# Efficient evaluation of weak forms in discontinuous Galerkin methods

**Background:** This material gives an introduction to sum factorization as an efficient evaluation technique for the integrals in discontinuous Galerkin methods on quadrilateral elements. It is intended for two lectures of 90 minutes given to advanced students (Master or PhD level) with good knowledge on finite element methods.
The SPPEXA project ExaDG, financed by DFG in 2016–2018, brings these methods to a generic finite element context through the deal.II finite element library, `www.dealii.org`. Example C++ implementations of these algorithms can be found in the deal.II tutorials, available as open source. This text concentrates on the mathematical aspects of fast evaluation of differential operators that builds the basis for large-scale simulation codes in the usual domain decomposition framework, be it in explicit time integration or implicit linear solvers. The techniques in this text have their roots in spectral element methods established in the 1980s. Textbooks with more extensive discussion of these methods are:

- Karniadakis, G.E., Sherwin, S.J.: Spectral/hp element methods for computational fluid dynamics, 2nd edn. Oxford University Press (2005)
- Kopriva, D.: Implementing spectral methods for partial differential equations. Springer (2009)

## 1 DG-FEM discretization

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

$$\begin{aligned} &\frac{\partial u(\boldsymbol{x},t)}{\partial t} + \nabla \cdot \boldsymbol{f}(u(\boldsymbol{x},t),\boldsymbol{x},t) = 0, \quad && \boldsymbol{x} \in \Omega \subset \mathbb{R}^d, \\ &u(\boldsymbol{x},t) = g(\boldsymbol{x},t), && \boldsymbol{x} \in \partial\Omega_i, \\ &u(\boldsymbol{x},0) = u_0(\boldsymbol{x}), \end{aligned} \tag{1}$$

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 $\Omega$, with $\mathsf{D}^k$ polygonal elements. We assume a mapping from the reference element to the real shape by a polynomial transformation, which can include curved boundaries.
We assume the local solution $u_h^k$ to be a weighted sum of polynomials on the element $\mathsf{D}^k$,

$$u_h^k(\boldsymbol{x},t) = \sum_{i=1}^{N_p} u_h^k(\boldsymbol{x}_i^k,t)\ell_i^k(\boldsymbol{x}), \tag{2}$$

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 derived by multiplication of Equation (1) by the test functions $\ell_j^k$, $j = 1, \ldots, N_p$ and integration by parts of the spatial derivative:

$$\int_{\mathsf{D}^k} \left[ \frac{\partial u_h^k}{\partial t}\ell_j^k(\boldsymbol{x}) - \boldsymbol{f}_h^k \cdot \nabla \ell_j^k(\boldsymbol{x}) \right] d\boldsymbol{x} + \int_{\partial \mathsf{D}^k} \hat{\boldsymbol{n}} \cdot \boldsymbol{f}^* \ell_j^k(\boldsymbol{x}) d\boldsymbol{x} = 0. \tag{3}$$

The numerical flux $\boldsymbol{f}^*$ can be computed in various ways. In this lecture, we consider 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 $\boldsymbol{f} = (f_1, f_2)$.
Similar to continuous finite elements, two sets of element shapes are in wide use for discontinuous Galerkin methods:

**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). Finally, triangular elements with affine mappings allow to factor out the geometric factors. This way, only reference element matrices and a single Jacobian factor per element are necessary for representing the integrals in (3), significantly increase arithmetic intensity and thus performance on modern computers.

**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.

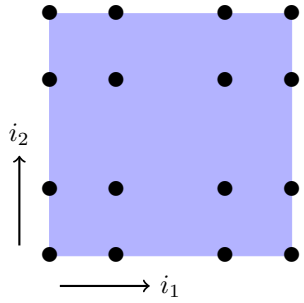
This text concentrates on the case of quadrilateral elements in 2D and hexahedral elements in 3D and presents fast matrix-free evaluation schemes.

