The Discontinuous Galerkin method for the Diffusion and Advection-Reaction equations

Matteo Cicuttin

Applied and Computational Electromagnetics group
Montefiore institute, University of Liège

UPMC - Paris, February 18, 2022

Outline

In this class we will discover how to translate PDEs in running code that solves them.
Very practical class. We will cover:

- Numerical integration
- Meshes
- Polynomial representation
- dG for diffusion
- dG for advection-reaction

Some hands-on exercises using Yaourt-FEM-DG

Yaourt-FEM-DG

Yaourt-FEM-DG is a code to teach the implementation of FEM/DG.

https://github.com/datafl4sh/yaourt-fem-dg

Goals of Yaourt-FEM-DG:

- Not a trivial Matlab script and not a complex monster: just what you need to start doing interesting things
- No aim for high performance or support for fancy meshes
- A single person should be able to master the whole code (but not in the three hours of this class!)
- No external dependencies (well, almost...)

Questions:

- Which is your level of C++?
- Is everyone set up to run the code?

The ingredients of a PDE solver

Let $\Omega \subset \mathbb{R}^d$ with $d \in \{1, 2, 3\}$ be an open, bounded and connected polytopal domain. We will consider the model problem

$$\begin{aligned} -\Delta u &= f \quad && \text{in } \Omega, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

with $f \in L^2(\Omega)$. By setting $V := H_0^1(\Omega)$. In weak form, find $u \in V$ such that, for all $v \in V$

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v.$$

By looking at the last equation we can deduce what we will need:

- $\int$: numerical integration
- Ω: domain representation
- u, v: basis for approx. space
- u is unknown: linear solver

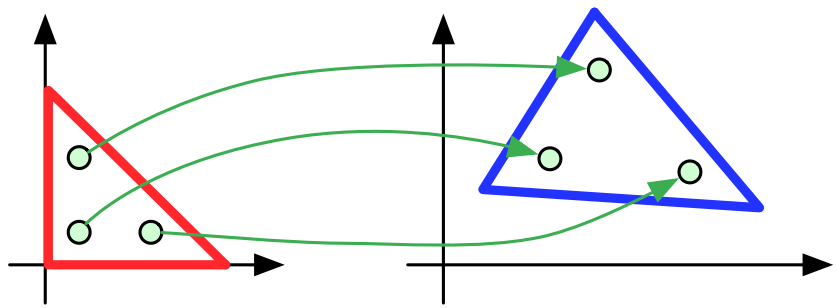
Numerical integration: Quadrature rules

Quadrature $Q = (Q_w, Q_p)$: collection of $|Q|$ points and associated weights. Definite integrals are computed as weighted sum of evaluations of the integrand on the points prescribed by the quadrature:

$$\int_{-1}^{1} f(\eta)\, d\eta = \sum_{i=1}^{|Q|} w_i f(\eta_i), \qquad w_i \in Q_w, \eta_i \in Q_p$$

A quadrature is given on a specific *reference element*. Because of that you need to map it on your physical element. In particular:

- Map points from the reference to physical (affine transform)
- Multiply weights by measure of physical element (Jacobian)



Same thing of change of variable in classical integration.

Numerical integration: Gaussian quadrature in 1D

An n-point Gaussian quadrature integrates exactly polynomials up to degree $2n-1$ and is defined on a reference element (usually $[-1,1]$).

To integrate $f(x)$ on a generic interval:

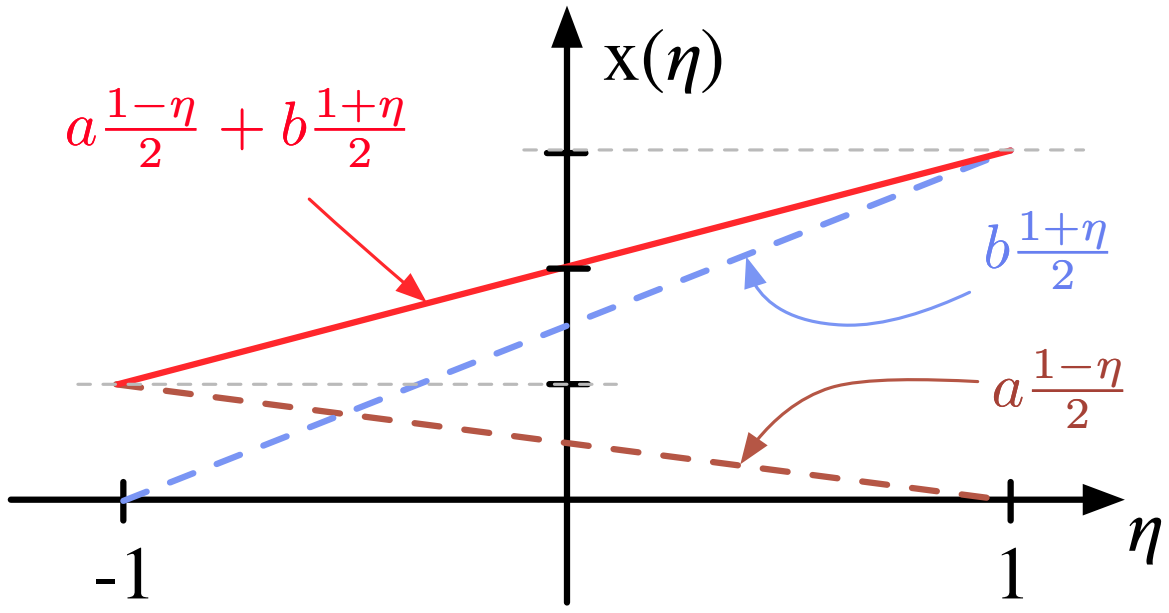
- Find the mapping $x(\eta) : [-1,1] \to [a,b]$
- Rescale w_i with scale factor $|x'(\eta_i)|$
- Compute the weighted sum

