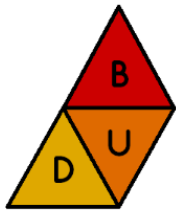
POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE



DUBeat

DUBeat: a C++ library for high order discontinuous Galerkin methods and applications to cardiac electrophysiology

Final Project for the course of
Advanced Programming for Scientific Computing

Authors: **Federica Botta, Matteo Calafà**

Advisors: Prof. Paola Antonietti, Prof. Christian Vergara
Co-advisor: Dr. Pasquale Africa
Academic Year: 2021-2022

Contents

Contents **i**

Introduction **1**

1 The Dubiner high-order discontinuous Galerkin method **2**
1.1 A brief presentation of discontinuous Galerkin methods . . . 2
1.2 The Dubiner spectral basis . . . 4
1.2.1 Dubiner basis in two dimensions . . . 5
1.2.2 Dubiner basis in three dimensions . . . 7
1.3 From modal to nodal evaluations . . . 9

2 Implementation details **11**
2.1 General structure . . . 11
2.2 Dubiner basis classes . . . 12
2.2.1 DUBValues . . . 12
2.2.2 DUBFEMHandler . . . 13
2.3 Geometry classes . . . 13
2.3.1 VolumeHandlerDG . . . 13
2.3.2 FaceHandlerDG . . . 14
2.3.3 DoFHandlerDG . . . 14
2.4 Quadrature classes . . . 15
2.4.1 QGaussLegendreSimplex . . . 15
2.5 Assembly classes . . . 15
2.5.1 AssembleDG . . . 15
2.6 Error classes . . . 17
2.6.1 ComputeErrorsDG . . . 17
2.7 Models . . . 18
2.7.1 ModelDG . . . 19
2.7.2 ModelDG_t . . . 19

2.7.3 LaplaceDG 20
2.7.4 HeatDG 20
2.7.5 MonodomainFHNDG 22

3 Dependencies, installation and execution 25
3.1 Dependencies 25
3.2 Installation 26
3.3 Execution 26
3.3.1 How to run the template simulation 26
3.3.2 Visualization of the results 27
3.4 Documentation, indentation and cleaning 27
3.5 Personalize the models to solve 28

4 Numerical results 29
4.1 Grid refinement convergence tests 29
4.1.1 Results for LaplaceDG 29
4.1.2 Results for HeatDG 34
4.1.3 Results for MonodomainFHNDG 34
4.2 Higher order convergence rates 37
4.2.1 Pseudo-realistic simulation 39

5 Conclusion 42

Acknowledgements 43

Bibliography 43

List of Figures 45

List of Tables 46

List of Acronyms 46

Introduction

DUBeat[1,2] (/dɑbit/) is a C++ open-source header library that provides high-order discontinuous Galerkin methods and applications to models from the cardiac electrophysiology. More precisely, it is possible to use DUBeat to solve differential problems using either the standard DGFEM basis or the high-order Dubiner basis (proposed by [8], see Section 1.2), both in 2 or 3 dimensions. The library is developed based on the `lifex` ([1]) and `deal.II` ([4]) libraries from which it collects core structures, algebraic methods and electrophysiology models.
This work originates from the research already performed in [6] about Dubiner methods and their performance in electrophysiology problems (mainly, the monodomain and bidomain problems [7], [12]). The original intuition in this work was the great suitability of high-order discontinuous methods such as Dubiner basis method to solve this kind of problems where solutions are represented by steep and fast waves. It is indeed well-known that standard methods are prone to not correctly capture the wave propagation unless the grid is very refined (see [16]). This last option is often not feasible while the increment of the order in high-order methods seems to return correct and accurate results with lighter computational resources overcoming this issue. The promising results in [6] lead to the creation of DUBeat.
The main theoretical setting is taken from this work while the main challenges of this project were the implementation and fine-tuning of a high-performance C++ code, the generalization to different problems and the extension of the Dubiner basis to three dimensions.
To fully comprehend the tools developed in this library, Section 1 is aimed to present the involved numerical methods under a rigorous but simple mathematical description. The structure of the code is instead comprehensively shown and discussed in Section 2 while Section 3 is useful for all the information regarding the personal use of the library. To conclude, we make use of Section 4 to guarantee the correct behaviour of the implemented numerical schemes and to show the outstanding performance of high-order methods.
To conclude, the authors of the library address the official documentation[1] for more information and to the official open-source repository[2] for the user contribution. Use and modification of the codes are limited under the GNU General Public License.

[1] https://matteocalafa.com/DUBeat
[2] https://github.com/teocala/DUBeat

1 | The Dubiner high-order discontinuous Galerkin method

In this section, we aim to present and describe the numerical methods that have been implemented in the codes. In Section 1.1 we start with a general description of the discontinuous Galerkin methods applied to standard linear elliptic problems. From this background, one can then proceed to Section 1.2 for the description of the high-order DG methods based on Dubiner basis. To conclude, the following Section 1.3 is aimed to describe the most relevant disparities between the DGFEM method and the Dubiner method and how to overcome these challenges.

1.1. A brief presentation of discontinuous Galerkin methods

A classical and complete description of DG methods can be found in [5], however we now refer to the derivations and notations from [11]. For our scopes, it is sufficient to present discontinuous Galerkin methods when applied to a standard Poisson problem with homogeneous Dirichlet boundary conditions. It reads

$$\begin{cases} -\Delta u = f & \mathbf{x} \in \Omega, \\ u = 0 & \mathbf{x} \in \partial\Omega, \end{cases} \tag{1.1}$$

where $D \in \mathbb{R}^d$ is a Lipschitz bounded domain and $f \in L^2(\Omega)$. The weak formulation of (1.1) is known to be

$$u \in H_0^1(\Omega) : \quad \int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v \quad \forall v \in H_0^1(\Omega). \tag{1.2}$$

We will call $V := H_0^1(\Omega)$ the original analytical space. We define a quasi-uniform triangulation $\mathcal{T}_h$ on Ω that is indexed by the mean grid size $h > 0$. Differently from the continuous FEM method, we aim to solve (1.2) projected on

$$V_h := \{u_h \in L^2(\Omega) : u_h|_{\mathcal{K}} \in \mathbb{P}^p(\mathcal{K}) \;\; \forall \mathcal{K} \in \mathcal{T}_h\},$$

i.e., the discretized space is composed of functions that are discontinuous on the borders between adjacent elements and are piece-wise polynomials of order at most $p \in \mathbb{N}$.
In addition, let us define $\mathcal{F}_h$ the set of faces of $\mathcal{T}_h$. Hence, the DG formulation adapted to the problem in (1.2) reads:

$$\begin{aligned} &\text{Find } u_h \in V_h : \\ &\sum_{\mathcal{K}\in\mathcal{T}_h} \int_{\mathcal{K}} \nabla u_h \cdot \nabla v_h - \sum_{F\in\mathcal{F}_h} \int_F \{\{\nabla_h u_h\}\} \cdot [\![v_h]\!] - \theta \sum_{F\in\mathcal{F}_h} \int_F [\![u_h]\!] \cdot \{\{\nabla_h v_h\}\} \\ +&\sum_{F\in\mathcal{F}_h} \int_F \gamma [\![u_h]\!] \cdot [\![v_h]\!] = \int_\Omega f v_h \qquad \forall v_h \in V_h, \end{aligned} \tag{1.3}$$

where:

- ∇_h is the piece-wise gradient (i.e. does not consider the discontinuity),
- u^+, u^- are used to indicate the evaluations of u on the two sides of one face,
- $[\![u]\!] := \begin{cases} u^+\mathbf{n}^+ + u^-\mathbf{n}^- & F \text{ is an interior face,} \\ u\mathbf{n} & F \text{ is an exterior face,} \end{cases}$
- $\{\{\nabla u\}\} := \begin{cases} (\nabla u^+ + \nabla u^-)/2 & F \text{ is an interior face,} \\ \nabla u & F \text{ is an exterior face,} \end{cases}$
- $\theta \in \{-1, 0, 1\}$ is the *penalty coefficient*[1],
- $\gamma = \mathcal{O}\left(\frac{p^2}{h}\right)$ is a regularity parameter.