## 2 Basic DG scheme

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 $\mathsf{D}^k$ 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. This gives well-conditioned interpolation also when going to high orders, as opposed to equidistant point distributions.
We insert these polynomial functions into the representation (2) and the integrals for DG-FEM. In matrix notation, the equation reads

$$\mathcal{M}\frac{\partial \boldsymbol{u}}{\partial t} - \mathcal{S}^T f(\boldsymbol{u}) + \mathcal{F} f(\boldsymbol{u}) = \boldsymbol{0}, \tag{4}$$

where $\boldsymbol{u}$ describes the collection of all nodal unknowns on all elements, $\mathcal{M}$ is the mass matrix, $-\mathcal{S}^T$ the convection matrix and $\mathcal{F}$ the flux matrix the represents the coupling between elements through the Lax–Friedrichs flux. In this text, we target explicit time integration. Since the mass matrix does not couple between elements, we can write the equation in the form:

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathcal{M}^{-1}\left(\mathcal{S}^T - \mathcal{F}\right) f(\boldsymbol{u}). \tag{5}$$

Instead of building all three matrices, the algorithm in the following only makes use that the inverse mass matrix $\mathcal{M}^{-1}$ is explicitly computed whereas the integrals underlying $\mathcal{S}$ and $\mathcal{F}$ are evaluated on the fly as we insert the known nodal values $\boldsymbol{u}^n$ in an explicit time stepping scheme. This form of evaluation is called matrix-free evaluation.
Note that the mass matrix $\mathcal{M}$ is a dense matrix within one element for the Gauss–Lobatto node points. For other basis functions (e.g. orthogonal basis functions), the mass matrix becomes diagonal and thus even cheaper to invert. The techniques described below can also be used to invert the mass matrix $\mathcal{M}$ or apply the inverse $\mathcal{M}^{-1}$ more quickly than a naive approach, by transforming to a basis with diagonal mass matrix.

### 2.1 Computation of element mass matrix

For the mass matrix, we compute the following integral:

$$\mathcal{M}^k_{ij} = \int_{\mathsf{D}^k} \ell^k_i(\boldsymbol{x})\ell^k_j(\boldsymbol{x})\, d\boldsymbol{x}.$$

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 $2\times 2$ matrix $\boldsymbol{J}^k$ is given by $\boldsymbol{J}^k = \frac{d\boldsymbol{F}^k}{d\boldsymbol{r}} = \begin{bmatrix} \frac{\partial F^k_1}{\partial r} & \frac{\partial F^k_1}{\partial s} \\ \frac{\partial F^k_2}{\partial r} & \frac{\partial F^k_2}{\partial s} \end{bmatrix}$, where $\boldsymbol{F}^k = (F^k_1, F^k_2)^T$ 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:

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

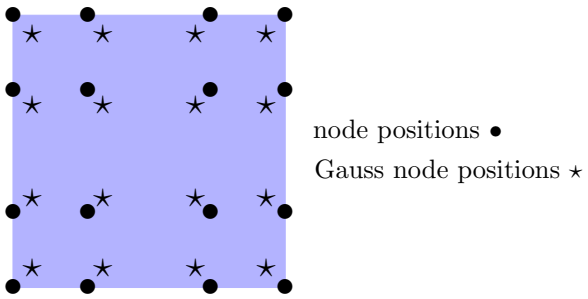
The integral is approximated by quadrature. Here, we consider the case of Gaussian quadrature based on the Gauss–Legendre quadrature nodes $\xi_a$ and Gauss weights $w_a$. From a one-dimensional Gauss rule, a multi-dimensional rule is constructed by a tensor product, i.e., we select

- 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 quadrature 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 tensor degree is at most 1: The first column in $\boldsymbol{J}^k$ is constant in $r$ and linear in $s$ and the second is constant $s$ and linear in $r$, such that the product in the determinant contains at most a product of a linear function in $r$ and a linear function in $s$. In 3D, the tensor degree is at most 2. If a polynomial mapping of degree $l$ is used, it can contain polynomials up to degree $2l-1$ (or $3l-1$ 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 higher order than the $N+1$ accuracy of the polynomial interpolation. (The actual situation is more involved, see finite element textbooks on variational crimes. 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 as compared to geometry aliasing.)
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 ⋆. 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.