n	η_i	w_i
1	0	2
2	$\pm\frac{1}{\sqrt{3}}$	1
3	0	$\frac{8}{9}$
	$\pm\sqrt{\frac{3}{5}}$	$\frac{5}{9}$

$$\int_a^b f(x)dx = \int_{-1}^{1} f(x(\eta))|x'(\eta)|d\eta \approx \sum_{i=1}^{|Q|} w_i |x'(\eta_i)| f(x(\eta_i)).$$

Numerical integration: mapping in 1D

A linear transform in 1D is easily obtained:



$$x(\eta) = a\frac{1-\eta}{2} + b\frac{1+\eta}{2} \qquad x'(\eta) = \frac{b-a}{2}$$

Therefore:

$$\int_a^b f(x)dx = |\frac{b-a}{2}| \int_{-1}^{1} f(x(\eta))d\eta \approx \sum_{i=1}^{|Q|} w_i |\frac{b-a}{2}| f(x(\eta_i)).$$

Numerical integration: Using a Gaussian quadrature

Using a Gaussian quadrature, compute the integral

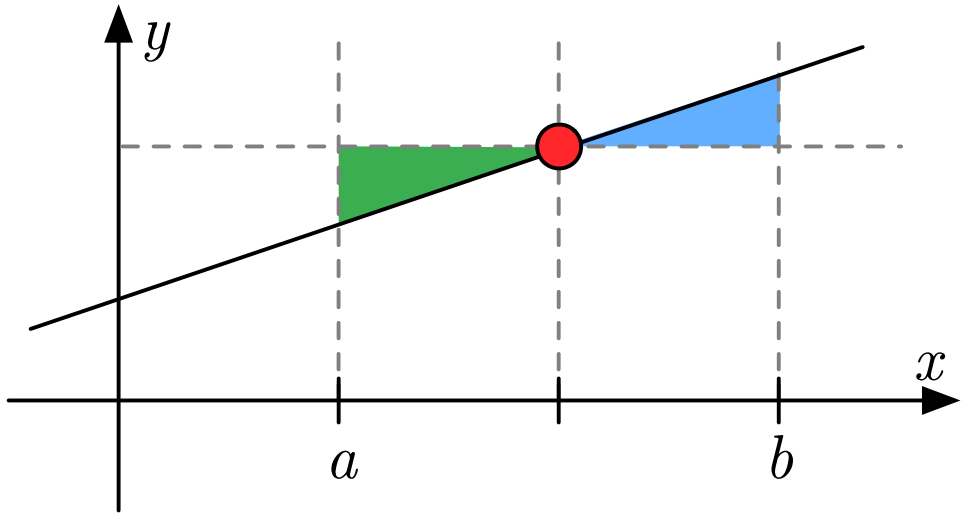
$$\int_2^3 2x + 1 \, dx.$$

1. Find transformation: $x(\eta) := 2\frac{1-\eta}{2} + 3\frac{1+\eta}{2} = \frac{\eta+5}{2}$. Note that $x(-1) = 2$ and $x(1) = 3$.
2. Map quadrature points: for order 1, only one point $\eta_0 = 0$. $x(0) = \frac{5}{2}$, midpoint of integration interval.
3. Rescale weights: for order 1, $w_0 = 2$. Rescaled weight is $2\frac{3-2}{2} = 1$.
4. Evaluate weighted sum: $1 \cdot (2\frac{5}{2} + 1) = 6$.

Correct: $\int_2^3 2x + 1 \, dx = 2\left[\frac{x^2}{2}\right]_2^3 + [x]_2^3 = 6.$

Numerical integration: Intuition behind order 1 quadrature

As we saw, **order 1** quadrature requires to evaluate function in the middle of integration interval $[a, b]$. Moreover, the weight reduces to $b - a$.



- Integrand is constant: you are computing the area of a rectangle $\Longrightarrow$ result you get is exact.
- Integrand is affine: area of rectangle still works, missing green area compensates excess blue area $\Longrightarrow$ result you get is exact.

Higher order quadratures work similarly. In general, quadratures for polynomials are exact.