In particular, the first two terms of (1.3) come directly from the integration by parts on every element $\mathcal{K}$ and the other two terms are added for stability purposes. At this point, one can easily derive the DG linear system to solve. Let $\{\varphi_i\}_{i=1}^{N_h}$ be a set of V_h basis functions and define $\mathcal{F}_h^I$ and $\mathcal{F}_h^B$ as respectively the set of interior and exterior faces. Therefore, we have that:

$$\left(V - I^T - \theta I + S\right) \mathbf{u} = \mathbf{f}, \tag{1.4}$$

where:

- $\mathbf{u} \in \mathbb{R}^{N_h}$ is the vector that contains the coefficients of the discretized solution with respect to the basis functions,
- $\mathbf{f}_i := \int_\Omega f\varphi_i$,

[1] It defines which DG method is used: Symmetric Interior Penalty method (SIP) for $\theta = 1$, Incomplete Interior Penalty method (IIP) for $\theta = 0$ and Non Symmetric Interior Penalty method (NIP) for $\theta = -1$. Note that, for Section 4 the SIP method has been used.

- $V_{i,j} := \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j$,
- $I_{i,j} := \sum_{F\in\mathcal{F}_h^I} \left(\frac{1}{2}\int_F \nabla_h\varphi_i^+ \cdot \mathbf{n}^+\varphi_j^+ - \frac{1}{2}\int_F \nabla_h\varphi_i^+ \cdot \mathbf{n}^+\varphi_j^-\right) + \sum_{F\in\mathcal{F}_h^B}\int_F \nabla_h\varphi_i^+\cdot\mathbf{n}^+\varphi_j^+$,
- $S_{i,j} := \sum_{F\in\mathcal{F}_h^I}\left(\int_F \gamma\varphi_i^+\varphi_j^+ - \int_F\gamma\varphi_i^+\varphi_j^-\right) + \sum_{F\in\mathcal{F}_h^B}\int_F\gamma\varphi_i^+\varphi_j^+$

It is important to notice that the basis functions have not been specified yet and, therefore, the previous formulation is valid both for a DGFEM method (i.e., a DG method that employs piece-wise FE basis functions) and, as we will see, also for the high-order method with Dubiner basis (presented in Section 1.2).
To conclude this section, we show how the choice for the reference problem in (1.1) can really generalize to a larger family of problems. First of all, the DG problem applied to (1.1) with non-homogeneous Dirichlet boundary conditions ($u = g$ on $\partial\Omega$) can be easily generalized as

$$\begin{aligned}
&\text{Find } u_h \in V_h :\\
&\sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \nabla u_h\cdot\nabla v_h - \sum_{F\in\mathcal{F}_h}\int_F \{\{\nabla_h u_h\}\}\cdot[\![v_h]\!] - \theta\sum_{F\in\mathcal{F}_h}\int_F [\![u_h]\!]\cdot\{\{\nabla_h v_h\}\}\\
&+\sum_{F\in\mathcal{F}_h}\int_F \gamma[\![u_h]\!]\cdot[\![v_h]\!] = \int_\Omega f v_h - \theta\sum_{F\in\mathcal{F}_h^B}\int_F g\nabla_h v_h\cdot\mathbf{n} + \sum_{F\in\mathcal{F}_h^B}\int_F \gamma g v_h \qquad \forall v_h\in V_h,
\end{aligned} \tag{1.5}$$

while the DG formulation for (1.1) with non-homogeneous Neumann boundary conditions ($\nabla u \cdot \mathbf{n} = g$ on $\partial\Omega$) reads

$$\begin{aligned}
&\text{Find } u_h \in V_h :\\
&\sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \nabla u_h\cdot\nabla v_h - \sum_{F\in\mathcal{F}_h^I}\int_F \{\{\nabla_h u_h\}\}\cdot[\![v_h]\!] - \theta\sum_{F\in\mathcal{F}_h^I}\int_F [\![u_h]\!]\cdot\{\{\nabla_h v_h\}\}\\
&+\sum_{F\in\mathcal{F}_h^I}\int_F \gamma[\![u_h]\!]\cdot[\![v_h]\!] = \int_\Omega f v_h + \sum_{F\in\mathcal{F}_h^B}\int_F g v_h \qquad \forall v_h\in V_h.
\end{aligned}$$

In addition, we stress the fact that additional zero and first order terms are treated as in the continuous case. For example, the addition of a forcing term ku in equation (1.1) simply requires to sum a matrix in the form $M_{i,j} := \int_\Omega k\varphi_i\varphi_j$. This is intuitively correct since stability terms are already added and no further integration by parts is involved. Hence, the above formulations extend very simply to general linear second order PDEs.

1.2. The Dubiner spectral basis

The Koornwinder-Dubiner basis, or simply Dubiner basis, is a spectral basis that originates from the works of Koornwinder [10] before and Dubiner [8] later. Instead, the extension to the

three-dimensional case has been firstly presented in [14].
We provide now the definition and basic properties of such a basis in both the two-dimensional and three-dimensional settings. In order to achieve this goal, we will refer to the notations from [14] and [3]. In particular, we adopt the same reference triangle as in [3].

1.2.1. Dubiner basis in two dimensions

Contrarily to the standard spectral basis, the Dubiner basis is defined on simplices rather than squares. This is possible thanks to a special transformation. Define the 2D reference triangle

$$\hat{K} = \{(\xi, \eta) : \xi, \eta \geq 0, \xi + \eta \leq 1\}, \tag{1.6}$$

and the reference square

$$\hat{Q} = \{(a, b) : -1 \leq a \leq 1, -1 \leq b \leq 1\}. \tag{1.7}$$

Then, consider the following transformation from $\hat{Q}$ to $\hat{K}$:

$$\xi = \frac{(1+a)(1-b)}{4}, \quad \eta = \frac{(1+b)}{2}. \tag{1.8}$$

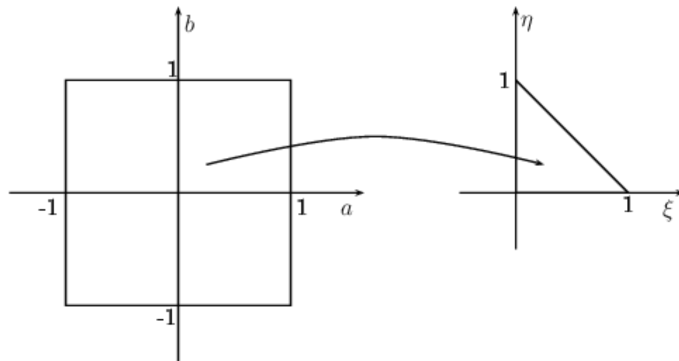


Figure 1.1: 2D transformation from the reference square $\hat{Q}$ to the reference triangle $\hat{K}$ (courtesy of Paola F. Antonietti).

Hence, the Dubiner basis functions are constructed as the transformations of suitable functions initially defined on $\hat{Q}$. More precisely, these original functions are defined as tensor products of Jacobi polynomials. Therefore, before proceeding, we give here one of the possible definitions and the main property of these orthogonal polynomials.

Definition 1.1 (Jacobi polynomials [15])**.** The n-th Jacobi polynomial of indices $\alpha, \beta \in \mathbb{R}$ is defined as:

$$P_n^{\alpha,\beta}(x) = \frac{(-1)^n}{2^n n!}(1-x)^{-\alpha}(1+x)^{-\beta}\frac{d^n}{dx^n}\left[(1-x)^{\alpha}(1+z)^{\beta}(1-z^2)^n\right].$$

We now state the main property of the Jacobi polynomials.