## 2.2 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 $\ell_j^k$ 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 $\boldsymbol{r}_1, \boldsymbol{r}_2, \dots$. 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 $\ell_j$ 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,

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

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

### 2.2.1 Evaluation of face integrals

For the face computations, we need to evaluate the integrals over products of $\ell^k_j$ and the DG-FEM solution $u^k_h$ from both sides. 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 $u^{k^-}_h$ and $u^{k^+}_h$ on the face quadrature points by multiplication with appropriate matrices $\mathcal{N}^T_{e^-}$ and $\mathcal{N}^T_{e^+}$ 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. (3). 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. (3).

## 3 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 |

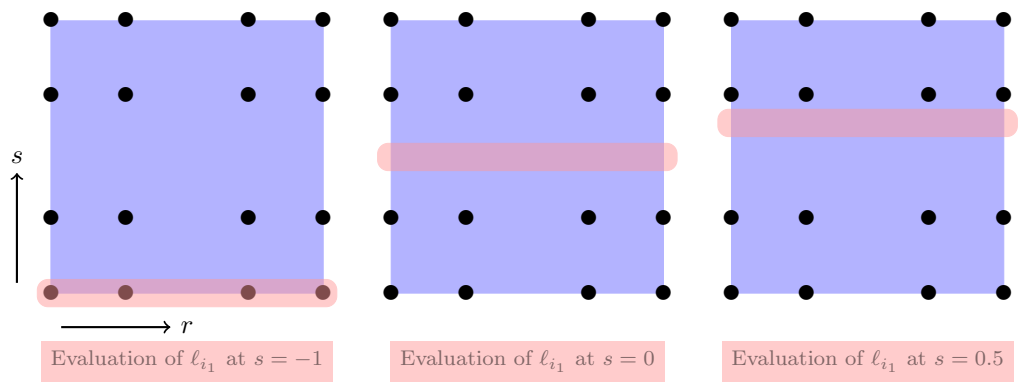
In order to make the method competitive, we need to improve on the evaluation of the shape functions on quadrature points, in particular for 3D.
The idea to make this operation faster is so-called tensorial evaluation. The idea is to exploit that both the basis functions $\ell_i(\boldsymbol{r}_q) = \ell_{i_1}(r_{q_1})\ell_{i_2}(s_{q_2})$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

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

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 $\ell_i^{\text{1D}}(r_{q_1})$ is always the same.



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

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

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

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

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,

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

In the above example for $\mathcal{Q}_3$ basis functions (cubic polynomials on a tensor grid), the 1D matrices are $4 \times 4$. The tensor product $\mathcal{N}^{1\mathrm{D}} \otimes \mathcal{N}^{1\mathrm{D}}$ is a $16 \times 16$ 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,

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

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

### 3.1 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

$$\mathcal{B} \otimes \mathcal{A} = \begin{bmatrix} b_{11}\mathcal{A} & \cdots & b_{1q}\mathcal{A} \\ \vdots & \ddots & \vdots \\ b_{p1}\mathcal{A} & \cdots & b_{pq}\mathcal{A} \end{bmatrix} \tag{6}$$

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 (6), 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 $\boldsymbol{y}^{(2)} = [y_{n+1}, y_{n+2}, \ldots y_{2n}]^T$ 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

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

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 (6), 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

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

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

$$\mathcal{Z} = \mathcal{A}\mathcal{Y}\mathcal{B}^T, \tag{7}$$

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 $\boldsymbol{z}$.