Numerical integration: exercise

Using a second-order quadrature, compute the integral

$$\int_{-\sqrt{3}}^{\sqrt{3}} 3x^2 - 2x - 1 \, dx$$

Numerical integration: exercise

Using a second-order quadrature, compute the integral

$$\int_{-\sqrt{3}}^{\sqrt{3}} 3x^2 - 2x - 1 \, dx$$

1. Find transformation: $x(\eta) := -\sqrt{3}\frac{1-\eta}{2} + \sqrt{3}\frac{1+\eta}{2} = \sqrt{3}\eta$
2. Map quadrature points: $x(-\frac{1}{\sqrt{3}}) = -1$ and $x(\frac{1}{\sqrt{3}}) = 1$
3. Rescale weights: $1\frac{\sqrt{3}-(-\sqrt{3})}{2} = \sqrt{3}$
4. Evaluate in -1: $\sqrt{3}(3(-1)^2 - 2(-1) - 1) = 4\sqrt{3}$
5. Evaluate in 1: $\sqrt{3}(3(1)^2 - 2(1) - 1) = 0$

The final result is $4\sqrt{3}$.

Numerical integration: some pointers

There are **lots** of different types of quadrature. Keywords for simplices:

- 1D: Gauss, Gauss-Lobatto, ...
- 2D: Dunavant, Grundmann-Moeller, ...
- 3D: Keast, ARBQ, Grundmann-Moeller, ...

On quads: tensorized 1D quadratures (see `core/quadratures.hpp`).

If you need quadrature code look here:

https://people.sc.fsu.edu/~jburkardt/

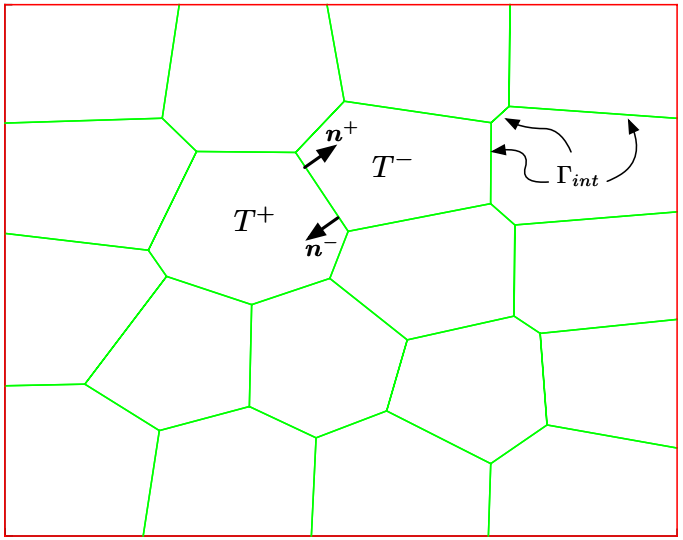
most of the code is needlessy complicated and messy, but if you're in a hurry...

Representation of the problem domain Ω

The problem domain Ω is discretized by dividing it in small elements. Let $\mathcal{T}_h$ be such a (suitable) subdivision of Ω in polytopal cells T:

We define:

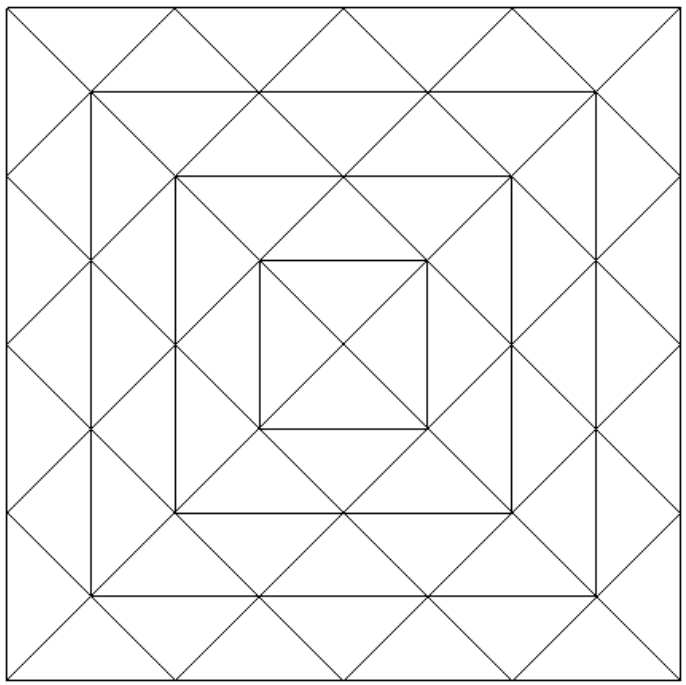
- $\Gamma := \cup_{T \in \mathcal{T}_h} \partial T$ (skeleton)
- $\Gamma_{int} = \Gamma \setminus \partial\Omega$
- T^+ and T^- generic elements sharing a face
- $e := T^+ \cap T^- \subset \Gamma_{int}$
- $\boldsymbol{n}^+$ and $\boldsymbol{n}^-$ normals of T^+ and T^- on e