Proposition 1.1 (Orthogonality of Jacobi polynomials [15])**.** *$\{P_n^{\alpha,\beta}\}_{n\in\mathbb{N}}$ for $\alpha, \beta > -1$ is a set of $L^2(-1,1)$-orthogonal functions with respect to the Jacobi weight $w(x) = (1-x)^{\alpha}(1+x)^{\beta}$, indeed:*

$$\int_{-1}^{1}(1-x)^{\alpha}(1+x)^{\beta}P_n^{\alpha,\beta}(x)P_m^{\alpha,\beta}(x)\,dx = \frac{2^{\alpha+\beta+1}}{2n+\alpha+\beta+1}\frac{\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{\Gamma(n+\alpha+\beta+1)n!}\delta_{nm}.$$

We are ready now to define the Dubiner basis on the 2D reference triangle.

Definition 1.2 (2D Dubiner basis [3])**.** The 2D Dubiner basis function indexed by $(i,j) \in \mathbb{N}^2$ is defined as:

$$\varphi_{i,j}(\xi,\eta) = c_{i,j}(1-\eta)^i P_i^{0,0}\left(\frac{2\xi}{1-\eta}-1\right)P_j^{2i+1,0}(2\eta-1),$$

where $c_{i,j} = \sqrt{2(2i+1)(i+j+1)}$.

Figure 1.2: Dubiner basis functions that generate $\mathbb{P}^5(\hat{K})$ (courtesy of G. Steiner).

The main advantage of this construction through the transformation in (1.8) is that Dubiner basis is in fact an orthonormal polynomial basis on simplices.

Proposition 1.2 (Orthonormality of 2D Dubiner basis)**.** *$\{\varphi_{i,j}\}_{i+j\leq p}$ is an L^2-orthonormal basis of $\mathbb{P}^p(\hat{K})$, i.e. the space of polynomials of degree at most $p \in \mathbb{N}$ on the reference triangle $\hat{K}$.*

Proof. We just need to perform the computation of the scalar product between two basis functions $\varphi_{i,j}, \varphi_{k,l}$ and prove that it is 1 if and only if $i = k$ and $j = l$, zero otherwise.

$$
\begin{aligned}
&\int_{\hat{K}} \varphi_{i,j}\varphi_{k,l} = \\
&c_{i,j}c_{k,l}\int_{\hat{K}} (1-\eta)^{i+k} P_i^{0,0}\left(\frac{2\xi}{1-\eta}-1\right) P_k^{0,0}\left(\frac{2\xi}{1-\eta}-1\right) P_j^{2i+1,0}(2\eta-1)P_l^{2k+1,0}(2\eta-1).
\end{aligned}
$$

We aim now to compute this integral on $\hat{Q}$ instead of $\hat{K}$. Therefore, we use (1.8) and we compute the determinant of the Jacobian as:

$$
|J(a,b)| = \begin{vmatrix} \frac{1-b}{4} & -\frac{1+a}{4} \\ 0 & \frac{1}{2} \end{vmatrix} = \frac{1-b}{8}.
$$

Hence, the transformation to the square yields

$$
\begin{aligned}
&c_{i,j}c_{k,l}\int_{\hat{Q}} \left(\frac{1-b}{2}\right)^{i+k} P_i^{0,0}(a)P_k^{0,0}(a)P_j^{2i+1,0}(b)P_l^{2k+1,0}(b)\frac{1-b}{8} \\
=&\frac{c_{i,j}c_{k,l}}{2^{i+k+3}}\int_{-1}^{1} P_i^{0,0}(a)P_k^{0,0}(a)\int_{-1}^{1}(1-b)^{i+k+1}P_j^{2i+1,0}(b)P_l^{2k+1,0}(b).
\end{aligned}
$$

Thanks to Proposition 1.1, we can compute the left integral and it reads

$$
\begin{aligned}
&\frac{c_{i,j}c_{k,l}}{2^{i+k+3}}\frac{2}{2i+1}\delta_{i,k}\int_{-1}^{1}(1-b)^{i+k+1}P_j^{2i+1,0}(b)P_l^{2k+1,0}(b) \\
=&\frac{c_{i,j}c_{i,l}}{2^{2i+3}}\frac{2}{2i+1}\delta_{i,k}\int_{-1}^{1}(1-b)^{2i+1}P_j^{2i+1,0}(b)P_l^{2i+1,0}(b).
\end{aligned}
$$

Using again Proposition 1.1, we finally get

$$
\frac{c_{i,j}c_{i,l}}{2^{2i+3}}\frac{2}{2i+1}\delta_{i,k}\frac{2^{2i+2}}{2j+2i+2}\delta_{j,l} = \frac{c_{i,j}^2}{2(2i+1)(2i+2j+1)}\delta_{i,k}\delta_{j,l} = \delta_{i,k}\delta_{j,l}.
$$

□

1.2.2. Dubiner basis in three dimensions

We now proceed to the extension in three dimensions. In this case, we redefine the reference domains as:

$$
\begin{aligned}
\hat{K} &= \{(\xi,\eta,\nu) : \xi,\eta,\nu \geq 0, \xi+\eta+\nu \leq 1\}, \\
\hat{Q} &= \{(a,b,c) : -1 \leq a,b,c \leq 1\}
\end{aligned} \tag{1.9}
$$

with $\hat{K}$ as the reference tetrahedron and $\hat{Q}$ as the reference cube, also the transformation becomes:

$$\xi = \frac{(1+a)(1-b)(1-c)}{8}, \quad \eta = \frac{(1+b)(1-c)}{4}, \quad \nu = \frac{(1+c)}{2}. \tag{1.10}$$

Definition 1.3 (3D Dubiner basis [14])**.** The 3D Dubiner basis function indexed by $(i,j,k) \in \mathbb{N}^3$ is defined as:

$$\varphi_{i,j,k}(\xi,\eta,\nu) =$$
$$c_{i,j,k} P_i^{0,0}\left(\frac{2\xi}{1-\eta-\nu}-1\right)(1-\eta-\nu)^i P_j^{2i+1,0}\left(\frac{2\eta}{1-\nu}-1\right)(1-\nu)^j P_k^{2i+2j+2,0}(2\nu-1)$$

where $c_{i,j,k} = \sqrt{(2i+2j+2k+3)(2i+2j+2)(2i+1)}$.

Proposition 1.3 (Orthonormality of 3D Dubiner basis)**.** *$\{\varphi_{i,j,k}\}_{i+j+k\leq p}$ is an L^2-orthonormal basis of $\mathbb{P}^p(\hat{K})$, i.e. the space of polynomials of degree $p \in \mathbb{N}$ on the reference tetrahedron $\hat{K}$.*

Proof. We again need to compute the scalar product between two basis functions $\varphi_{i,j,k}, \varphi_{l,m,n}$ and check that it is 1 if and only if the two sets of indices coincide.
In this case, the determinant of the Jacobian is

$$|J(a,b,c)| = \begin{vmatrix} \frac{(1-b)(1-c)}{8} & -\frac{(1+a)(1-c)}{8} & -\frac{(1+a)(1-b)}{8} \\ 0 & \frac{1-c}{4} & -\frac{1+b}{4} \\ 0 & 0 & \frac{1}{2} \end{vmatrix} = \frac{(1-b)(1-c)^2}{64}.$$

