$$
$$(u_h)^* = \{\{u_h\}\} - \hat{\boldsymbol{\beta}} \cdot [\![u_h]\!].$$

Again, we check for the behavior of the conditioning and accuracy for various choices of τ:



The behavior of the condition numbers is very similar to the central flux. However, the following observation can be made:

$$\kappa(\mathcal{A}_{\text{LDG}}) \simeq 2\kappa(\mathcal{A}_{\text{central}}).$$

Thus, iterative solvers whose convergence depends on the condition number of $\mathcal{A}$ will converge somewhat more slowly for the LDG flux than for the central flux.

The convergence with the LDG flux shows a more pronounced difference as compared to the central flux: For the latter, the choice of τ mostly affects the conditioning (with the convergence relatively similar). For the LDG flux, on the other hand, we observe an increasing error as τ becomes very small. Thus, the LDG flux more crucially depends on the stabilization term than the central flux.

Regarding the overall convergence behavior of the LDG flux with $\tau = 1$, we have optimality as for the central flux:

$$\|u - u_h\| = \mathcal{O}(h^{N+1}).$$

From the conditioning and convergence behavior, it appears that the central flux is superior over the LDG flux. However, the LDG flux as an advantage in terms of the coupling between matrix entries. Let us consider this for the scheme with $N = 0$ (piecewise constant ansatz functions). This gives the following two equations (all spatial derivatives vanish for constant shape functions):

$$\begin{aligned} q_h^*(q_h^k, q_h^{k+1}, u_h^k, u_h^{k+1}) - q_h^*(q_h^k, q_h^{k-1}, u_h^k, u_h^{k-1}) &= hf_h^k, \\ u_h^*(u_h^k, u_h^{k+1}) - u_h^*(u_h^k, u_h^{k-1}) &= hq_h^k. \end{aligned}$$

For the central flux, we have

$$q_h^*(q_h^-, q_h^+, u_h^-, u_h^+) = \{\{q_h\}\} - \tau[\![u_h]\!], \quad u_h^*(u_h^-, u_h^+) = \{\{u_h\}\},$$

and thus, the following stencil is recovered

$$\frac{u_h^{k+2} - 2u_h^k + u_h^{k-2}}{(2h)^2} + \tau \frac{u_h^{k+1} - u_h^{k-1}}{h} = f_h^k.$$

For the LDG flux, we instead have:

$$q_h^*(q_h^-, q_h^+, u_h^-, u_h^+) = q_h^- - \tau[\![u_h]\!], \quad u_h^*(u_h^-, u_h^+) = u_h^+,$$

which gives the stencil:

$$\frac{u_h^{k+1} - 2u_h^k + u_h^{k-1}}{h^2} + \tau \frac{u_h^{k+1} - u_h^{k-1}}{h} = f_h^k.$$

This result carries over to the high-order case and also to the case in higher dimensions, where the LDG method gives a considerably sparser method. A sparser representation reduces the cost for a matrix-vector product in iterative solvers and also the number of fill-in entries in factorizations of direct solvers; reducing the cost in both cases.

6.2.3 Symmetric interior penalty (Nitsche) methods

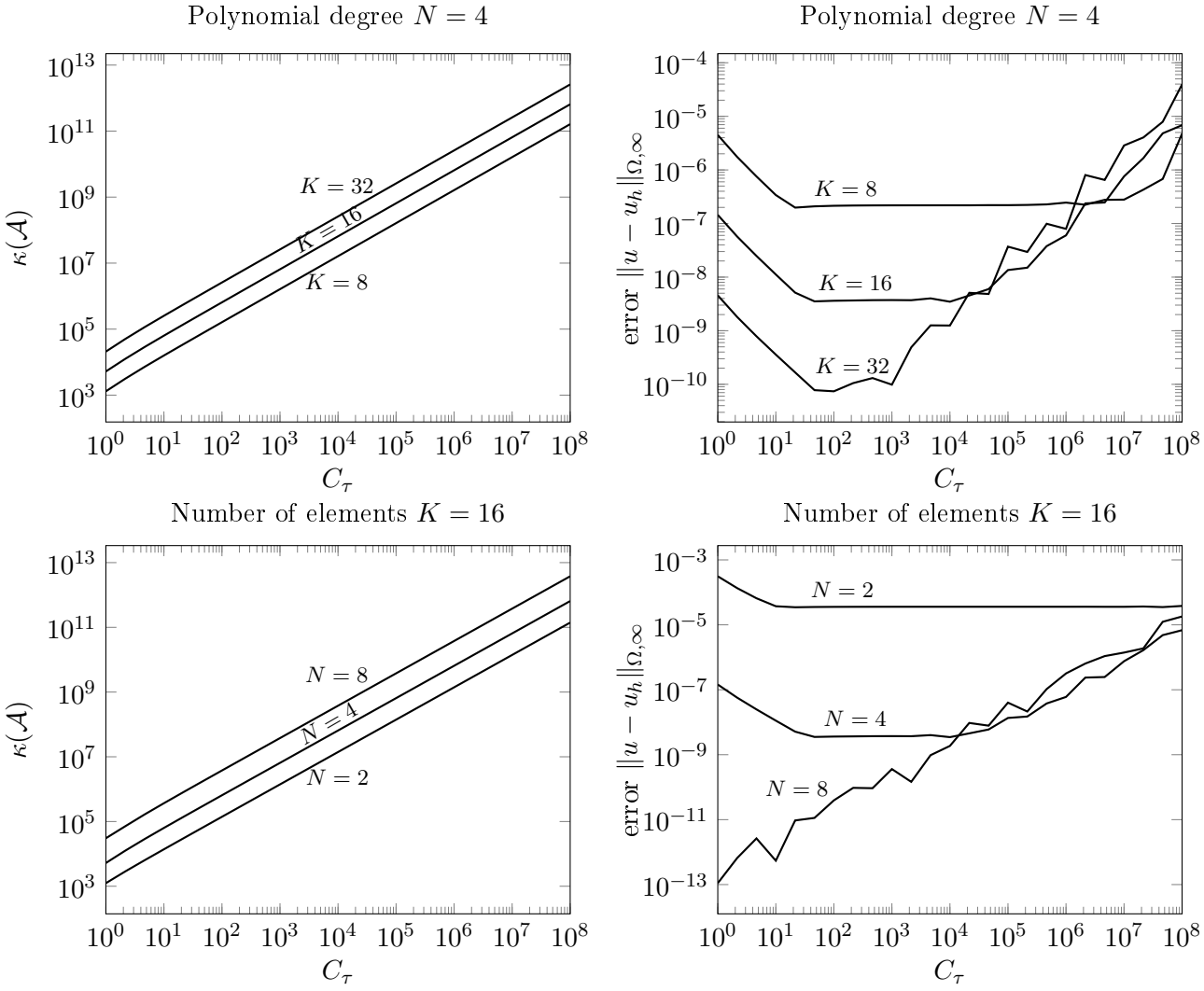
A third alternative for the construction of DG-FEM methods for the second derivative operator are so-called interior penalty methods. In terms of the numerical flux, the following definition is used:

$$\begin{aligned} q_h^* &= \{\{(u_h)_x\}\} - \tau[\![u_h]\!], \\ u_h^* &= \{\{u_h\}\}. \end{aligned}$$