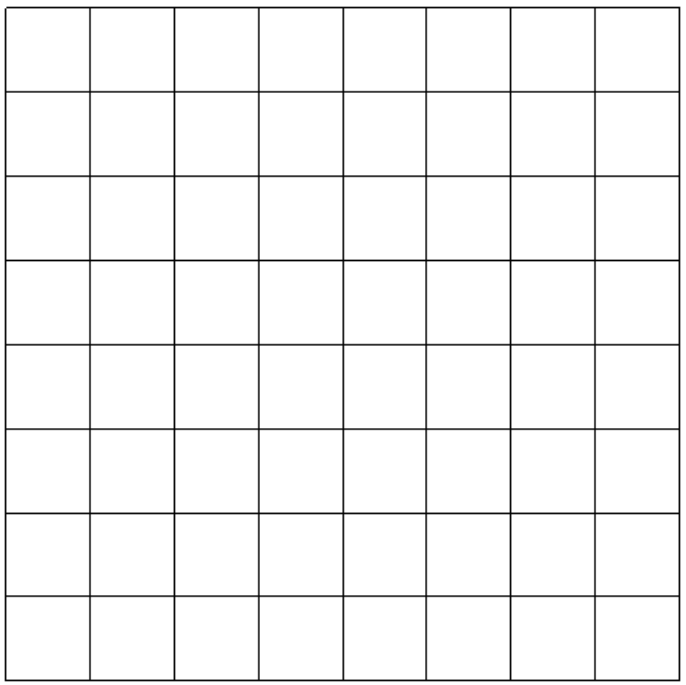
Codes usually expose this information. We won't discuss data structures, we'll only look at the Yaourt API.

Meshes

In this class we will consider only simplicial and cartesian meshes.



`simplicial_mesh<T>`



`quad_mesh<T>`

Code session 1

You have some time now to start exploring the code.

1. Look at `ex1_geometry.cpp`, it gives you a basic idea on how to iterate on elements and do operations on them.
2. Look at `core/mesh.hpp`, you can see the implementation of many operations on elements.
3. Pick an operation from `core/mesh.hpp` and modify `ex1_geometry.cpp` to do it. I suggest `normal()`, it is a 1-line modification.
4. Look at `ex2_quadrature.cpp`. Modify it as required in the comments.

Representing the DG space in a computer

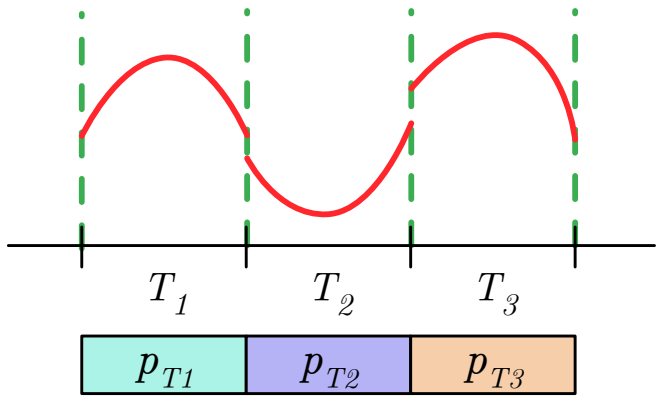
In dG we attach a polynomial of degree k to each mesh cell:

$$V_h := \{v \in L^2(\Omega) \mid \forall T \in \mathcal{T}_h, v_{|T} \in \mathbb{P}_d^k(T)\}.$$

Choose a basis of $\mathbb{P}_d^k(T)$, for example $\phi := \{1, x, x^2, \ldots, x^k\}$ in 1D.
Once the basis is fixed, any polynomial can be represented by storing only the coefficients p_i in

$$p(x) = \sum_{i=1}^{N_d^k} p_i \phi_i(x), \qquad N_d^k = \binom{k+d}{d}.$$

Collect the coefficients p_i for all the elements T in a vector and you get a representation of an element of V_h.
The coefficients are also called **degrees of freedom** (DoFs).

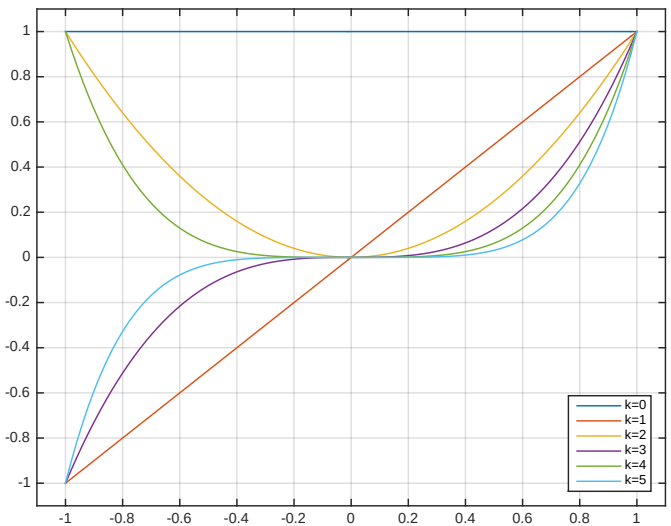


Choosing a polynomial basis

Plain monomials quickly lead to ill-conditioned matrices. There are better bases, for example the *scaled monomials* and the *Legendre polynomials*.