Hence, we can directly compute the integral on $\hat{Q}$. In addition, transformation (1.10) and Proposition 1.1 will be again used. So, we have

$$\frac{1}{c_{i,j,k}c_{l,m,n}}\int_{\hat{K}} \varphi_{i,j,k}\varphi_{l,m,n}$$
$$= \int_{\hat{Q}} P_i^{0,0}(a)P_l^{0,0}(a)\frac{((1-b)(1-c))^{i+l}}{4^{i+l}}P_j^{2i+1}(b)P_m^{2l+1}(b)\left(\frac{1-c}{2}\right)^{j+m} P_k^{2i+2j+2,0}(c)P_n^{2l+2m+2,0}(c)J$$
$$= \frac{2\delta_{i,l}}{2i+1}\int_{-1}^{1}\int_{-1}^{1}\frac{((1-b)(1-c))^{i+l}}{4^{i+l}}P_j^{2i+1,0}(b)P_m^{2l+1,0}(b)\left(\frac{1-c}{2}\right)^{j+m} P_k^{2i+2j+2,0}(c)P_n^{2l+2m+2,0}(c)J$$

$$
\begin{aligned}
=&\frac{\delta_{i,l}}{2i+1}\int_{-1}^{1}(1-b)^{2i+1}P_j^{2i+1,0}(b)P_m^{2i+1,0}(b)\int_{-1}^{1}\frac{(1-c)^{2+2i+j+m}}{2^{5+4i+j+m}}P_k^{2i+2j+2,0}(c)P_n^{2i+2m+2,0}(c)\\
=&\frac{\delta_{i,l}}{(2i+1)(2^{5+4i+j+m})}\cdot\frac{2^{2i+2}}{2i+2j+2}\delta_{j,m}\int_{-1}^{1}(1-c)^{2+2i+j+m}P_k^{2i+2j+2,0}(c)P_n^{2i+2m+2,0}(c)\\
=&\frac{\delta_{i,l}\delta_{j,m}}{(2i+1)(2^{3+2i+2j})(2i+2j+2)}\int_{-1}^{1}(1-c)^{2+2i+2j}P_k^{2i+2j+2,0}(c)P_n^{2i+2j+2,0}(c)\\
=&\frac{\delta_{i,l}\delta_{j,m}}{(2i+1)(2^{3+2i+2j})(2i+2j+2)}\cdot\frac{2^{2i+2j+3}}{2i+2j+2k+3}\delta_{k,n}\\
=&\frac{1}{c_{i,j,k}c_{l,m,n}}\delta_{i,l}\delta_{j,m}\delta_{k,n}.
\end{aligned}
$$

□

Remark 1. As stated above, the Dubiner basis and its orthonormality are well-known in literature for a long time ([8], [14], [3]). However, the current state of the art rarely provides explicit definitions (especially, in three dimensions) and these few cases are often referred to different reference tetrahedra and do not lack of oversights. This is the reason why we decided to propose here explicit definitions and proofs of orthonormality, to finally give a complete overview in 2 and 3 dimensions and to justify the code that is implemented in the `DUBValues` class (see Section 2.2.1).

Remark 2. We refer again to [6] for a motivation for the choice of the Dubiner basis in the context of cardiac electrophysiology and, so, to the creation of `DUBeat`.

1.3. From modal to nodal evaluations

The FEM and DGFEM bases have the important property that each basis function is associated to a spatial point (called indeed DOF point) such that the Fourier coefficient of a numerical solution with respect to the i-th basis function coincides with the evaluation of the solution on the i-th DOF point. At this point, it is soon evident that the Dubiner basis satisfies the outstanding ortho-normality property abandoning, however, this point-wise/nodal property.
In this small section, we resume the simple strategies to pass from a nodal/DGFEM representation to a modal/Dubiner representation and how to discretize analytical functions in the two standards. Such steps are necessary when working with modal basis such as Dubiner basis. For instance, the discretization of initial solutions in time-dependent problems requires a strategy to discretize analytical functions as modal evaluations and the generation of contour plots at the end of the numerical resolution requires, instead, a method to convert a solution vector from the modal to the nodal evaluation.
We now recall what we have already been presented in [6] and we start from the transforma-

tion of a discretized solution u_h written as a Dubiner coefficient vector into nodal coefficients. Because of the point-wise property discussed above, to obtain the i-th DGFEM coefficient we just need to evaluate the discretized solution in the DOF point x_i. Keeping in mind the modal meaning of the solution vector components, the nodal evaluation writes

$$\hat{u}_i = u_h(x_i) = \sum_{j=1}^{\tilde{N}_h} \tilde{u}_j \varphi_j(x_i), \qquad i = 1, \dots \hat{N}_h,$$

where $\tilde{N}_h, \hat{N}_h$ are respectively the Dubiner and DGFEM space dimensions, $\tilde{u}_j, \hat{u}_i$ the j-th Dubiner coefficient and the i-th DGFEM coefficient and φ_j the j-th Dubiner basis function.
We now present the general technique to project an analytical function u as a Dubiner function vector. As stated above, Dubiner basis is not nodal but the ortho-normality property turns out to be crucial, indeed thanks to this property, proved in Section 1.2, we can state that the i-th Dubiner coefficient is simply the L^2 scalar product between the function and the i-th basis function. Explicitly:

$$\tilde{u}_j = \int_\Omega u(x)\varphi_j(x)\,dx = \int_{\mathcal{K}_j} u(x)\varphi_j(x)\,dx \qquad j = 1, \dots \tilde{N}_h, \tag{1.11}$$

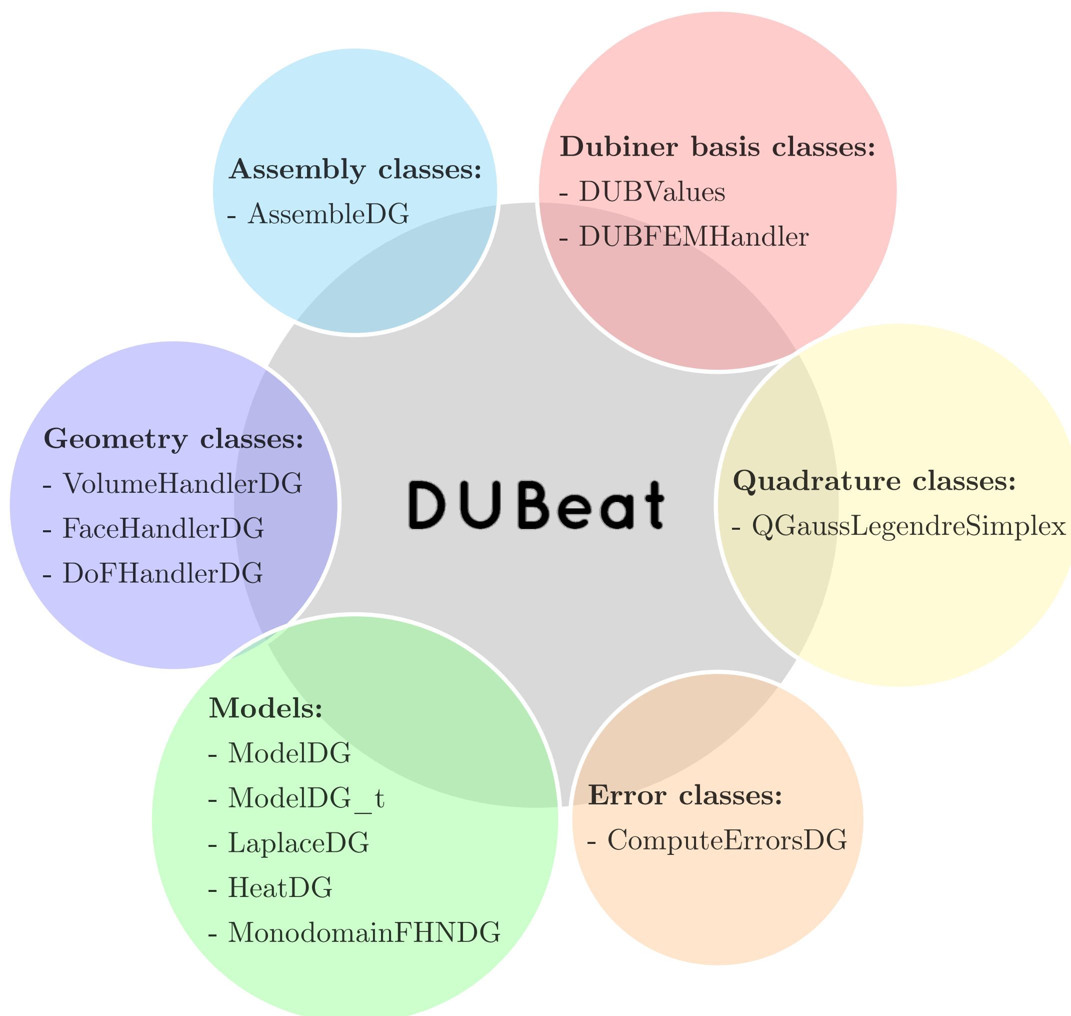
where Ω is the space domain and $\mathcal{K}_j$ is the element that contains the j-th Dubiner function.