### 3.2 Benefits of the tensorized matrix-vector product

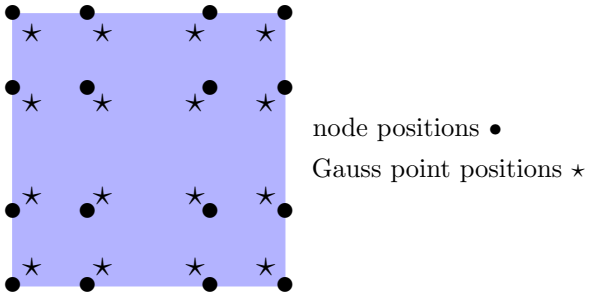
Let us look at the computational cost for the computation of $\mathcal{A}\mathcal{Y}\mathcal{B}^T$ instead of $(\mathcal{B}\otimes\mathcal{A})\boldsymbol{y}$. For simplicity, assume that the matrix dimensions are all the same, $m = n = p = q$. The original matrix-vector costs $m^4$ multiplications and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

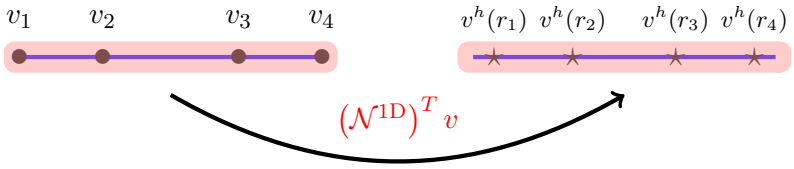
This algorithm reduces the cost from $2m^4$ arithmetic operations to $4m^3$ operations. If $m$ is large (i.e., higher polynomial degree), this results in a considerable saving. This evaluation concept is also called *sum factorization* because it takes out common factors of the summation in the matrix-vector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