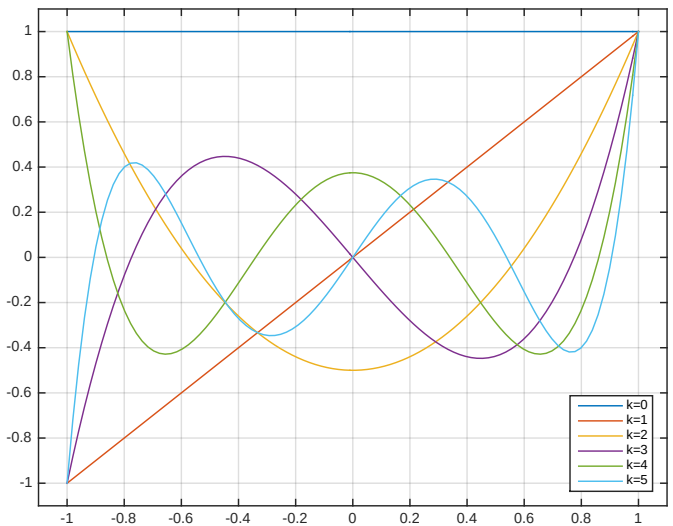
Scaled monomial basis:

- Simple to implement
- Cheap to evaluate
- Not orthogonal
- Any element shape



Scaled legendre basis:

- Less simple to implemented
- Expensive to evaluate
- Orthogonal
- Tricky on i.e. triangles



Scaled monomial basis

For simplicity we will use the scaled monomial basis centered on the barycenter $\bar{\mathbf{x}}_T$ of T:

$$\mathbb{P}_d^k(T) = \operatorname{span}\left\{ \prod_{i=1}^{d} \tilde{x}_{T,i}^{\alpha_i} \mid 1 \leq i \leq d \ \wedge \ 0 \leq \sum_{i=1}^{d} \alpha_i \leq k \right\}.$$

where $\tilde{\mathbf{x}}_T = (\mathbf{x} - \bar{\mathbf{x}}_T)/h_T$ and $\tilde{x}_{T,i}$ is the i-th component of $\tilde{\mathbf{x}}_T$.

Using bases and quadratures

Given a mesh $\mathcal{T}_h$, let $V_h := \{v \in L^2(\Omega) \mid \forall T \in \mathcal{T}_h, v_{|T} \in \mathbb{P}_d^k(T)\}$.
Suppose we want project a function f on V_h by solving the problem

$$(u_h, v_h)_\Omega = (f, v_h)_\Omega \qquad \forall v_h \in V_h.$$

By splitting integration element by element, we rewrite

$$\sum_{T \in \mathcal{T}_h} (u_T, v_T)_T = \sum_{T \in \mathcal{T}_h} (f, v_T)_T.$$

Note that summation terms are independent, we can solve element-by-element.

Using bases and quadratures: projecting a function (I)

We want to obtain a matrix representation of our problem. Warning: I will drop function subscripts!
For each element, we expand local LHS in the polynomial basis:

$$\int_T u(x)v(x) = \int_T \sum_{i=1}^{N_d^k} v_i \phi_i(x) \sum_{j=1}^{N_d^k} u_j \phi_j(x).$$

Remember that u_i and v_i are coefficients of a polynomial. Now we introduce the *mass matrix*

$$\mathbf{M}_{ij} = \int_T \phi_i(x)\phi_j(x)dx.$$

Finally, let $\mathbf{u} = \{u_j\}$ and $\mathbf{v} = \{v_i\}$ and rewrite using $\mathbf{M}$:

$$\int_T u(x)v(x)dx = \sum_{i=1}^{N_d^k} v_i \sum_{j=1}^{N_d^k} \mathbf{M}_{ij} u_j = \mathbf{v}^T \mathbf{M} \mathbf{u}.$$

Using bases and quadratures: projecting a function (II)

The integrals are still hidden in the mass matrix, time to use quadratures:

$$\mathbf{M}_{ij} = \int_T \phi_i(x)\phi_j(x)dx = \sum_{q=1}^{|Q|} w_q \phi_i(x_q)\phi_j(x_q),$$

where w_q and x_q are weights and points prescribed by the quadrature.
Let $\mathbf{\Phi}_k(x) = \{\phi_i(x)\}_{1 \leq i \leq N_d^k}$, where k is the degree of the space $\mathbb{P}_d^k(T)$.
The mass matrix can finally be written as

$$\mathbf{M} = \sum_{q=1}^{|Q|} w_q \mathbf{\Phi}_k(x_q)\mathbf{\Phi}_k(x_q)^T$$

Using bases and quadratures: projecting a function (III)

We proceed similarly for the right-hand side:

$$\mathbf{b} = \sum_{q=1}^{|Q|} w_q f(x_q) \mathbf{\Phi}_k(x_q)$$

We finally translate our local problem $(u_T, v_T)_T = (f, v_T)_T$ to the linear system

$$\mathbf{M}\mathbf{u} = \mathbf{b}.$$

Solving for $\mathbf{u}$ on each T we get the coefficients of the polinomial which is the L^2 projection of f on $\mathbb{P}_d^k(T)$. Collect all the local solution in an array and you get the global projection on V_h.

Code session 2

You now have some time to work on the code implementing the projection problem

1. Open `ex3_projection.cpp` and analyze "assembly" loop and "postprocess" loop.
2. In the postprocess loop, add the code to compute the L^2-error of the projection.