Remark 3. The procedure in equation (1.11) can be adopted also to convert FEM vectors into Dubiner vectors. Simply, the u function can be rewritten as linear combination of FEM coefficients and FEM basis functions.

Remark 4. The previous integrals are obviously implemented with quadrature formulas. The `DUBFEMHandler` class contains these operations under the methods: `dubiner_to_fem`, `fem_to_dubiner`, `analytical_to_dubiner` (see Section 2.2.2).

2 | Implementation details

2.1. General structure



`DUBeat` is composed by different template classes that can be discerned into two main groups:

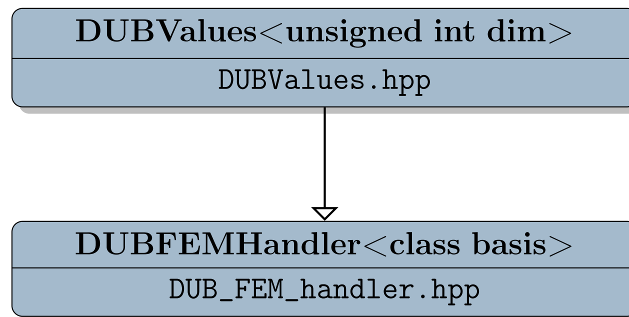
- The *core* classes under the `source` folder that contain the methods for which the discontinous Galerkin methods can effectively work.
- The *model* classes under the `models` folder that correspond to model problems where

DUBeat can be executed and tested (Section 2.7).

The classes could have to be specialized with one of the following parameters:

- <unsigned int dim> is the space dimension, often associated with the lifex configuration lifex::dim. DUBeat works indeed in both 2 and 3 dimensions.
- <class basis> is instead the choice for the basis functions, the user can indeed easily switch between
  - DUBValues<unsigned int dim> for the Dubiner basis.
  - dealii::FE_SimplexDGP<unsigned int dim> for DGFEM basis.

2.2. Dubiner basis classes



2.2.1. DUBValues

The class DUBValues<unsigned int dim> represents the Dubiner basis functions evaluations coherently to the definitions in Section 1. The main routines are the evaluations of the basis functions and their gradients on the reference cell (as defined in equations (1.7), (1.9)). Namely:

- shape_value which evaluates the equation in the Definition 1.3 in a specific point of the reference cell,
- shape_grad which evaluates the gradient of the equation in the Definition 1.3 at a specific point of the reference cell.

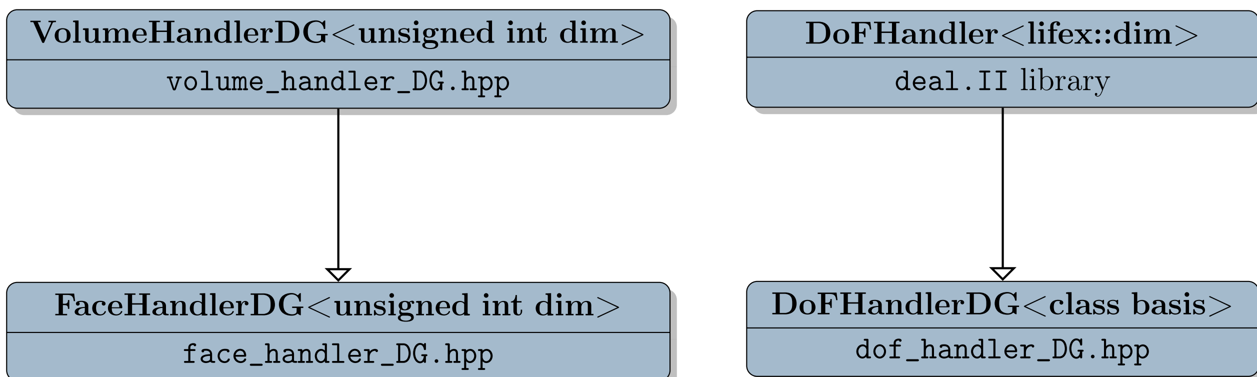
Other useful functions are: eval_jacobi_polynomial, which exploits the equation in the Definition 1.1, and fun_coeff_conversion which represents the conversion from the FEM basis taxonomy (1st basis function, 2nd basis function, $\dots$) to the Dubiner basis taxonomy (where every basis function is indexed by a tuple of indexes (i, j) in 2D and three indexes (i, j, k) in 3D). Although, these functions have only internal scopes to let the basis evaluations possible.

2.2.2. DUBFEMHandler

The class DUBFEMHandler<class basis> inherits from DUBValues<lifex::dim> from Section 2.2.1 and it provides conversions for solution vectors with respect to the DGFEM basis to vectors with respect to the Dubiner basis and viceversa, as explained in Section 1.3. The main functions are:

- analytical_to_dubiner to project an analytical solution as a vector of Dubiner coefficients, to be used for instance with analytical initial solutions before the system solving in time-dependent problems.
- dubiner_to_fem to convert Dubiner solution vectors into DGFEM solution vectors, to be used for instance when creating contour plots at the end of the simulation.
- fem_to_dubiner to do the same in the opposite direction, it might be useful if the initial solution in a time-dependent problem is not analytical but only a vector of evaluations in the DOF points.

2.3. Geometry classes



2.3.1. VolumeHandlerDG

The class VolumeHandlerDG<unsigned int dim> is used to accomplish the main operations on a discontinuous Galerkin volume element. It has to be associated with a domain cell (using the reinit method, it can be reinitialized on a different cell) and performs some element-wise operations such as:

- quadrature_real(const unsigned int q): returns the q-th spatial quadrature point position on the actual element,
- quadrature_ref(const unsigned int q): returns the q-th spatial quadrature point

position on the reference element,

- quadrature_weight(const unsigned int q): returns the quadrature weight associated to the correspondent q-th quadrature node,
- get_jacobian_inverse(): returns the inverse of the Jacobian of the reference-to-actual transformation.

2.3.2. FaceHandlerDG

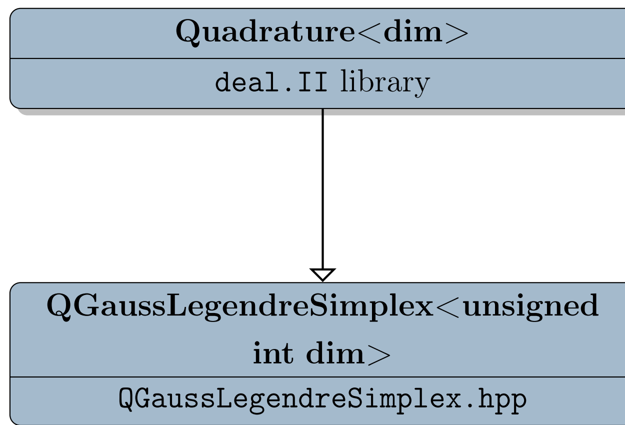
The class FaceHandlerDG <unsigned int dim> is instead useful for the main operations on the faces of a discontinuous Galerkin element. It inherits from the VolumeHandlerDG<unsigned int dim> class in Section 2.3.1 and many methods work similarly also because the template specialization permits to easily decrease the spatial dimension by 1. On the other hand, the reinit method associates the class not only to an element but also to a face and, furthermore, some methods are added:

- corresponding_neigh_index(const unsigned int q): to obtain the associated quadrature point index of the neighbor element on the shared face,
- get_normal(): outward normal vector on the current element and face,
- get_measure(): returns the measure of the face.