### 3.3 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.



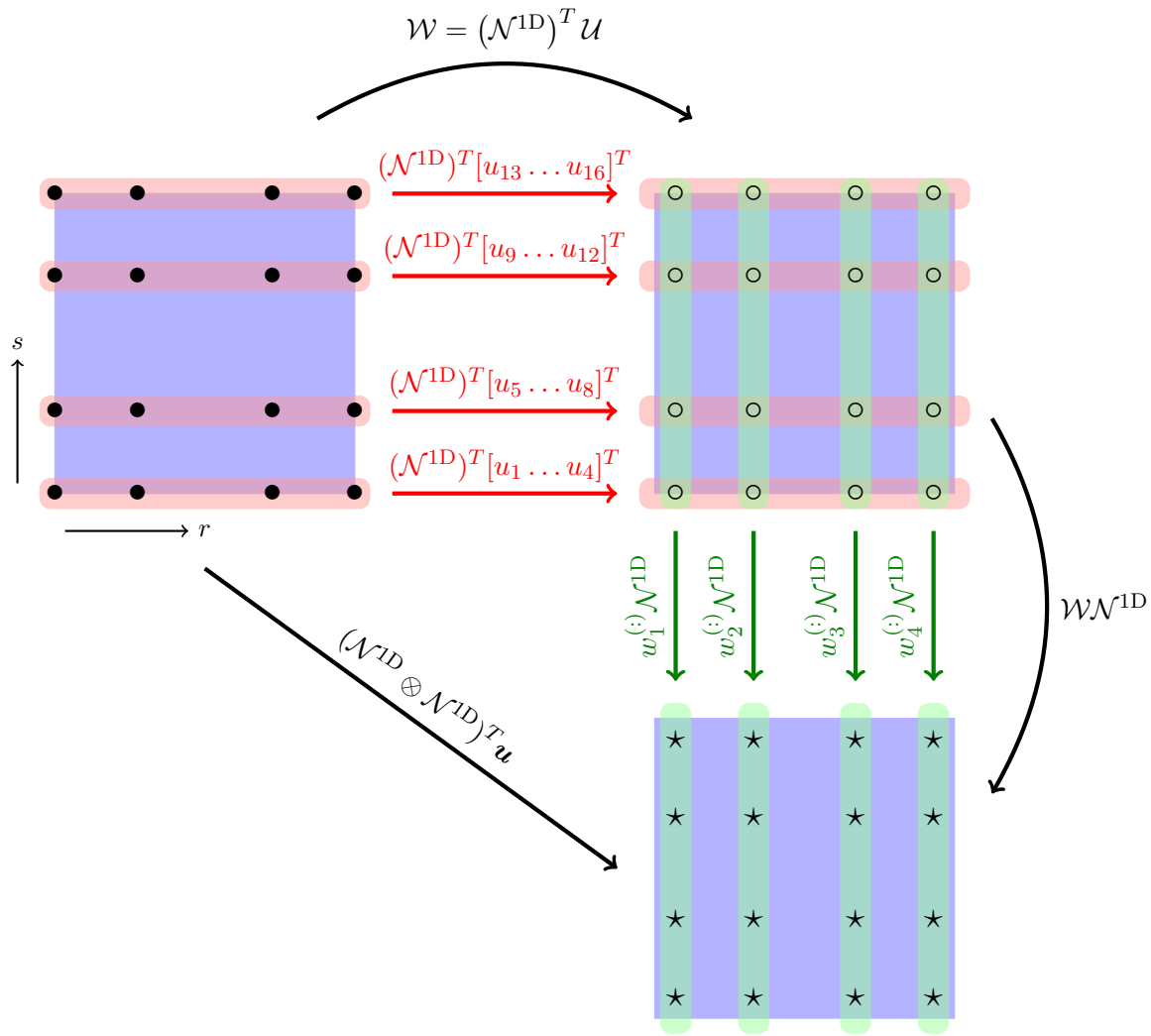

The first step is the multiplication $\mathcal{W} = \left(\mathcal{N}^{1D}\right)^T \mathcal{U}$. The multiplication $\left(\mathcal{N}^{1D}\right)^T \boldsymbol{v}$ represents the evaluation of a one-dimensional function with node values $v$ on the 1D quadrature nodes. We depict this as follows:




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 $\left(\mathcal{N}^{1D}\right)^T \mathcal{U}$ 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,

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

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.

### 3.4 Extension to three space dimensions

The ideas shown in the graph above extend straight-forwardly to three spatial dimensions. The Kronecker product $\mathcal{N}^{1D} \otimes \mathcal{N}^{1D} \otimes \mathcal{N}^{1D}$ 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 $\left(\mathcal{N}^{1D} \otimes \mathcal{N}^{1D} \otimes \mathcal{N}^{1D}\right) \boldsymbol{u}^k$ 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

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

### 3.5 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.,

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

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 $\ell_i$ 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.

## 4 Details of computational cost

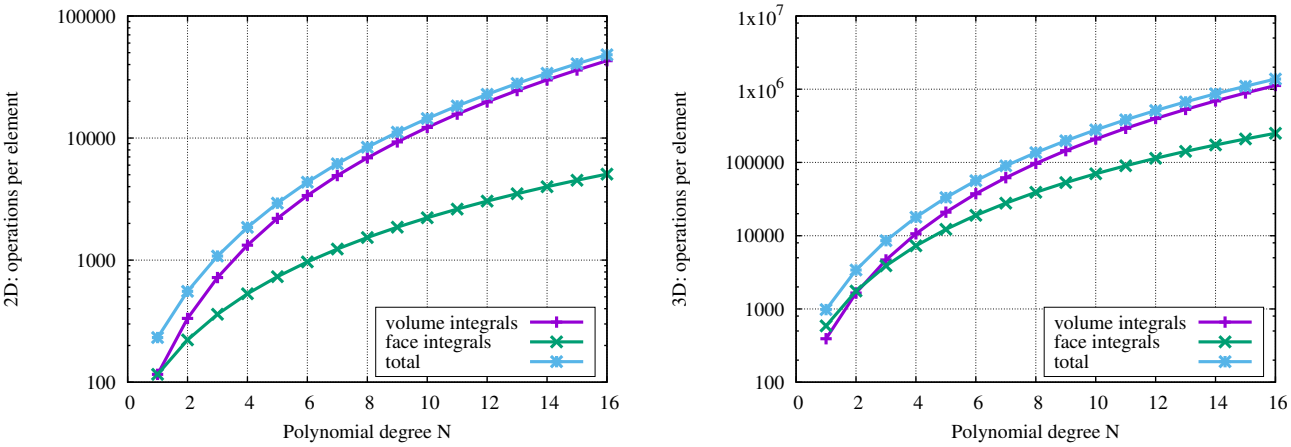
As we have seen above, the sum factorization step considerably reduces the computational effort to change from node values to values in quadrature points and the other way around. Let us re-consider the table from the beginning of this subsection and list the number of arithmetic operations for the operation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