Remember:

- Look into the assembly loop how to obtain $\boldsymbol{\phi}(\boldsymbol{x})$
- Once you recovered the local solution, you can use the function `dot(a,b)` to compute the weighted sum giving the value of the polynomial
- Use a quadrature of degree $2k+1$

Summary

We learned the following things:

- How the problem domain is handled in the code
- How we compute integrals numerically
- How polynomials are represented numerically
- How simple problems like the computation of a projection are solved

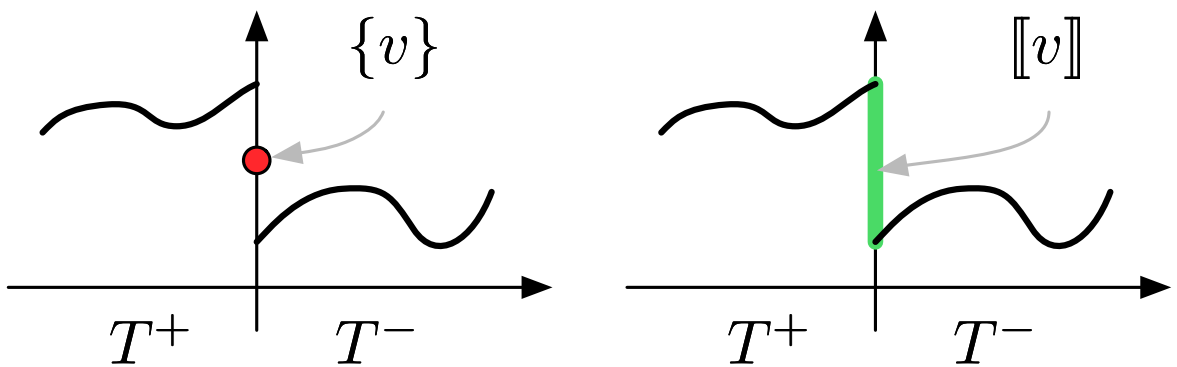
Questions?

Notation recall

Let $v : \Omega \to \mathbb{R}$ and let $F \in \Gamma_{int}$ be the face shared by elements T^+, T^-

$$\text{Average: } \{v\}_F(x) := \frac{1}{2}\left[v|_{T^+}(x) + v|_{T^-}(x)\right]$$

$$\text{Jump: } [\![v]\!]|_F(x) := v|_{T^+}(x) - v|_{T^-}(x)$$



If F belongs to the boundary of the domain (i.e. $e \subset \partial T \cap \partial\Omega$):

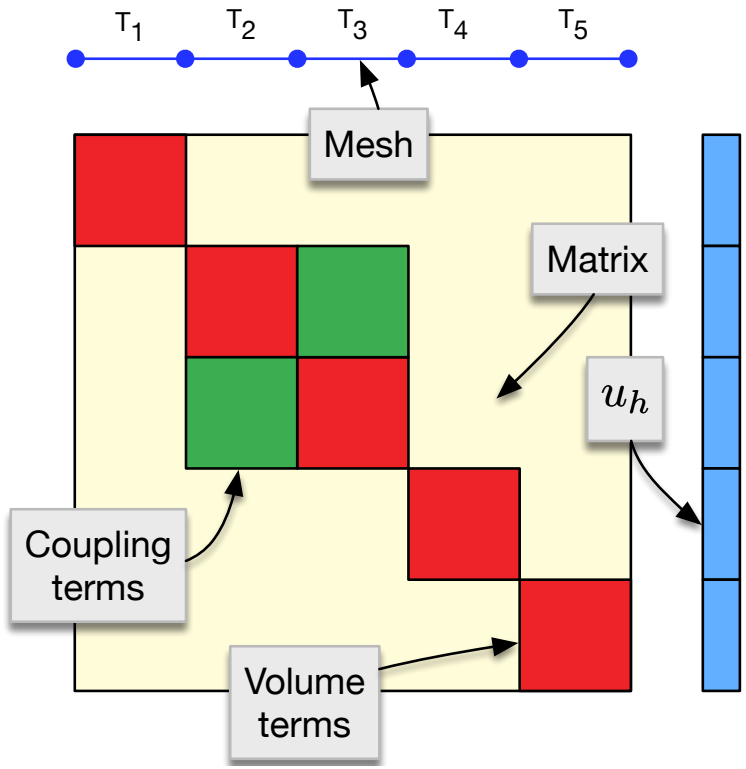
$$\{v\}_F(x) := v|_T(x) \qquad \text{and} \qquad [\![v]\!]|_F(x) := v|_T(x)$$

If v is vector-valued, the average and jump operators act component-wise.

Diffusion: Symmetric Interior Penalty dG

Let $V_h := \{v \in L^2(\Omega) \mid \forall T \in \mathcal{T}_h, v|_T \in \mathbb{P}_d^k(T)\}$. $u_h, v_h \in V_h$.

$$a_h^{sip}(u_h, v_h) = \sum_{T\in\mathcal{T}} \int_T \nabla_h u_h \cdot \nabla_h v_h$$
$$- \sum_{F\in\Gamma} \int_F \{\nabla_h u_h\} \cdot \boldsymbol{n}_F [\![v_h]\!]$$
$$- \sum_{F\in\Gamma} \int_F [\![u_h]\!] \{\nabla_h v_h\} \cdot \boldsymbol{n}_F$$
$$+ \sum_{F\in\Gamma} \int_F \frac{\eta}{h_F} [\![u_h]\!][\![v_h]\!]$$



Find $u_h \in V_h$ s.t. $\quad a_h^{sip}(u_h, v_h) = \int_\Omega f v_h \quad$ for all $v_h \in V_h$

Goal: obtain matrices for a_h^{sip} and r.h.s. This process is called **assembly**.