At first sight, this definition looks very similar to the central flux since the auxiliary variable q_h is nothing else than the spatial derivative of u_h. The crucial difference is that we formulate the flux in terms of the derivative on u_x rather than the auxiliary variable.
In terms of the stability of the method, this method is more restrictive than the other two methods. For example, we see that for $N = 0$ the flux q_h^* only depends on the penalty parameter τ. To make the method viable, τ needs to be chosen large enough,

$$\tau \geq C_\tau \frac{(N+1)(N+d)}{d} \frac{1}{h},$$

for d spatial dimensions on triangles and tetrahedra. Besides the constant C_τ, an additional weight with different values for interior and boundary faces, respectively, is used, namely $\frac{1}{2}$ for the interior faces and 1 for boundary faces. We refer to [K. Shahbazi, An explicit expression for the penalty parameter of the interior penalty method, *J. Comput. Phys.* 205:401–407, 2005], for details. For quadrilaterals and hexahedra, the term $\frac{(N+1)(N+d)}{d}$ is replaced by $(N+1)^2$.
Let us have a look at the conditioning and accuracy similarly to the other methods. As opposed to the previous case, we now display the value of the constant C_τ rather than τ in order to make different mesh sizes h comparable:



As for the central method, the condition number $\kappa(\mathcal{A})$ increases as τ with increasing τ (but stability does not allow for $C_\tau < 1$). The accuracy is best for small values of τ, which makes τ close to the stability limit the preferred range. Regarding the relative conditioning compared to the central flux, we have that

$$\kappa(\mathcal{A}_{\text{IP}}) \simeq \kappa(\mathcal{A}_{\text{central}}).$$

The sparsity of the interior penalty method is between the LDG flux and the central flux, which makes the interior penalty method a viable compromise between the two methods.

Finally, let us discuss an alternative derivation of the symmetric interior penalty method. We start with the pointwise differential equation

$$-\nabla^2 u = f \quad \text{in } \Omega, \qquad u = 0 \quad \text{on } \partial\Omega.$$

We assume a subdivision of the domain into elements, multiply by test functions ℓ_i, and enforce the following equation on each element:

$$-\int_{\mathsf{D}^k} \ell_i \nabla^2 u \, d\boldsymbol{x} = \int_{\mathsf{D}^k} \ell_i f \, d\boldsymbol{x}.$$

Integration by parts gives:

$$\int_{\mathsf{D}^k} \nabla \ell_i \cdot \nabla u \, d\boldsymbol{x} - \int_{\partial \mathsf{D}^k} \ell_i \hat{\boldsymbol{n}} \cdot (\nabla u)^* \, d\boldsymbol{x} = \int_{\mathsf{D}^k} \ell_i f \, dx.$$

Taking all elements together and using the central flux $(\nabla u)^* = \{\{\nabla u\}\}$, the following integral form is obtained:

$$(\nabla u, \nabla \ell_i)_{\Omega_h} - \langle \hat{\boldsymbol{n}} \cdot \{\{\nabla u\}\}, \ell_i \rangle_{\Gamma_h} = (f, \ell_i)_{\Omega_h} \qquad \text{for all test functions} \ell_i.$$

As before, the form $\langle \cdot \rangle_{\Gamma_h}$ denotes the collection of integrals over all faces. For boundaries, we assume the function to be extended by symmetry, $\nabla u^+ = \nabla u^-$ and $u^+ = -u^-$. For non-homogeneous Dirichlet conditions or Neumann conditions, suitable modifications are applied.

We can re-write the equation for a more general test function v that contains contributions from both sides of a face as follows:

$$(\nabla u, \nabla v)_{\Omega_h} - \langle \{\{\nabla u\}\}, [\![v]\!] \rangle_{\Gamma_h} = (f, v)_{\Omega_h} .$$

We note that the face integral over $\partial \mathsf{D}^k$ makes the operator non-symmetric, which is undesired for the elliptic equation which is inherently symmetric. Therefore, we add the "transpose" of the face term to the equation:

$$(\nabla u, \nabla v)_{\Omega_h} - \langle \{\{\nabla u\}\}, [\![v]\!] \rangle_{\Gamma_h} - \langle [\![u]\!], \{\{\nabla v\}\} \rangle_{\Gamma_h} = (f, v)_{\Omega_h} .$$

Note that the added term is consistent with the partial differential equation because for the analytic solution we have $[\![u]\!] = 0$. Since this term represents a transposed (adjoint) form of the consistent jump integral, it is also labeled *adjoint consistency term* in the literature. (Mathematically speaking, it makes sure that the discretization fulfills a property called *adjoint consistency*, see [R. Hartmann, Adjoint consistency analysis of dicontinuous Galerkin discretizations, *SIAM J. Numer. Anal.* 45(6):2671–2696, 2007].) This equation is now in the form of the central flux but without the stabilization term. In this form, the bilinear form defined above is still unstable because it is not coercive (i.e., it has zero or negative eigenvalues). To stabilize, we add a penalty term to the equation, which gives the final *symmetric interior penalty* method:

$$(\nabla u, \nabla v)_{\Omega_h} - \langle \{\{\nabla u\}\}, [\![v]\!] \rangle_{\Gamma_h} - \langle [\![u]\!], \{\{\nabla v\}\} \rangle_{\Gamma_h} + \langle [\![u]\!], \tau [\![v]\!] \rangle_{\Gamma_h} = (f, v)_{\Omega_h} .$$

We note that this technique has its origin in a technique for weakly imposing Dirichlet boundary conditions in continuous finite element methods, the Nitsche method.

6.3 Example code: Incompressible Navier–Stokes solver

In our research group, we have recently developed a solver for the incompressible Navier–Stokes equations based on discontinuous Galerkin methods. Details on the work can be found in the preprint [B. Krank, N. Fehn, W. A. Wall, M. Kronbichler, A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow, *arXiv preprint*, `http://arxiv.org/abs/1607.01323`].