2.3.3. DoFHandlerDG

The class DoFHandlerDG<class basis> is used to work with global and local degrees of freedom and their mapping. DUBeat exploits this class instead of the DoFHandler<lifex::dim> class from deal.II because the latter, at the moment, cannot distribute DOFs on tetrahedra with polynomial orders greater than 2. When using the Dubiner basis, this implementation permits overcoming this issue thanks to the use of an internal dof_map and the definition of the basis of every order (even $>$2) from DUBValues (Section 2.2.1). On the other hand, it is not possible to do the same with DGFEM case because the basis functions come directly from the deal.II FiniteElement classes. Consequently, the Dubiner basis can be used with every order, while DGFEM with order at most 2.
Concretely, most of the methods override from the base class while the key feature is the overwriting of the distribute_dofs method that, for the DGFEM basis, it works as in the base class (with the restriction of order ≤ 2) and, for Dubiner basis, it exploits the internal dof_map and, so, permits higher orders.

2.4. Quadrature classes

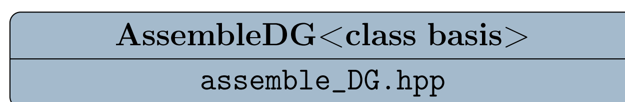


2.4.1. QGaussLegendreSimplex

The class `QGaussLegendreSimplex<unsigned int dim>` represents the Gauss-Legendre quadrature formula on simplex elements. This quadrature formula is not implemented yet in `deal.II` and, for this reason, `DUBeat` adds it to the `deal.II` quadrature methods inheriting from the `Quadrature<unsigned int dim>` class.
The conceptual idea is the following:

1. Computation of the Gauss-Legendre nodes and weights on a given 1-dimensional interval, this is done by the external method `gauleg(const double a,const double b, const unsigned n_points)`,
2. Tensorization of the computed 1D nodes to the 2D or 3D domains depending on the template specialization.
3. Transformation from the square domain to the simplex domain (see equations (1.8), (1.10)).

2.5. Assembly classes



2.5.1. AssembleDG

The class `AssembleDG<class basis>` is used to construct the main local matrices for discontinuous Galerkin methods from Section 1.1. As for `VolumeHandlerDG` and `FaceHandlerDG`, it

exploits a `reinit` method in order to be associated to a local element and/or face. Then, different methods can be used to compute local matrices on that cell/face. Consider the following notation:

- $\mathcal{K}$ is the volume of the associated element,
- $\mathcal{F}$ is the associated face,
- φ_i is the i-th basis function from the associated element,
- the $+$ superscript indicates that the basis function evaluated on the face is the one from the same associated element,
- the $-$ superscript indicates that a basis function evaluated on the face is the one from the neighbor element,
- n^+, n^- are the outward and inward normal vectors with respect to the associated face.

Then, the main methods can be summarised in the following list.

- `local_V`: assembly of local stiffness matrix $V(i,j) = \int_{\mathcal{K}} \nabla\varphi_j \cdot \nabla\varphi_i \, dx$.
- `local_M`: assembly of local mass matrix $M(i,j) = \int_{\mathcal{K}} \varphi_j \varphi_i \, dx$.
- `local_rhs`: assembly of the local forcing term $\mathbf{f}(i) = \int_{\mathcal{K}} f \varphi_i \, dx$, where f is the known forcing term of the problem.
- `local_rhs_edge_dirichlet`: assembly of the local matrix associated to non-homogeneous Dirichlet boundary conditions $\mathbf{f}_{\text{dir}}(i) = \int_{\mathcal{F}} \gamma u \varphi_i^+ - \theta u \nabla\varphi_i^+ \cdot n \, ds$, where γ is the regularity coefficient, θ is the penalty coefficient and u the known solution of the problem.
- `local_rhs_edge_neumann`: assembly of the right hand side term associated to non-homogeneous Neumann boundary conditions $\mathbf{f}_{\text{neu}}(i) = \int_{\mathcal{F}} \varphi_i g \, ds$, where g is the gradient of the known solution of the problem.
- `local_SC`: assembly of the component of the local matrix S (see equation (1.4)) that is evaluated on the same side of the edge, the method returns $SC(i,j) = \int_{\mathcal{F}} \gamma \varphi_j^+ \varphi_i^+ \, ds$, where γ is the regularity coefficient.
- `local_SN`: assembly of the component of the local matrix S that is evaluated on the two sides of the edge, the method returns $SN(i,j) = -\int_{\mathcal{F}} \gamma \varphi_j^+ \varphi_i^- \, ds$, where γ is the regularity coefficient
- `local_IC`: assembly of the component of the local matrix I that is evaluated on the same side of the edge, the method returns $\theta \cdot IC(i,j) = -\frac{\theta}{2} \int_{\mathcal{F}} \nabla\varphi_i^+ \cdot n^+ \varphi_j^+ \, ds$ and $IC^T(i,j) = -\frac{1}{2} \int_{\mathcal{F}} \nabla\varphi_j^+ \cdot n^+ \varphi_i^+ \, ds$, where θ is the penalty coefficient.

- `local_IB`: assembly of the component of the local matrix I that is evaluated on the boundary edges, the method returns $\theta \cdot IB(i,j) = -\,\theta \int_{\mathcal{F}} \nabla\varphi_i^+ \cdot n^+ \varphi_j^+ \, ds$ and $IB^T(i,j) = -\int_{\mathcal{F}} \nabla\varphi_j^+ \cdot n^+ \varphi_i^+ \, ds$, where θ is the penalty coefficient.
- `local_IN`: assembly of the component of the local matrix I that is evaluated on the two sides of the edge, the method returns $\theta \cdot IN(i,j) = \frac{\theta}{2} \int_{\mathcal{F}} \nabla\varphi_i^+ \cdot n^+ \varphi_j^- \, ds$ and $IN^T(i,j) = \frac{1}{2} \int_{\mathcal{F}} \nabla\varphi_j^+ \cdot n^+ \varphi_i^- \, ds$, where θ is the penalty coefficient.
- `local_u0_M_rhs`: assembly of the right hand side term associated to the previous time-step solution in time-dependent problems $F(i) = \int_{\mathcal{K}} u^n \varphi_i \, dx$, where u^n is the solution at a generic previous step n.

In addition to the core local matrices, there are also some specific definitions that are used for particular models such as `local_w0_M_rhs` and `local_non_linear_fitzhugh`.
Note that the matrices I and S from Section 1.1 correspond to:

- $-\theta \cdot I = \sum_{F \in \mathcal{F}_h^I}$ `local_IC.first` $+ \sum_{F \in \mathcal{F}_h^I}$ `local_IN.first` $+ \sum_{F \in \mathcal{F}_h^B}$ `local_IB.first`,
- S$=\sum_{F \in \mathcal{F}_h}$ `local_S` $+ \sum_{F \in \mathcal{F}_h^I}$ `local_SN`,

where we indicate with `.first` the first matrix of the returned tuple.

2.6. Error classes

ComputeErrorsDG<class basis>
compute_errors_DG.hpp

2.6.1. ComputeErrorsDG

The class `ComputeErrorsDG<class basis>` computes the errors between the numerical solution and the exact solution. Four definitions of error are implemented:

- `compute_error_inf`: $\|u - u_h\|_{L^\infty(\Omega)} := \sup_{x\in\Omega} |u(x) - u_h(x)|$,
- `compute_error_L2`: $\|u - u_h\|^2_{L^2(\Omega)} := \int_\Omega |u(x) - u_h(x)|^2 \, dx$,
- `compute_error_H1`: $\|u - u_h\|^2_{H^1(\Omega)} := \|u - u_h\|^2_{L^2(\Omega)} + \|\nabla u - \nabla u_h\|^2_{L^2(\Omega)}$,
- `compute_error_DG`: $\|u - u_h\|^2_{DG(\Omega)} := \|\nabla u - \nabla u_h\|^2_{L^2(\Omega)} + \gamma \|[\![u - \nabla u_h]\!]\|^2_{L^2(\mathcal{F})}$.