Assembly - Cell contributions

Let's start to assemble the **cell contributions only**:

$$\sum_{T\in\mathcal{T}} \int_T \nabla_h u_h \cdot \nabla_h v_h \qquad \text{and} \qquad \int_T f v_h$$

In single element T we write **for each test function** ϕ_n $(1 \leq n \leq N_d^k)$:

$$u_1 \int_T \nabla\phi_1 \cdot \nabla\phi_1 + \ldots + u_n \int_T \nabla\phi_n \cdot \nabla\phi_1 \qquad \text{and} \qquad \int_T f\phi_1$$

$$\vdots$$

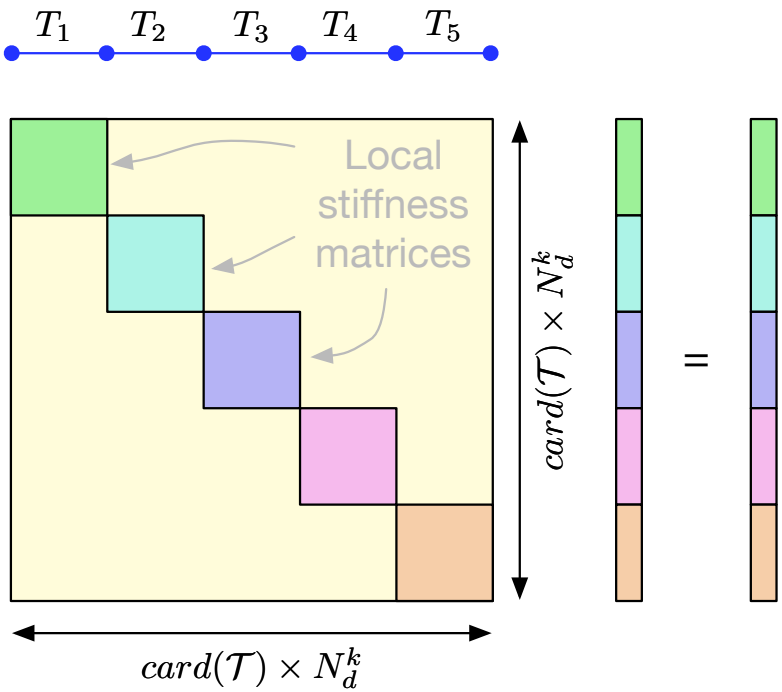
$$u_1 \int_T \nabla\phi_1 \cdot \nabla\phi_n + \ldots + u_n \int_T \nabla\phi_n \cdot \nabla\phi_n \qquad \text{and} \qquad \int_T f\phi_n$$

Notice similarity with projection. This time **stiffness matrix** in the LHS.

$$\mathbf{S} = \sum_{q=1}^{|Q|} w_h (\nabla\mathbf{\Phi}_k(x_q))(\nabla\mathbf{\Phi}_k(x_q))^T \qquad \mathbf{b} = \sum_{q=1}^{|Q|} w_h f(x_q)\mathbf{\Phi}(x_q)$$

Assembly - Cell contributions

Local stiffness matrices must be assembled in a global matrix now.



Consider a 1D mesh composed on 5 elements (depicted in blue).

- Each element gets its own set of equations in the global matrix.
- The structure of the global matrix is related to the mesh.
- Knowing the mesh, it is easy to determine the size of the system.

There is no coupling between elements yet, face terms are missing.

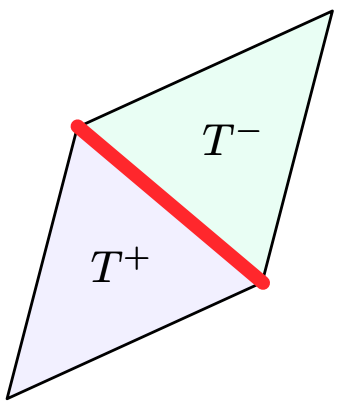
Assembly - Face-related terms

Let's proceed with consistency, symmetry and stabilization terms:

$$-\sum_{F\in\Gamma}\int_F(\{\nabla_h u\}\cdot\boldsymbol{n}_F[\![v_h]\!]+[\![u]\!]\{\nabla_h v_h\}\cdot\boldsymbol{n}_F)$$

$$+\sum_{F\in\Gamma}\int_F\frac{\eta}{h_F}[\![u]\!][\![v_h]\!]$$

T^+: the element **I am visiting**, T^- the neighbour!



- These terms will produce off-diagonal contributions
- They will "couple" adjacent elements
- To see where they go in the matrix, expand the jump and average operators

From now on, $\boldsymbol{n}_F$ is the outward normal from T^+!