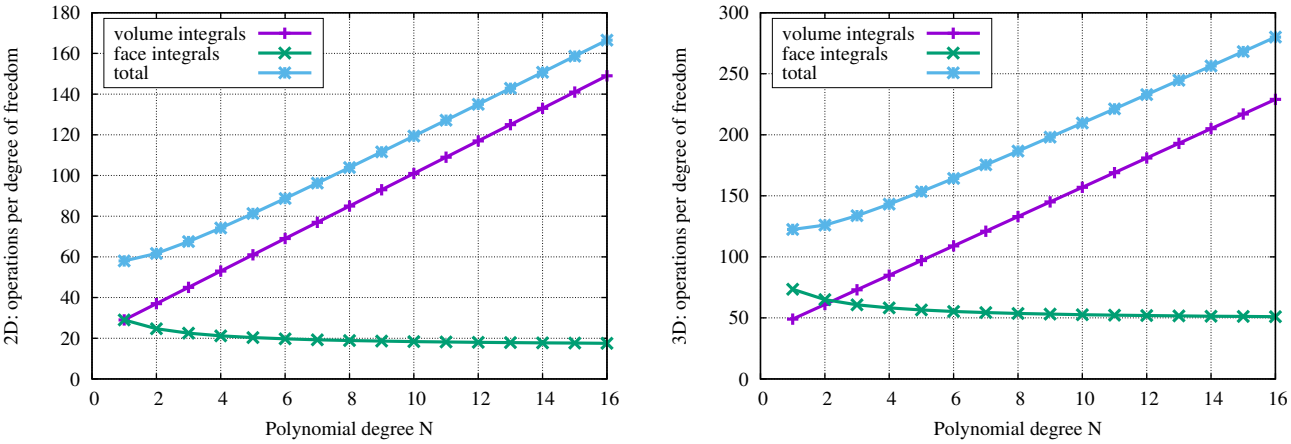
We see that the improvement in operations is about one order of magnitude for $N = 4$ in 3D.
Next, we analyze the cost per degree of freedom for the evaluation routines based on sum factorization for the advection equation with local integrals

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

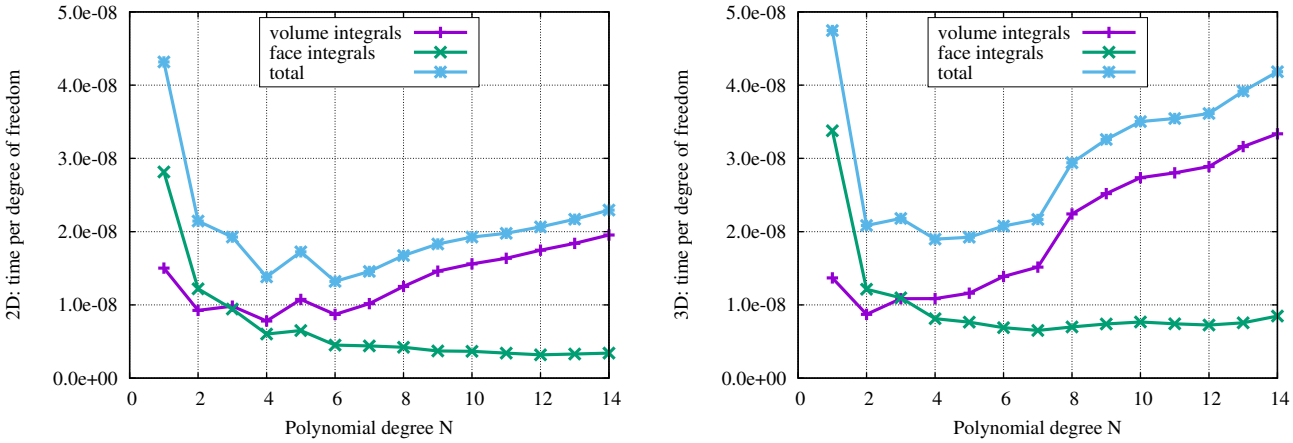
We display the work done per element for the 2D case (quadrilateral elements) on the left and the 3D case (hexahedral elements) on the right. We separately display the work for the volume integral and the face integrals. In order to make the work comparable, we assign the complete work on two faces (from 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. The data has been measured with the implementation from the `deal.II` finite element library (`www.dealii.org`) when running the integration on all six cores of an Intel Xeon E1650 (Sandy Bridge) CPU:



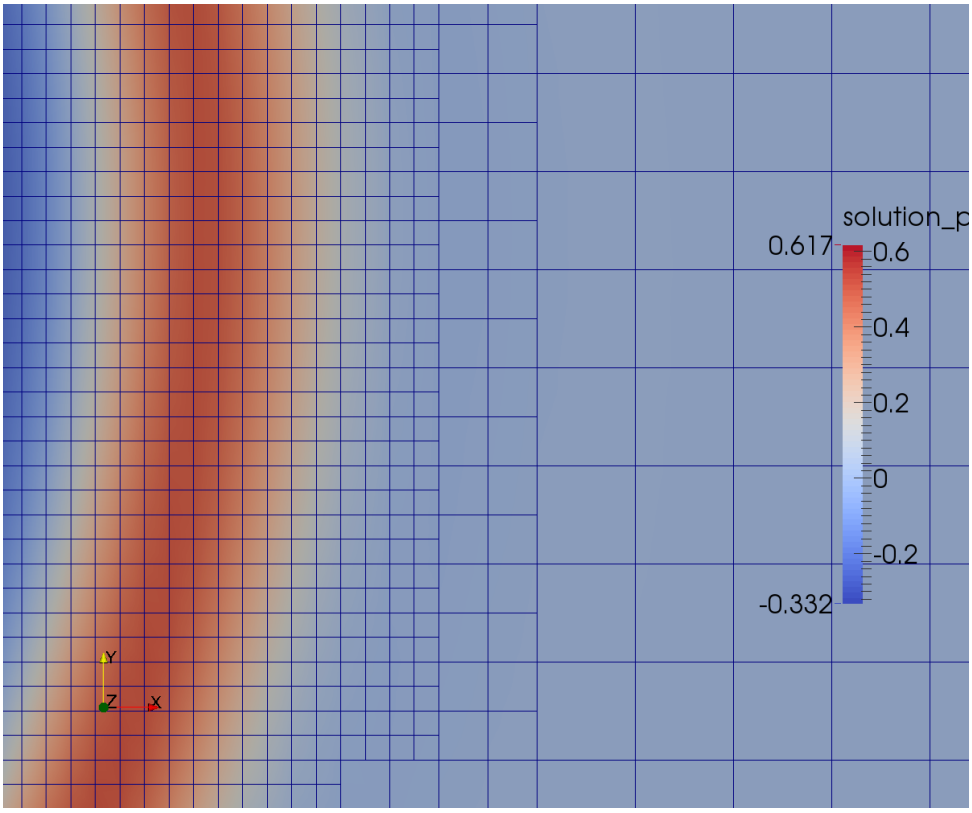

**Conclusions**

- 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

## 5 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)

## 6 Check yourself

- How are the nodes for quadrilaterals and hexahedra distributed?
- 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 (or matrix-based 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?