The solver solves the incompressible Navier–Stokes equations,

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} - 2\nu\boldsymbol{\epsilon}(\boldsymbol{u})) &= \boldsymbol{f}, \\ \nabla \cdot \boldsymbol{u} &= 0, \end{aligned}$$

where $\boldsymbol{u}$ denotes the fluid velocity, p the fluid pressure, ν the fluid viscosity, and $\boldsymbol{\epsilon}(\boldsymbol{u}) = \frac{1}{2}(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T)$ the strain rate tensor.

Initial conditions are set as $\boldsymbol{u}(\cdot, t = 0) = \boldsymbol{u}_0$. On the boundary, Dirichlet conditions $\boldsymbol{u} = \boldsymbol{g_u}$ on $\partial\Omega^D$ and Neumann conditions $(-p\boldsymbol{I} + 2\nu\boldsymbol{\epsilon}(\boldsymbol{u})) \cdot \boldsymbol{n} = \boldsymbol{h}$ on $\partial\Omega^N$ are set.

Since the particular form of the incompressible Navier–Stokes equations is challenging (saddle point structure, the divergence equation determines the pressure, but no pressure variable appears in this equation), a particular time stepping scheme is used. It uses a **splitting method** based on BDF-3 time integration (see [Karniadakis, Israeli, Orszag, *J. Comput. Phys.* 97(2):414–443, 1991]):

- Explicit convective step: $\gamma_0 \hat{\boldsymbol{u}} = \sum_{i=0}^{2} \left(\alpha_i \boldsymbol{u}^{n-i} - \Delta t \beta_i \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u})^{n-i} \right) + \Delta t \boldsymbol{f}^{n+1}$, where γ_0 and α_i are coefficients of the BDF-3 time integrator and β_i the respective coefficients for third-order extrapolation to the new time level t^{n+1}
- Pressure step: $-\nabla^2 p^{n+1} = -\frac{\gamma_0}{\Delta t} \nabla \cdot \hat{\boldsymbol{u}}$ (including suitable boundary condition for high order)
- Projection step: $\hat{\hat{\boldsymbol{u}}} = \hat{\boldsymbol{u}} - \frac{\Delta t}{\gamma_0} \nabla p^{n+1}$
- Implicit viscous step (Helmholtz-like): $\frac{\gamma_0}{\Delta t} \left(\boldsymbol{u}^{n+1} - \hat{\hat{\boldsymbol{u}}} \right) = \nabla \cdot \boldsymbol{\epsilon}(\boldsymbol{u}^{n+1})$

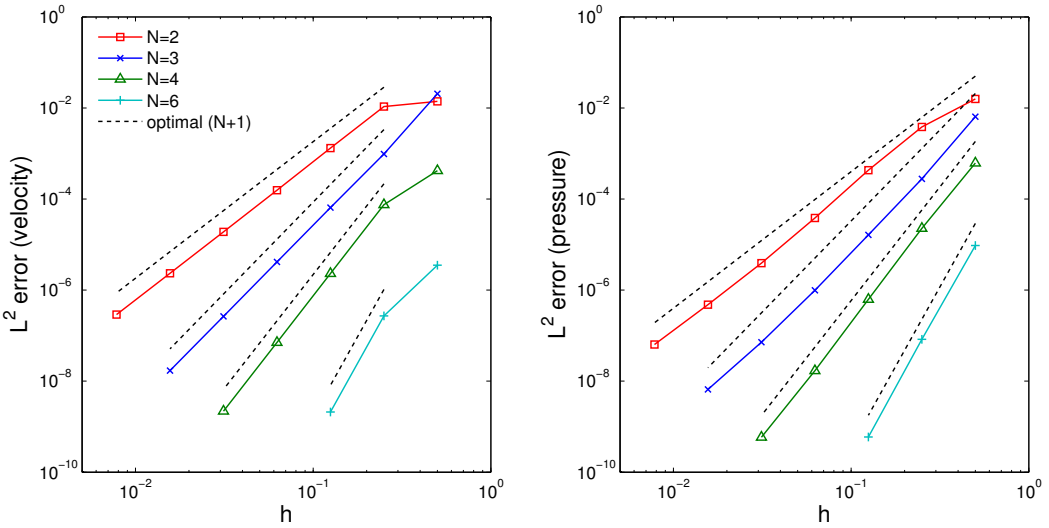
For space discretization, DG methods are used:

- Convective term: Local Lax–Friedrichs flux
  - $\mathcal{F}^{c,*}(\boldsymbol{u}_h) = \{\{\boldsymbol{u} \otimes \boldsymbol{u}\}\} + \frac{\Lambda}{2} [\![\boldsymbol{u}]\!]$, Λ max eigenvalue of flux Jacobian
  - Boundary conditions at t^{n-i} imposed for $\boldsymbol{u}^{n-i}$, $i = 0, \ldots, 2$
- Pressure Poisson equation: Symmetric interior penalty method
  - Geometric multigrid solver for solving the final linear system, takes 5–10 V-cycles for convergence (smoother: polynomial Chebyshev, degree 5)
- Projection step for divergence-free velocity

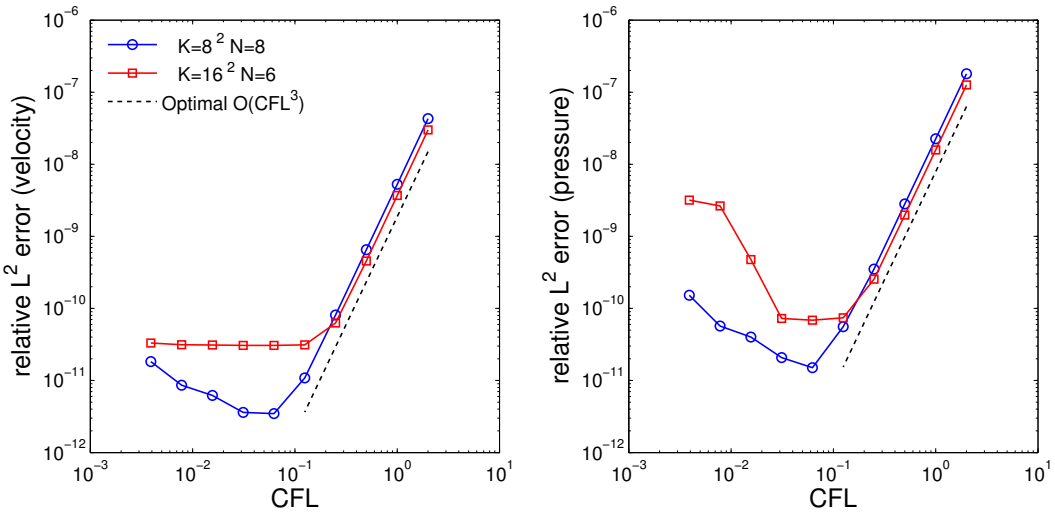
$$(\boldsymbol{v}_h, \hat{\hat{\boldsymbol{u}}}_h)_{\Omega_e} + \underbrace{(\nabla \cdot \boldsymbol{v}_h, \tau_D \nabla \cdot \hat{\hat{\boldsymbol{u}}}_h)_{\Omega_e}}_{\text{div-div penalty}} = (\boldsymbol{v}_h, \hat{\boldsymbol{u}}_h)_{\Omega_e} - \left(\boldsymbol{v}_h, \frac{\Delta t}{\gamma_0} \nabla p_h^{n+1} \right)_{\Omega_e}$$