Assembly - Face-related terms

Consider the consistency term. Let me use $v^+ := v|_{T^+}$ and $v^- := v|_{T^-}$

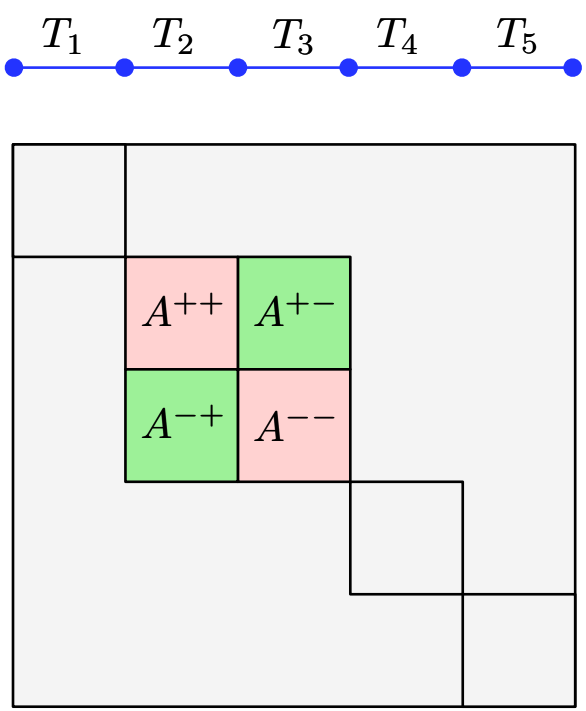
$$\int_F \{\nabla_h u\} \cdot \boldsymbol{n}_F [\![v_h]\!] = \frac{1}{2} \int_F (\nabla_h u^+ \cdot \boldsymbol{n}_F + \nabla_h u^- \cdot \boldsymbol{n}_F)(v_h^+ - v_h^-)$$

- The terms in red will be on the diagonal
- The terms in green will be off-diagonal

$$A^{++} = \frac{1}{2} \int_F (\nabla u^+ \cdot \boldsymbol{n}_F) v^+ \qquad A^{+-} = \frac{1}{2} \int_F (\nabla u^- \cdot \boldsymbol{n}_F) v^+$$

$$A^{-+} = -\frac{1}{2} \int_F (\nabla u^+ \cdot \boldsymbol{n}_F) v^- \qquad A^{--} = -\frac{1}{2} \int_F (\nabla u^- \cdot \boldsymbol{n}_F) v^-$$

Assembly - Face-related terms



Suppose $T^+ = T_2$ and $T^- = T_3$

- T_2 and T_3 are adjacent in the global vector of DoFs
- Off-diagonal terms introduce a coupling between adjacent elements
- Then you assemble cell T^+, you need to assemble only A^{++} and A^{+-}, the others two get assembled when you pass on T^-
- Don't forget the boundary face terms!

Assembly - Dirichlet boundary conditions

We want now to consider non-homogeneous Dirichlet boundary conditions:

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega, \\ u &= g \qquad \text{on } \partial\Omega, \end{aligned}$$

Non-homogeneous BCs are enforced weakly adding two terms to the right hand side:

$$l_D(g, v_h) := \int_\Omega f v_h - \int_{\partial\Omega} g \nabla_h v_h \cdot \boldsymbol{n}_F + \sum_{F \in \partial\Omega} \frac{\eta}{h_F} \int g v_h.$$

Advection-reaction equation

The advection-reaction model problem is the following

$$\begin{aligned} \beta \cdot \nabla u + \mu u &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{in } \partial\Omega^-, \end{aligned}$$

where $\partial\Omega^-$ is the *inflow boundary*

$$\partial\Omega^- := \{x \in \partial\Omega | \beta(x) \cdot n(x) < 0\}.$$

dG for advection-reaction

Let's consider now the dG method for the advection-reaction equation. The left-hand side bilinear form is:

$$a_h^{upw}(u_h, v_h) := \int_\Omega [\mu u_h v_h + (\beta \cdot \nabla_h u_h) v_h] + \int_{\partial\Omega} (\beta \cdot \boldsymbol{n})^{\ominus} u_h v_h$$
$$- \sum_{F \in \Gamma_{int}} \int_F (\beta \cdot \boldsymbol{n}_F) [\![u_h]\!] \{v_h\}$$
$$+ \sum_{F \in \Gamma_{int}} \int_F \frac{\eta}{2} |\beta \cdot \boldsymbol{n}_F| [\![u_h]\!] [\![v_h]\!]$$

- Remember that $x^{\ominus} := \frac{1}{2}(|x| - x)$
- Note the asymmetry of the problem

Code session 3

Now we will work on the actual DG solvers:

- I will give you a quick overview of the diffusion and advection-reaction solvers
- Task #1: complete the diffusion assembly with symmetry and stabilization terms
- Task #2: verify convergence rates
- Task #3: complete the assembly with Dirichlet b.c. contributions
- Task #4 complete the advection-reaction assembly with the skeleton terms
- Task #5 verify visually the results

Linear solvers

In the code you might have remarked that the conjugated gradient was used to solve the linear system.

- For the diffusion problem, CG is fine. A preconditioner could help speeding up convergence.
- For the advection-reaction problem, plain CG **does not work**. You should use other solvers like BiCGStab, GMRES or QMR. In the code I used what is known as CG on the normal equations: instead of solving $\mathbf{Ax} = \mathbf{b}$ I solved $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$. This works, but for a number of reasons is not optimal.
- Please read "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain" by J. R. Shewchuk. It is a **must**.

Questions/feedback

matteo.cicuttin@enpc.fr
matteo.cicuttin@uliege.be