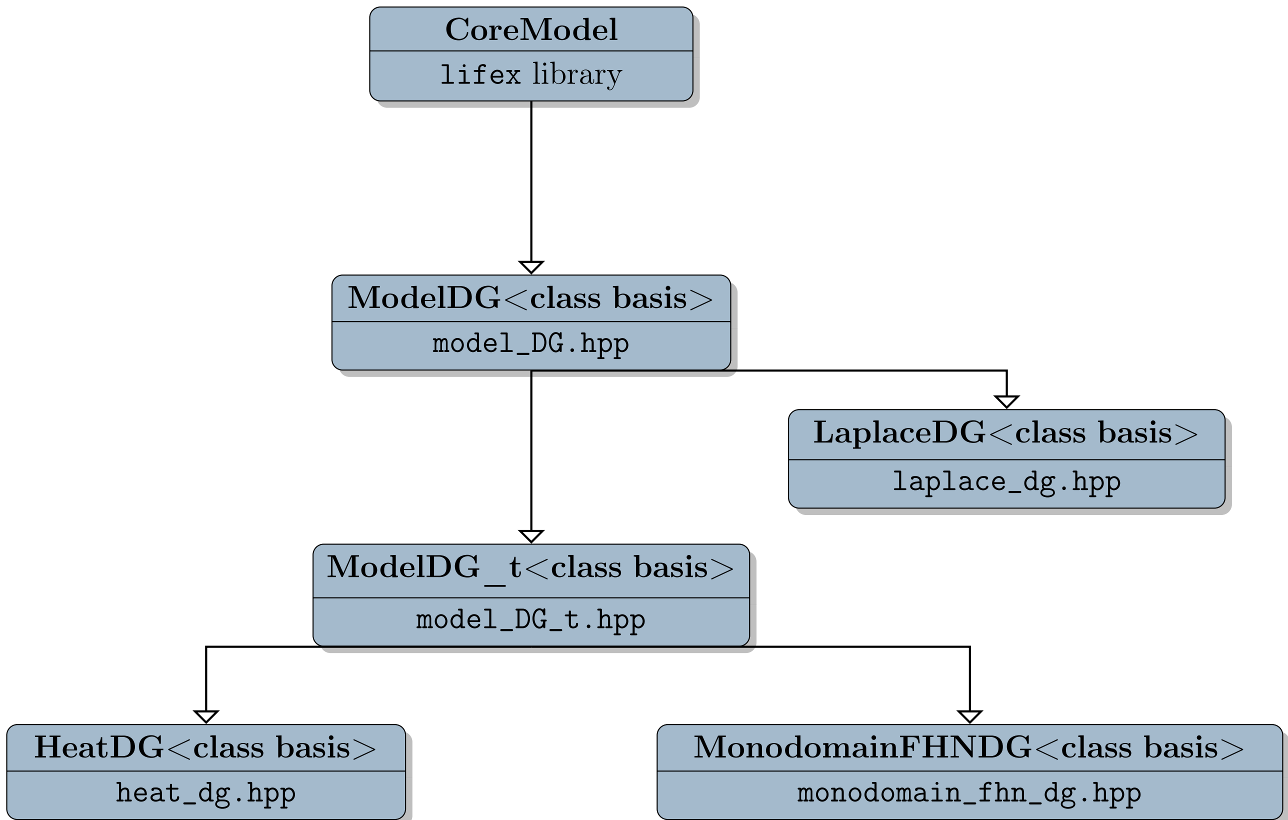
where:

- u is the exact solution,

- u_h is the numerical solution,
- The $L^2(\mathcal{F})$ norm is computed on the faces instead of the volume.
- All the other terms follow the nomenclature of Section 1.1.

Furthermore, the class can create and update a datafile containing the errors from different simulations that have been previously run with different polynomial orders and grid refinements. This is a `.data` file and it is created by the `initialize_datafile` method and updated by the `update_datafile` method.

2.7. Models



As anticipated, `DUBeat` provides some models that one can use to test and validate the library. These classes can be split into three groups:

- `ModelDG` and `ModelDG_t` are not proper models, but instead templates that are helpful to define models, indeed they are abstract classes. They are placed under the `source` folder as the core classes.
- `LaplaceDG` and `HeatDG` are example problems to test, respectively, stationary problems and time-dependent problems.

- `MonodomainFHNDG` is the first electrophysiology problem in `DUBeat`. Other problems are aimed to be added in a future release.

2.7.1. ModelDG

The class `ModelDG<class basis>` represents a template for the resolution of problems using discontinuous Galerkin methods.
The main method is `run` which indeed calls in sequence all the steps that are necessary for a complete simulation:

1. `create_mesh`: loading of the mesh (picking the default mesh files from the `meshes` folder or from a user-defined path),
2. `setup_system`: setup the problem with its objects and variables before the resolution,
3. `initialize_solution`: initialize the vectors for the exact and numerical solutions,
4. `discretize_analytical_solution`: projection of the analytical exact solution to basis coefficients, needed for the computation of some errors. Note that this operation is template-specialized depending on the basis used.
5. `assemble_system`: assembly of the linear system using the `AssembleDG` class. It is characteristic of the specific problem and, therefore, it is pure virtual.
6. `solve_system`: resolution of the algebraic system,
7. `compute_errors`: compute and output of the errors using the `ComputeErrorsDG` class.
8. `output_results`: output the numerical and exact solution contour plots as files that can be viewed using `ParaView`. Note that the generation of the graphical output is possible only if the polynomial order is strictly less than 3 because it requires a conversion to the DGFEM discretization. Read Section 2.3.3 for the full explanation.

2.7.2. ModelDG_t

The class `ModelDG_t<class basis>` inherits from `ModelDG<class basis>` (Section 2.7.1) and represents the resolution of time-dependent problems using discontinuous Galerkin methods. The method `run` is overridden since it performs a loop over the time steps, calling for each iteration `assemble_system` and `solve_system`. Furthermore, new methods are added because they are necessary for this kind of problem:

- `time_initialization`: setup for time-dependent problems at the initial time,

- `update_time`: to perform the time increment,
- `intermediate_error_print`: computation of the errors at an intermediate time step.

Moreover, this class introduces the BDF time advancing schemes through the `lifex` class `BDFHandler` and it is possible for the user to set the order for the BDF scheme.

2.7.3. LaplaceDG

The class `LaplaceDG<class basis>` solves the Laplace equation with Dirichlet boundary condition:

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = u_{\text{ex}} & \text{on } \partial\Omega. \end{cases}$$

Adopting a discontinuous Galerkin method, the weak formulation of the problem has been already shown in equation (1.5) and the correspondent linear system (using the matrices defined in Section 1.1) is:

$$(V - I^T - \theta I + S)\mathbf{u} = \mathbf{f} + \mathbf{f}_{\text{dir}}$$

where:

$$\mathbf{f}_{\text{dir}_i} = -\theta \sum_{F \in \mathcal{F}_h^B} \int_F u_{\text{ex}} \nabla_h \varphi_i \cdot \mathbf{n} + \sum_{F \in \mathcal{F}_h^B} \int_F \gamma u_{\text{ex}} \varphi_i$$

It can be solved using the FEM basis (`basis=dealii::FE_SimplexDGP<lifex::dim>`) or the Dubiner basis (`basis=DUBValues<lifex::dim>`). Boundary conditions and forcing terms are assigned knowing that the exact solution is:

$$\begin{aligned} d = 2: \; & u_{\text{ex}}(x, y) = e^{x+y}, & (x, y) \in \Omega = (1, 1)^2, \\ d = 3: \; & u_{\text{ex}}(x, y, z) = e^{x+y+z}, & (x, y, z) \in \Omega = (1, 1)^3. \end{aligned}$$

2.7.4. HeatDG

The class `HeatDG<class basis>` solves the heat equation with Neumann boundary condition:

$$\begin{cases} \frac{\partial u}{\partial t} - \Delta u = f & \text{in } \Omega \times t \in (t_{start}, t_{end}), \\ \nabla u \cdot \mathbf{n} = g & \text{on } \partial\Omega \times t \in (t_{start}, t_{end}), \\ u = u_{\text{ex}} & \text{in } \Omega \times t = t_{start}. \end{cases}$$