- Viscous step: Interior penalty discretization of viscous term

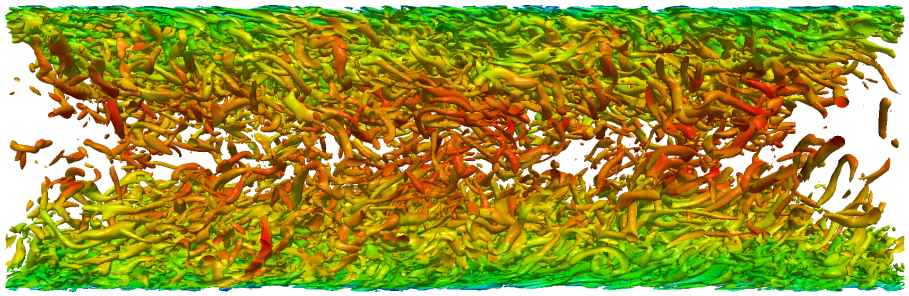
Most operations in the solver involve either computations of right hand sides or matrix-vector products inside iterative solvers, include the multigrid smoother. These are implemented efficiently by the sum factorization techniques presented in Section 4.2.
With these techniques, optimal convergence of rate $\mathcal{O}(h^{N+1})$ is obtained on a vortex test problem against an analytic solution:



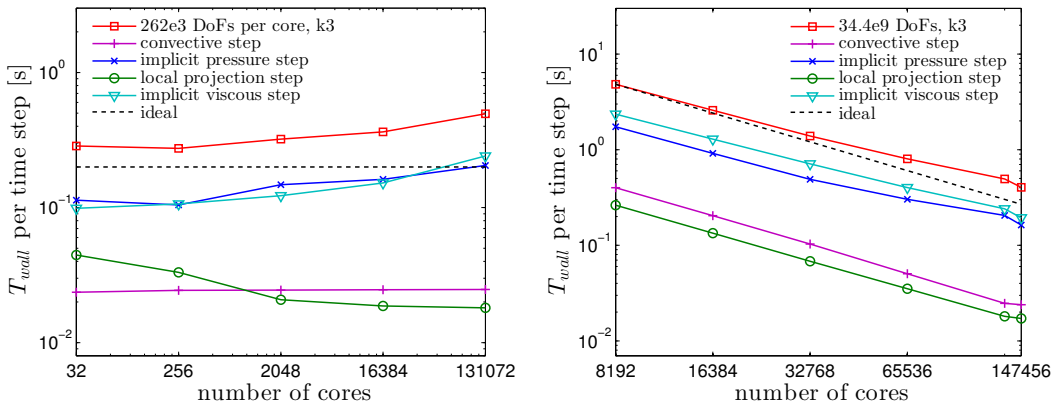
Similarly, optimal convergence of rate $\mathcal{O}(\Delta t^3)$ in time is obtained:



Example application: Simulation of turbulent flow in a periodic channel (x and z directions periodic, y direction bounded by walls, friction Reynolds number $\mathrm{Re} = 590$, corresponding to Reynolds number over diameter of $\sim 30\,000$), visualization of turbulent eddies by Q-criterion:



Turbulent flow requires very high resolution. In order to meet these, large-scale parallel computations are necessary. Due to careful software design, the code is capable of running on today's largest supercomputers. Example numbers from the SuperMUC system (located in Garching, currently 27th fastest computer on `top500.org`, June 2016):



6.4 Hybridizable Discontinuous Galerkin Methods

When comparing discontinuous Galerkin methods with continuous finite elements, we observe that the number of degrees of freedom is higher for the DG methods of the same polynomial degree than in

finite elements. This is because degrees of freedom at faces are doubled in DG. For quadrilateral and hexahedral elements, this is expressed by the simple formula (see Chapter 1)

$$\frac{n_{\text{dofs,DG}}}{n_{\text{dofs,FEM}}} = \frac{(N+1)^d}{N^d}, \qquad \text{e.g. 8 for N=1 in 3D.}$$

In transport problems, the increased stability due to the in-build directionality usually compensates for the higher number of degrees of freedom. In elliptic and parabolic problems (second derivatives, stationary and time-dependent case), however, finite elements are optimal methods in the sense of approximation: Finite elements represent the minimal error in the so-called energy norm of all possible interpolations on the continuous mesh. This translates to the more important L_2 error, where the approximation is often very close to best possible interpolant on the finite element mesh.
Since the polynomial approximation space of DG-FEM methods is similar to the one for continuous finite elements on the same mesh, we can in general not expect superior solution quality. Remember that the jump in the DG solution over faces reduces as the numerical error $\mathcal{O}(h^{N+1})$ in the ideal case. Therefore, the increased number of degrees of freedom and an increased number of matrix entries in DG method typically produces methods with lower efficiency than comparable continuous finite element methods.
Therefore, more efficient discontinous Galerkin methods have been developed. A promising method is the relatively recent **hybridizable discontinous Galerkin (HDG) method**, which was introduced by [B. Cockburn, J. Gopalakrishnan, R. Lazarov, Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems, *SIAM J. Numer. Anal.* 47(2): 1319–1365, 2009] and later applied to the convection-diffusion equation by [N. C. Nguyen, J. Peraire, B. Cockburn, An implicit high-order hybridizable discontinuous Galerkin method for linear convection–diffusion equations, *J. Comput. Phys.* 228: 3232–3254, 2009] and most other second-order differential equations in computational mechanics by the same authors. We have seen a numerical flux for the HDG method already in section 5.1 on the acoustic wave equation.
We introduce the method for the Poisson equation

$$-\nabla^2 u = f(\boldsymbol{x})$$

with boundary conditions:

$$\begin{aligned} u &= g_D, && \text{on the Dirichlet boundary } \Gamma_D, \\ \hat{\boldsymbol{n}} \cdot \nabla u &= g_N, && \text{on the Neumann boundary } \Gamma_N. \end{aligned}$$

