# Implementation of the dG method

Matteo Cicuttin

CERMICS - École Nationale des Ponts et Chaussées
Marne-la-Vallée

March 6, 2017

# Outline

- Model problem, fix notation
- Representing polynomials
- Computing integrals
- Assembly, solve, postprocess
- Matlab code to solve 1D model problem

# Model problem

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

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

with $f \in L^2(\Omega)$. By setting $V := H_0^1(\Omega)$, its weak form is

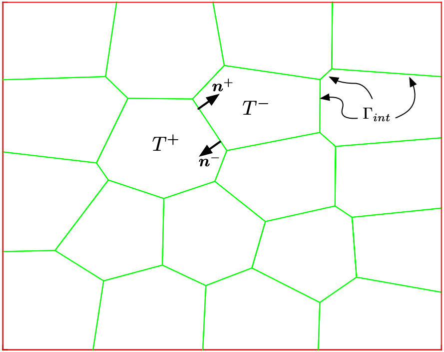
Find $u \in V$ such that $(\nabla u, \nabla v)_\Omega = (f, v)_\Omega$ for all $v \in V$.

# Notation I: mesh and elements

Let $\mathcal{T}$ be a suitable subdivision of $\Omega$ in polytopal elements $T$. We define the *skeleton* $\Gamma := \cup_{T\in\mathcal{T}} \partial T$

Moreover, we define:

- $\Gamma_{int} = \Gamma \setminus \partial\Omega$
- $T^+$ 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 $e$



# Notation II: jump and average

Let $q : \Omega \to \mathbb{R}$ and $\phi :\to \mathbb{R}^d$

$$\text{Average: } \{q\}|_e := \frac{1}{2}(q^+ + q^-) \qquad \{\boldsymbol{\phi}\}|_e := \frac{1}{2}(\boldsymbol{\phi}^+ + \boldsymbol{\phi}^-)$$

$$\text{Jump: } [\![q]\!]|_e := q^+\boldsymbol{n}^+ + q^-\boldsymbol{n}^- \qquad [\![\boldsymbol{\phi}]\!]|_e := \boldsymbol{\phi}^+ \cdot \boldsymbol{n}^+ + \boldsymbol{\phi}^- \cdot \boldsymbol{n}^-$$

If $e$ belongs to the boundary of the domain (i.e. $e \subset \partial T \cap \partial\Omega$) we just drop the terms with $-$:

$$\{\boldsymbol{\phi}\}|_e := \boldsymbol{\phi}^+ \qquad \text{and} \qquad [\![q]\!]|_e := q^+\boldsymbol{n}^+$$

# Symmetric Interior Penalty dG

SIP dG method is derived from the following equation:

$$\sum_{T\in\mathcal{T}}\int_T \nabla u\cdot\nabla v - \int_\Gamma(\{\nabla u\}\cdot[\![v]\!]+\{\nabla v\}\cdot[\![u]\!]-\eta[\![u]\!]\cdot[\![v]\!]) = \int_\Omega fv = (f,v)$$

In SIP dG we approximate the solution of our equation using piecewise continuous polynomials on the elements.

$$S_h^p := \left\{w_h \in L^2(\Omega) : w_h|T \in \mathbb{P}_d^k(T), T \in \mathcal{T}\right\}$$

SIP dG method will then be:

$$\text{Find } u_h \in S_h^p \text{ s.t. } a(u_h,v_h) = (f,v_h), \qquad \forall v_h \in S_h^p$$

where $a(u,v) : S_h^p \times S_h^p \to \mathbb{R}$

$$a(u,v) = \sum_{T\in\mathcal{T}}\int_T \nabla u\cdot\nabla v - \int_\Gamma(\{\nabla u\}\cdot[\![v]\!]+\{\nabla v\}\cdot[\![u]\!]-\eta[\![u]\!]\cdot[\![v]\!])$$

# Representing polynomials

We need to be able to represent $d$-variate polynomials of degree $k$ on cells: $p(x) \in \mathbb{P}_d^k(T)$. We introduce a basis of $\mathbb{P}_d^k(T)$: in 1D for example $\{1, x, x^2, \ldots\}$.

Once the basis is fixed, the coefficients $p_i$ fully determine the polynomial.

$$p(x) = \sum_{i=1}^{N_d^k} p_i \phi_i(x)$$

where $N_d^k$ is the size of the basis for $\mathbb{P}_d^k(T)$:

$$N_d^k = \binom{k+d}{d}$$

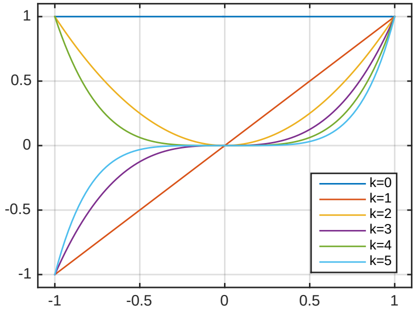
The coefficients of the basis will be called *degrees of freedom (DoFs)*.

# Scaled monomial basis

It is better, however, to use the so-called "scaled monomial basis" centered on the barycenter $\bar{\mathbf{x}}_T$ of $T$:

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

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



# Integrals and mass matrix

We want to compute $\int_T p(x)q(x)$, where $p, q \in \mathbb{P}_d^k$. As we discussed, we can express polynomials as linear combinations of basis functions:

$$\int_T p(x)q(x) = \int_T \sum_{i=1}^{N_d^k} q_i \phi_i(x) \sum_{j=1}^{N_d^k} p_j \phi_j(x)$$

Introduce mass matrix:

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

Rewrite using mass matrix:

$$\int_T p(x)q(x) = \sum_{i=1}^{N_d^k} q_i \sum_{j=1}^{N_d^k} \mathbf{M}_{ij} p_j.$$

Let $\mathbf{p} = \{p_j\}$ and $\mathbf{q} = \{q_i\}$:

$$\int_T pq = \mathbf{q}^T \mathbf{M} \mathbf{p}.$$

# Mass matrix

The integral is now hidden inside the mass matrix

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

How to compute it? We need to do numerical integration using *quadrature rules*.

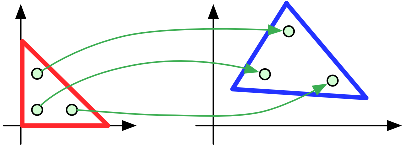
# Quadrature rules

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:

$$\int_{-1}^{1} f(x)\,dx = \sum_{i=1}^{|Q|} w_i f(x_i), \qquad w_i \in Q_w, x_i \in Q_p$$

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)



# Quadratures in practice

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, we usually tensorize.

Look here for code: `https://people.sc.fsu.edu/~jburkardt/`.

In Matlab code we use Golub-Welsch algorithm to compute Gauss quadrature.

# Mass matrix and stiffness matrix

We are now able to compute the mass matrix:

$$\mathbf{M}_{ij} = \int_T \phi_i(x)\phi_j(x) = \sum_{i=1}^{|Q|} \tilde{w}_i \phi_i(\tilde{x}_i)\phi_j(\tilde{x}_i),$$

where $\tilde{w}_i$ and $\tilde{x}_i$ are the quadrature weights and points after the transformations.
It is possible to build the stiffness matrix in the same way:

$$\mathbf{S}_{ij} = \int_T \nabla\phi_i(x) \cdot \nabla\phi_j(x) = \sum_{i=1}^{|Q|} \tilde{w}_i \nabla\phi_i(\tilde{x}_i) \cdot \nabla\phi_j(\tilde{x}_i).$$

These matrices will have size $N_d^k \times N_d^k$.

# A code

The numerical solution of a PDE, in general, consists of three phases:

- Assembly:
  Compute the local contributions for every $T$ and put them in the global system matrix,
- Solve:
  Solve the linear system $\mathbf{A}\mathbf{u} = \mathbf{f}$,
- Postprocess:
  Recover the values of the solution from the DoFs computed in the previous step.

# Assembly - Cell contributions

$$\sum_{T\in\mathcal{T}}\int_T \nabla u_h\cdot\nabla v_h - \int_\Gamma(\{\nabla u_h\}\cdot[\![v_h]\!] + \{\nabla v_h\}\cdot[\![u_h]\!] - \eta[\![u_h]\!]\cdot[\![v_h]\!]) = (f, v_h)$$

Remember:

- $v_h$ can be any function in $S_h^p$; we choose all the coefficients to be $1$
- for linearity, you can write one equation per basis function
- the coefficients $u_j$ are the unknowns

Then, for the terms in red, locally we get for $1 \le n \le N_d^k$

$$u_1\int_T \nabla\phi_1\cdot\nabla\phi_1 + \ldots + u_n\int_T \nabla\phi_n\cdot\nabla\phi_1 = f\phi_1$$

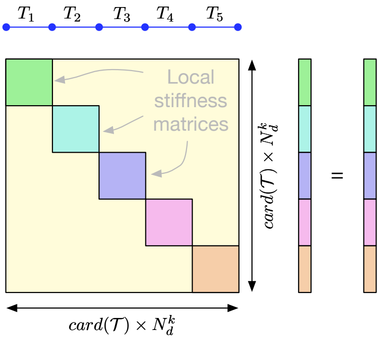
$$\vdots$$

$$u_1\int_T \nabla\phi_1\cdot\nabla\phi_n + \ldots + u_n\int_T \nabla\phi_n\cdot\nabla\phi_n = f\phi_n$$

We've got $N_d^k$ local equations for each element in $\mathcal{T}$.