As for the other methods, we rewrite the equation as a system of first-order equations in $(u_h, \boldsymbol{q}_h)$, multiply by test functions $(v, \boldsymbol{w})$, integrate over element $\mathcal{D}^k$, and integrate by parts:

$$\begin{aligned} &(\boldsymbol{q}_h, \nabla v)_{\mathsf{D}^k} - \langle (\boldsymbol{q}_h)^* \cdot \hat{\boldsymbol{n}}, v \rangle_{\partial \mathsf{D}^k} = (f, v)_{\mathsf{D}^k}, \\ &(\boldsymbol{q}_h, \boldsymbol{w})_{\mathsf{D}^k} + (u_h, \nabla \cdot \boldsymbol{w})_{\mathsf{D}^k} - \langle (u_h)^*, \boldsymbol{w} \cdot \hat{\boldsymbol{n}} \rangle_{\partial \mathsf{D}^k} = 0. \end{aligned}$$

The first equation represents the equation $-\nabla \cdot \boldsymbol{q}_h = f$, whereas the second equation enforces the equation for the auxiliary variable, $\boldsymbol{q}_h = \nabla u_h$.
In order to define the method, we need to define the numerical fluxes $\boldsymbol{q}_h^*$ and u_h^*. For the HDG method, we express the numerical flux for the gradient variable $\boldsymbol{q}_h^*$ in terms of the numerical flux on u_h:

$$(\boldsymbol{q}_h)^* = \boldsymbol{q}_h - \tau \left(u_h - (u_h)^*\right) \hat{\boldsymbol{n}},$$

where τ is a stabilization parameter. τ can be different on the two sides of an interior face. One typically selects τ to be proportional to the diffusivity, in this example $\tau = 1$. The HDG method defines the numerical flux $(u_h)^*$ by the following equation:

$$(u_h)^* = \frac{\tau^-}{\tau^- + \tau^+} u_h^- + \frac{\tau^+}{\tau^- + \tau^+} u_h^+ - \left(\frac{1}{\tau^- + \tau^+}\right) \left(\boldsymbol{q}_h^- - \boldsymbol{q}_h^+\right) \cdot \hat{\boldsymbol{n}}^-. \tag{6.2}$$

In terms of the definitions of the numerical fluxes expressed in Sec. 6.1, we see that the flux u_h^* depends on q_h, which means that elimination of the contributions associated to $\boldsymbol{q}_h$ from the final matrix system as discussed in Sec. 6.2 is not possible. In other words, it appears that the HDG flux makes the method less efficient because we need to solve a coupled system involving both the nodal values of the scalar variable u_h and the ones for the gradient variable $\boldsymbol{q}_h$, which is $(d+1)$ times as big.

6.4.1 Expressing the flux in terms of a new variable

At this point, one takes a counter-intuitive action in the HDG method for reducing the size of linear systems. Instead of using the formula for u_h^* above, one sets it to the new variable

$$(u_h)^* = \begin{cases} g_D, & \text{on faces on Dirichlet boundary, } \Gamma_h \cap \Gamma_D, \\ \lambda_h, & \text{on all other faces, } \Gamma_h \setminus \Gamma_D. \end{cases}$$

The new variable λ_h is defined on the faces Γ_h of the computational mesh $\Omega_h = \cup_k \mathsf{D}^k$. The set of all faces is the mesh skeleton and usually called the **trace**.

- The term **hybridized method** refers to a method where a numerical trace (corresponding to the numerical flux in this case) is introduced as an additional independent variable.

Usual Lagrange polynomials are used on the faces,

$$\lambda_h \in \{\mu \in L_2(\Gamma_h) : \mu|_f \text{ polynomial up to degree } N \text{ on face } f\}.$$

In 1D, the trace λ_h simply denotes point values in the vertices of the domain. For K elements, there are $K+1$ point values of λ_h, irrespective of the polynomial degree used inside the elements.
Having introduced a new variable λ_h, we need to also define an additional equation for closing the system. The equation is related to the definition of the numerical flux $\boldsymbol{q}_h^*$. On element D^k, the additional equation in the HDG method reads:

$$\langle \boldsymbol{q}_h \cdot \hat{\boldsymbol{n}} - \tau(u_h - \lambda_h), \mu \rangle_{\partial \mathsf{D}^k} = \langle g_N, \mu \rangle_{\partial \mathsf{D}^k \cap \Gamma_N}.$$

Note that this equation includes the Neumann boundary condition on faces belonging the Neumann boundary on the right hand side. This equation for all elements in the mesh is added up over the elements, similar to the equations in the interior.
Let us look at the meaning of this equation. Consider an internal face $f \in \Gamma_h \setminus (\Gamma_D \cup \Gamma_N)$. Since f is in the interior of the domain, the boundary term $\langle g_N, \mu \rangle_{\partial \mathsf{D}^k \cap \Gamma_N}$ is not present and we only need to add two contributions from the $()^-$ and $()^+$ sides of f:

$$\langle \boldsymbol{q}_h \cdot \hat{\boldsymbol{n}} - \tau(u_h - \lambda_h), \mu \rangle_{f^-} + \langle \boldsymbol{q}_h \cdot \hat{\boldsymbol{n}} - \tau(u_h - \lambda_h), \mu \rangle_{f^+} = 0,$$

which can be combined into the following equation (assuming $\lambda^- = \lambda^+ = \lambda$):

$$\left\langle (\boldsymbol{q}_h^- - \boldsymbol{q}_h^+) \cdot \hat{\boldsymbol{n}}^- - \tau(u_h^- + u_h^+) + 2\tau\lambda_h, \mu \right\rangle_f = 0.$$

This statement is equivalent to enforcing continuity on the flux, $[\![(\boldsymbol{q}_h)^*]\!] = 0$, namely:

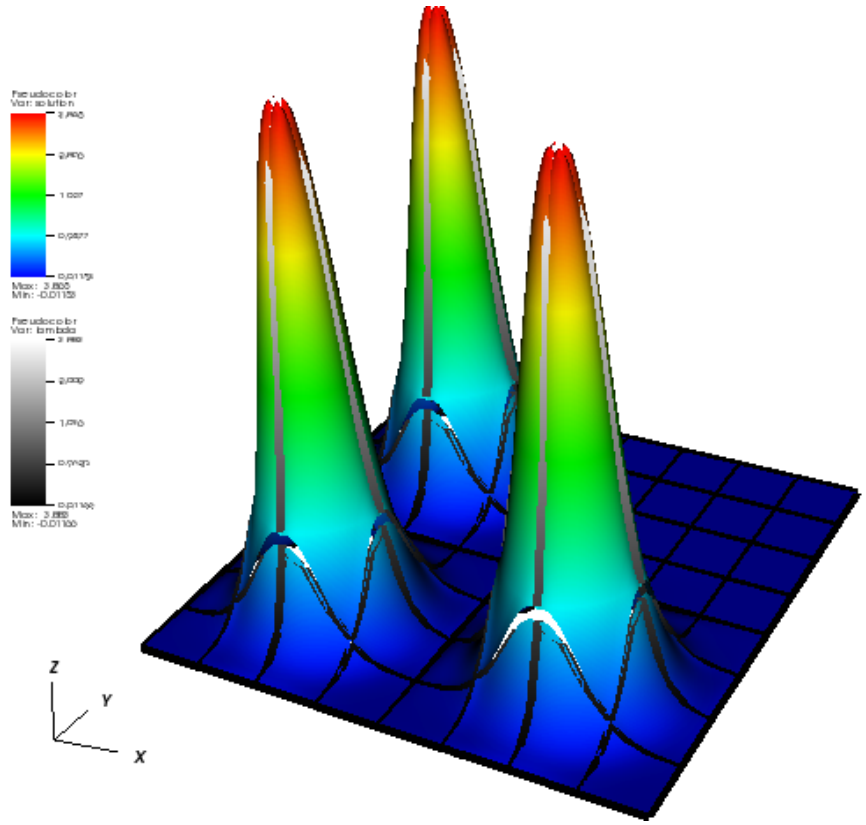
$$\langle [\![\boldsymbol{q}_h - \hat{\boldsymbol{n}}\tau(u_h - \lambda_h)]\!], \mu \rangle_f = 0.$$

Solved for λ_h on the combination of the two sides of a face, this is nothing else than equation (6.2) for $\lambda^- = \lambda^+ = \lambda$.
Regarding boundary conditions, the HDG method applies two strategies:

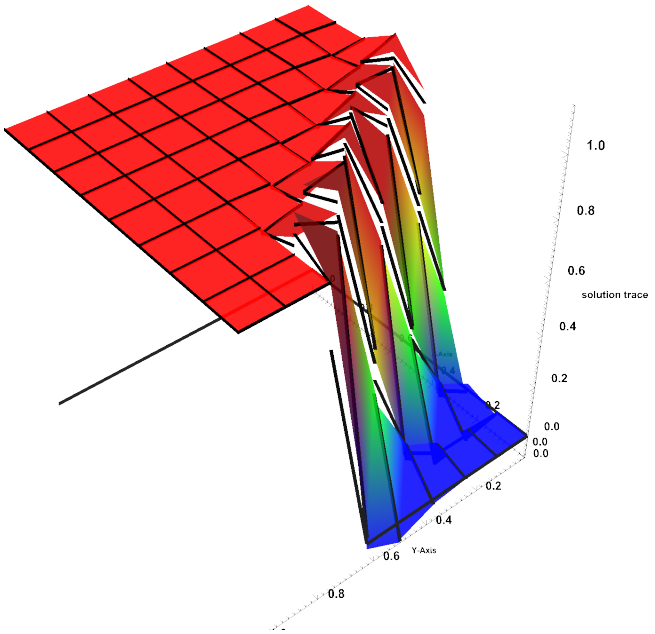
Dirichlet boundary conditions: Strongly imposed on the variable λ_h by setting $\lambda_h = g_D$.

Neumann boundary conditions: Weakly imposed on the equation for the flux by boundary integral $\langle g_N, \mu \rangle_{\partial \mathsf{D}^k \cap \Gamma_N}$.

Due to this construction, the space for λ_h can be seen as being between a completely discontinuous ansatz spaces used in DG-FEM where also Dirichlet boundary conditions are enforced by integrals, and the continuous finite elements which impose boundary conditions in a similar way but use a globally continuous solution. The role of λ_h can be seen as an average value between u_h^- and u_h^+ that additionally takes the jump in the normal value $[\![\boldsymbol{q}_h]\!]$ into account. If τ is very large, λ is almost exclusively determined by the average $(u_h^- + u_h^+)/2$. Note that the polynomials used for the HDG method are polynomials within faces but independent from one face to another, i.e., they are discontinuous over vertices in 2D and vertices and edges in 3D.
Let us visualize λ_h on a typical example with quadrilateral elements where the polynomial $N = 3$ on an 8×8 mesh:



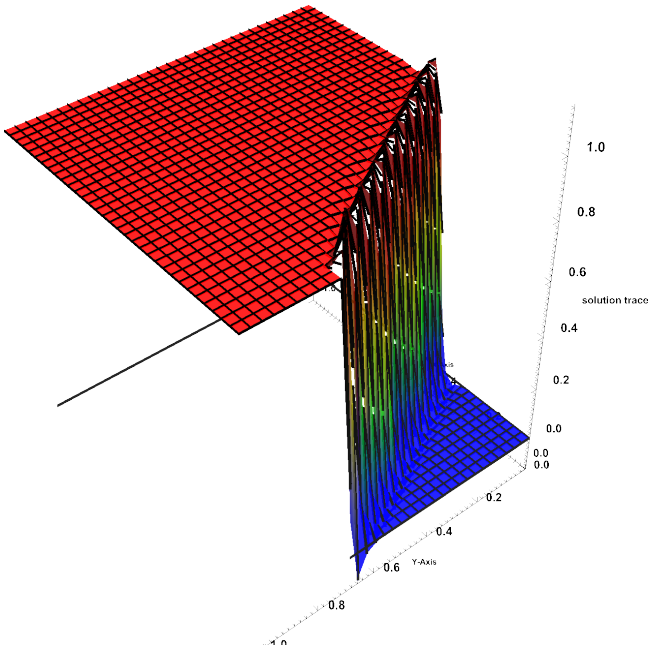
The figures hows the solution u_h (surfaces) together with the numerical trace λ_h (thick lines at element boundaries). It can be seen that the trace follows the solution on the two sides of the face. A second example is a stationary convection–diffusion problem with dominating convection (diffusivity $\kappa = 10^{-6}$), now for $N = 1$ on an 8×8 mesh:



In this example, we have a transport direction going from the lower left to the upper right,

$$\boldsymbol{a} = (\cos(\pi/3), -\sin(\pi/3))^T = \left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)^T.$$