# Assembly - Cell contributions

We now put the equations we obtained in a global matrix.



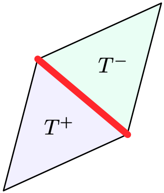

Consider a 1D mesh composed on 5 elements (depicted in blue).

- 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.

We haven't assembled the other terms yet. Note the decoupling.

# Assembly - Face-related terms

$$\sum_{T\in\mathcal{T}}\int_T \nabla u_h\cdot\nabla v_h - \int_\Gamma \left(\{\nabla u_h\}\cdot[\![v_h]\!] + \{\nabla v_h\}\cdot[\![u_h]\!] - \eta[\![u_h]\!]\cdot[\![v_h]\!]\right) = (f, v_h)$$




- We have three additional terms to assemble
- We expand them with the expressions for jump and average
- They will “couple” adjacent elements

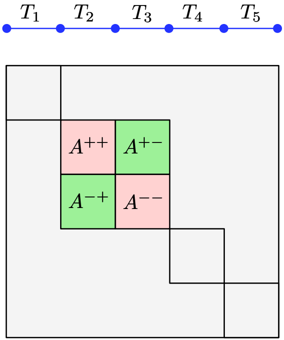
# Assembly - Face-related terms

$$\int_e \{\nabla u\} \cdot [\![v]\!] = \frac{1}{2}\int_e (\nabla u^+ + \nabla u^-) \cdot (v^+\boldsymbol{n}^+ + v^-\boldsymbol{n}^-) =$$
$$= \frac{1}{2}\int_e \left[ (\nabla u^+ \cdot v^+\boldsymbol{n}^+) + (\nabla u^+ \cdot v^-\boldsymbol{n}^-) + (\nabla u^- \cdot v^+\boldsymbol{n}^+) + (\nabla u^- \cdot v^-\boldsymbol{n}^-) \right]$$

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

$$A_1^{++} = \frac{1}{2}\int_e \nabla u^+ \cdot v^+\boldsymbol{n}^+ \qquad A_1^{+-} = \frac{1}{2}\int_e \nabla u^+ \cdot v^-\boldsymbol{n}^-$$
$$A_1^{-+} = \frac{1}{2}\int_e \nabla u^- \cdot v^+\boldsymbol{n}^+ \qquad A_1^{--} = \frac{1}{2}\int_e \nabla u^- \cdot v^-\boldsymbol{n}^-$$

# Assembly - Face-related terms




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

- You can see that the off-diagonal terms introduce a coupling between adjacent elements
- Remember that since in 1D faces are just points, integrating means that you need to just evaluate the functions there

# Assembly - Face-related terms

We have the two remaining terms, you handle them exactly as the previous one.

- $\int_e \{\nabla v\} \cdot [\![u]\!] = \frac{1}{2} \int_e (\nabla v^+ + \nabla v^-) \cdot (u^+ \boldsymbol{n}^+ + u^- \boldsymbol{n}^-)$
- $\int_e \eta [\![u]\!] \cdot [\![v]\!] = \int_e \eta (u^+ \boldsymbol{n}^+ + u^- \boldsymbol{n}^-) \cdot (v^+ \boldsymbol{n}^+ + v^- \boldsymbol{n}^-)$

Don't forget the two boundary faces!

# Solve

Once we have assembled the problem, we must solve it. In Matlab there are different ways:

- Use the backslash operator `u = A\f`
- Use one of the iterative solvers, `pcg()` is ok

# Postprocess

By solving, we computed the coefficients $u_{i,n}$, $1 \leq i \leq N_d^k$ for each element $1 \leq n \leq card(\mathcal{T})$. To recover the values of the solution at any point, we must evaluate them against the basis.

- We choose $N_p$ equispaced points on each element
- We evaluate there
- We plot the result

$$u_n(x_j) = \sum_{i=1}^{N_d^k} u_{i,n} \phi_i(x_i), \qquad 1 \leq j \leq N_p$$