The origin of the domain is the upper right boundary where axes are printed. As boundary conditions, zero Dirichlet conditions are set on the edges with $\{x = 1\}$ and $\{y = 0\}$, whereas $g_D = 1$ is set on the edge $\{x = 1\}$. On the remaining edge at $\{x = 0\}$ a zero value is set for $y \leq 0.7$ and one for $y > 0.7$. We can see that the zero Dirichlet boundary conditions in the two "upper" boundaries have the trace (black lines) set to zero, but the inner solution u_h displayed by the colored surfaces is completely agnostic of this value. This is because the diffusivity is so small that only the transport appears, for which only the interior transport of value 1 matters. This shows one of the strengths of DG methods: A continuous method with strong imposition of boundary conditions would need to resolve a boundary layer of width similar to 10^{-6} (or use stabilization), but the DG method simply not resolves the boundary layer. In this example, there is a second interior boundary layer which of course propagates through the domain. There are some over- and undershoots because no measures for discontinuity reduction (limiting) have been taken. Finally, the solution of the same problem on a 32×32 mesh looks as follows:



6.4.2 Efficient implementation

Taking all equations together, the HDG method results in the following discrete system of equations:

$$\begin{bmatrix} \mathcal{A} & -\mathcal{B} & \mathcal{C} \\ \mathcal{B}^T & \mathcal{D} & \mathcal{G} \\ \mathcal{C}^T & \mathcal{G}^T & \mathcal{H} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_h \\ \boldsymbol{q}_h \\ \boldsymbol{\lambda}_h \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \\ \boldsymbol{h} \end{bmatrix},$$

where the matrices are given by the following weak forms (over all indices in $u_h, \boldsymbol{q}_h, \lambda_h$ for columns and $v, \boldsymbol{w}, \mu$ for rows):

- $\mathcal{A} = \langle \tau u_h, v \rangle_{\Gamma_h}$
- $\mathcal{B} = (\nabla \cdot \boldsymbol{q}_h, v)_{\Omega_h}$
- $\mathcal{C} = -\langle \tau \lambda_h, v \rangle_{\Gamma_h}$
- $\mathcal{D} = (\boldsymbol{q}_h, \boldsymbol{w})_{\Omega_h}$
- $\mathcal{G} = -\langle \lambda_h, \boldsymbol{n} \cdot \boldsymbol{w} \rangle_{\Gamma_h}$
- $\mathcal{H} = \langle \tau \lambda_h, \mu \rangle_{\Gamma_h}$
- $\boldsymbol{f} = (f, v)_{\Omega_h}$
- $\boldsymbol{h} = \langle g_N, \mu \rangle_{\Gamma_N}$

We observe that the matrix

$$\begin{bmatrix} \mathcal{A} & -\mathcal{B} \\ \mathcal{B}^T & \mathcal{D} \end{bmatrix}$$

is block-diagonal over elements because only terms inside element D^k contribute and no coupling to other elements appears.
The coupling is only applied over the numerical trace λ_h.

Therefore, the matrix block can be condensed away and only a matrix system for the trace λ_h needs to be considered:

$$\mathcal{K}\boldsymbol{\lambda}_h = \boldsymbol{r}_h,$$

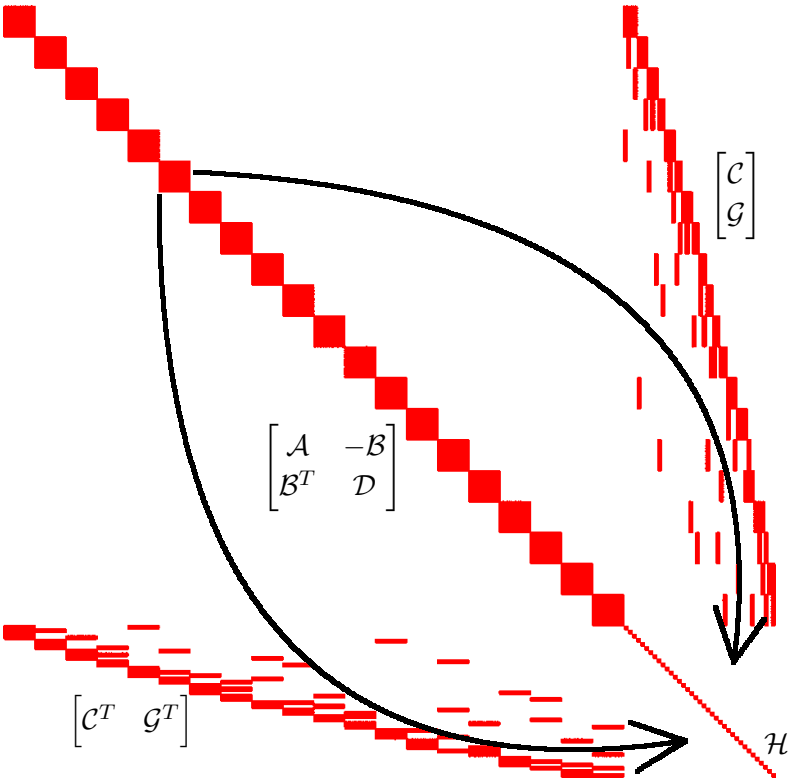
where the matrix is given by

$$\mathcal{K} = \mathcal{H} - \begin{bmatrix} \mathcal{C}^T & \mathcal{G}^T \end{bmatrix} \begin{bmatrix} \mathcal{A} & -\mathcal{B} \\ \mathcal{B}^T & \mathcal{D} \end{bmatrix}^{-1} \begin{bmatrix} \mathcal{C} \\ \mathcal{G} \end{bmatrix},$$

and the right hand side is

$$\boldsymbol{r} = \boldsymbol{h} - \begin{bmatrix} \mathcal{C}^T & \mathcal{G}^T \end{bmatrix} \begin{bmatrix} \mathcal{A} & -\mathcal{B} \\ \mathcal{B}^T & \mathcal{D} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix}.$$

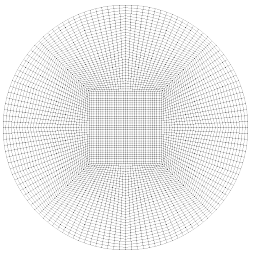
The process of removing the element-by-element contributions in favor of a matrix on the trace space λ_h is called **static condensation** and the matrix substitutions involved in this process are called **Schur complements**. This operation can be applied to any linear system (not necessarily related to DG) but it only makes sense computationally when the matrix to be inverted is sparse. This is the case for the HDG element variables $u_h, \boldsymbol{q}_h$ because the associated matrix is block-diagonal.
Let us visualize the process of computing the reduced system in terms of the matrix sparsities for a sample problem:



This results in a condensed matrix $\mathcal{K}$ of the following structure:

The result of the static condensation is a considerably smaller linear system. It is smaller than for other discontinuous methods and comes close to the size of linear systems in continuous finite elements because only degrees of freedom on a $(d-1)$-dimensional object, the mesh skeleton, are involved. For a two-dimensional mesh of quadrilaterals, we compare both the number of degrees of freedom in the primal variables and the number of nonzero entries in the final matrix system with continuous finite elements (FEM) and a symmetric interior penalty DG (DG SIP) formulation as a function of the polynomial degree N:

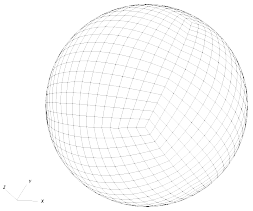
5120 elements
10304 faces



	matrix size (dofs)			matrix nonzeros		
	FEM	DG SIP	HDG	FEM	DG SIP	HDG
$N=1$	5 185	20 480	20 608	0.021m	0.41m	0.18m
$N=2$	20 609	46 080	30 912	0.32m	2.06m	0.64m
$N=3$	46 273	81 920	41 216	1.1m	6.52m	1.1m
$N=4$	82 177	128 000	51 520	2.9m	15.9m	1.8m
$N=5$	128 321	184 320	61 824	6.2m	33.0m	2.6m

In 3D, the volume-to-surface ratio is less beneficial than in 2D, which makes HDG comparably more expensive with comparison the continuous finite elements. Nonetheless, the savings are quite significant vis-à-vis other discontinuous Galerkin methods:

28672 elements
86784 faces



	matrix size (dofs)			matrix nonzeros		
	FEM	DG SIP	HDG	FEM	DG SIP	HDG
$N=1$	29 521	229 376	260 352	0.74m	12.7m	6.1m
$N=2$	232 609	774 144	520 704	14m	145m	34m
$N=3$	781 297	1 835 008	867 840	94m	816m	93m
$N=4$	1 847 617	3 584 000	1 301 760	390m	3110m	210m
$N=5$	3 603 601	6 193 152	1 822 464	1200m	9290m	410m

6.4.3 Superconvergent post-processing

The HDG method does not only reduce the number of unknowns in the system of equations, but also provides optimal convergence rates. Both the solution u_h and the gradient $\boldsymbol{q}_h$ converge at optimal order, i.e., the convergence rates are $\mathcal{O}(h^{N+1})$ for both variables. This is in contrast to the other DG-FEM methods that we discussed before where convergence rates are either sub-optimal for certain boundary conditions and polynomial degrees (LDG methods) or sub-optimal for the gradient variable $\boldsymbol{q}_h$ (interior penalty method) or both.
Since the gradient $\boldsymbol{q}_h = \nabla u_h$ converges at order $N+1$, one could hope for convergence of order $\mathcal{O}(h^{N+2})$ in some reconstructed solution variable. The HDG literature gives a positive answer to this question. The corresponding algorithm reads

Find $\tilde{u}_h$ in $\{$polynomials up to degree $N+1$ on $\mathsf{D}^k\}$ such that for all test functions $\tilde{v} \in \{$polynomials up to degree $N+1$ on $\mathsf{D}^k\}$ it holds:

$$(\nabla \tilde{u}_h, \nabla \tilde{v})_{\mathsf{D}^k} = (\boldsymbol{q}_h, \nabla \tilde{v})_{\mathsf{D}^k},$$
$$(\tilde{u}_h, 1)_{\mathsf{D}^k} = (u_h, 1)_{\mathsf{D}^k}.$$

The first equation sets the gradient of $\tilde{u}_h$ equal to $\boldsymbol{q}_h$, whereas the second equation forces the average of $\tilde{u}_h$ which is not captured by the second equation, to be the same as the average of u_h. This procedure

corresponds to finding $\tilde{u}_h$ along the following minimization problem:

$$\min_{\tilde{u}_h} \|\nabla \tilde{u}_h - \boldsymbol{q}_h\|^2,$$

under the constraint that

$$\int_{\mathsf{D}^k} \tilde{u}_h d\boldsymbol{x} = \int_{\mathsf{D}^k} u_h d\boldsymbol{x}.$$

Since $\boldsymbol{q}_h$ converges at rate $N+1$ and the average of u_h at rate $N+1$, the resulting variable $\tilde{u}_h$ is a polynomial of degree $N+1$ and converges at rate $\mathcal{O}(h^{N+2})$. This higher rate of convergence is called **superconvergence**.

While superconvergence results are also known for finite elements in certain contexts (structured meshes, constant coefficients, as used e.g. for the superconvergent patch recovery [O. C. Zienkiewicz, J. Z. Zhu, The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique, *Int. J. Numer. Meth. Eng.* 33(7):1331–1364, 1992]), the superconvergence for HDG translate to unstructured meshes and non-constant coefficients. Note that the formulas for the wave equation in Section 5.1 are also amenable to superconvergent post-processing.

6.5 Check yourself

- What is the purpose of transforming the heat equation

  $$\frac{\partial u}{\partial t} - \nabla^2 u = f$$

  into a first-order system

  $$\frac{\partial u}{\partial t} - \nabla \cdot \boldsymbol{q} = f,$$
  $$\boldsymbol{q} = \nabla u?$$

- Explain the central flux for the heat equation in 1D. State the matrix system in terms of the nodal values $\boldsymbol{u}_h$ and $\boldsymbol{q}_h$ and explain the coupling between the individual components. Given the particular form of the matrix system, explain how the variable $\boldsymbol{q}_h$ is treated in the numerical algorithm.

- Explain the flux of the local discontinuous Galerkin method and compare it to the central flux.

- Discuss the convergence rates of the (unstabilized) central and LDG fluxes for the heat equation, respectively.

- Explain the additional stabilization (penalty) term in the central and LDG flux necessary for elliptic problems (i.e., problems without time derivatives).

- How does the condition number of the system matrix depend on the stabilization parameter τ?

- Why does one prefer settings of τ where the condition number is moderate?

- Compare the matrix properties of the LDG flux and the central flux.

- How is the interior penalty method derived for the Poisson equation?

- Discuss the size of the penalty parameter τ for the interior penalty method and compare to the central and LDG fluxes, respectively.

Hybridizable discontinuous Galerkin method

- How is the numerical flux in the hybridizable discontinuous Galerkin method realized?
- What is the role of the additional trace variable λ_h?
- What makes the HDG method a very efficient alternative as compared to other DG-FEM aproaches for equations with second derivatives?
- What is the expected convergence rate for the HDG method in u_h and $\boldsymbol{q}_h$. What is super-convergent post-processing?