Adopting a discontinuous Galerkin method, the weak formulation of the model is the following:

$$\text{Find } u_h \in V_h \quad \text{such that } \forall t \in (t_{start}, t_{end}) :$$
$$\sum_{\mathcal{K}\in\mathcal{T}_h} \int_{\mathcal{K}} \frac{\partial u_h}{\partial t} v_h + \sum_{\mathcal{K}\in\mathcal{T}_h} \int_{\mathcal{K}} \nabla u_h \cdot \nabla v_h - \sum_{F\in\mathcal{F}_h^I} \int_F \{\{\nabla_h u_h\}\} \cdot [\![v_h]\!] - \theta \sum_{F\in\mathcal{F}_h^I} \int_F [\![u_h]\!] \cdot \{\{\nabla_h v_h\}\}$$
$$+ \sum_{F\in\mathcal{F}_h^I} \int_F \gamma [\![u_h]\!] \cdot [\![v_h]\!] = \int_\Omega f v_h + \sum_{F\in\mathcal{F}_h^B} \int_F g v_h \qquad \forall v_h \in V_h,$$

Being a time-dependent model (indeed this class is inherited from `ModelDG_t<class basis>`) we have to discretize the interval (t_{start}, t_{end}) in N_t intervals of length Δt (time step), such that the weak formulation with an explicit Euler scheme for the time discretization becomes:

Given $u_h^0 = u_{\text{ex}}$, find $u_h^{n+1} \in V_h$ such that $\forall n \in \{0, \dots, N_t - 1\}$:

$$\sum_{\mathcal{K}\in\mathcal{T}_h} \int_{\mathcal{K}} \frac{u_h^{n+1} - u_h^n}{\Delta t} v_h + \sum_{\mathcal{K}\in\mathcal{T}_h} \int_{\mathcal{K}} \nabla u_h^{n+1} \cdot \nabla v_h - \sum_{F\in\mathcal{F}_h^I} \int_F \{\{\nabla_h u_h^{n+1}\}\} \cdot [\![v_h]\!]$$
$$-\theta \sum_{F\in\mathcal{F}_h^I} \int_F [\![u_h^{n+1}]\!] \cdot \{\{\nabla_h v_h\}\} + \sum_{F\in\mathcal{F}_h^I} \int_F \gamma [\![u_h^{n+1}]\!] \cdot [\![v_h]\!] = \int_\Omega f^{n+1} v_h + \sum_{F\in\mathcal{F}_h^B} \int_F g^{n+1} v_h \qquad \forall v_h \in V_h,$$

Consequently, the linear system to solve at each time step is:

$$\left(\frac{1}{\Delta t} M + V - I^T - \theta I + S\right) \mathbf{u}^{n+1} = \mathbf{f}^{n+1} + \mathbf{f}_{\text{neu}}^{n+1} + \frac{1}{\Delta t} M \mathbf{u}^n$$

where:

$$M_{i,j} = \sum_{\mathcal{K}\in\mathcal{T}_h} \int_{\mathcal{K}} \varphi_i \varphi_j$$

$$\mathbf{f}_{\text{neu}_i}^n = \sum_{F\in\mathcal{F}_h^B} \int_F g^n \varphi_i$$

It can be solved using the FEM basis (`basis=dealii::FE_SimplexDGP<lifex::dim>`) or the Dubiner basis (`basis=DUBValues<lifex::dim>`). Boundary conditions and forcing terms are assigned knowing that the exact solution is:

$$d = 2: \; u_{\text{ex}}(x, y, t) = \sin(2\pi x)\sin(2\pi y)e^{-5t}, \qquad \Omega = (1, 1)^2,$$
$$d = 3: \; u_{\text{ex}}(x, y, z, t) = \sin\left(2\pi x + \frac{\pi}{4}\right)\sin\left(2\pi y + \frac{\pi}{4}\right)\sin\left(2\pi z + \frac{\pi}{4}\right)e^{-5t}, \qquad \Omega = (1, 1)^3.$$

and $t \in [0, T]$, where T is specified in the parameter file (see Section 3.3.1).

2.7.5. MonodomainFHNDG

The class `MonodomainFHNDG<class basis>` solves the monodomain equation coupled with the Fitzhugh-Nagumo ionic model and Neumann boundary conditions. We refer to [12] and [7] for a description of the monodomain problem from the cardiac electrophysiology. The choice of the Fitzhugh-Nagumo model is mainly due to its performance and simplicity. This model was originally proposed in [13] and a DG formulation of a such problem is already present in [2]. The system reads:

$$
\begin{cases}
\chi_m C_m \frac{\partial V_m}{\partial t} - \nabla \cdot (\Sigma \nabla V_m) + \chi_m I_{ion}(V_m, \omega) = I_{ext} & \text{in } \Omega \times (t_{start}, t_{end}), \\
I_{ion}(V_m, \omega) = k V_m (V_m - a)(V_m - 1) + \omega, & \text{in } \Omega \times (t_{start}, t_{end}), \\
\frac{\partial \omega}{\partial t} = \epsilon (V_m - \Gamma \omega) & \text{in } \Omega \times (t_{start}, t_{end}), \\
\Sigma \nabla V_m \cdot n = g & \text{on } \partial\Omega \times (t_{start}, t_{end}), \\
V_m = V_{m_{\text{ex}}} & \text{in } \Omega \times t = t_{start}, \\
\omega = \omega_{\text{ex}} & \text{in } \Omega \times t = t_{start}.
\end{cases}
$$

where:

- V_m is the trans-membrane potential,
- ω is the gating variable,
- Σ is a known positive definite tensor that represents the non-isotropic conductivity,
- χ_m, C_m are known positive constants of the general monodomain problem,
- k, a, ϵ, Γ are known constants of the Fitzhugh-Nagumo ionic model,
- I_{ext} is the ionic current,
- g is the Neumann boundary condition data for V_m,
- $V_{m_{\text{ex}}}, \omega_{\text{ex}}$ are the initial conditions for V_m, ω respectively.

For a complete explanation of these terms, we again refer to [12].

Solving this problem with the discontinuous Galerkin method, we obtain the following weak formulation:

For any $t \in (t_{start}, t_{end})$ find $V_m^h, \omega_h \in V_h$ such that:

$$\begin{cases} \displaystyle\sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \chi_m C_m \frac{\partial V_m^h}{\partial t} v_h + \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \Sigma\nabla V_m^h \cdot \nabla v_h - \sum_{F\in\mathcal{F}_h^I}\int_F \{\{\nabla_h V_m^h\}\} \cdot [\![v_h]\!] \\ \displaystyle -\theta \sum_{F\in\mathcal{F}_h^I}\int_F [\![V_m^h]\!] \cdot \{\{\nabla_h v_h\}\} + \sum_{F\in\mathcal{F}_h^I}\int_F \gamma [\![V_m^h]\!] \cdot [\![v_h]\!] + \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \chi_m k (V_m^h - 1)(V_m^h - a) V_m^h \\ \displaystyle + \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \chi_m \omega_h v_h = \int_\Omega I_{ext} v_h + \sum_{F\in\mathcal{F}_h^B}\int_F g v_h \qquad \forall v_h \in V_h, \\ \\ \displaystyle\sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \frac{\partial \omega_h}{\partial t} v_h = \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \epsilon (V_m^h - \Gamma\omega_h) v_h \qquad \forall v_h \in V_h. \end{cases}$$

As for the `HeatDG<class basis>` class, this system represents a time-dependent model that needs to be further discretized in time. We split the interval (t_{start}, t_{end}) into N_t intervals of length Δt (i.e. the time step), such that the weak formulation coupled with an explicit Euler scheme for the time discretization becomes:

Given $V_m^0 = V_{m_{\text{ex}}}, \omega^0 = \omega_{\text{ex}}$, find $V_m^{n+1}, \omega^{n+1} \in V_h$ such that $\forall n = 0, \ldots, N_t - 1$:

$$\begin{cases} \displaystyle\sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \chi_m C_m \frac{V_h^{n+1} - V_m^n}{\Delta t} v_h + \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \Sigma\nabla V_m^{n+1} \cdot \nabla v_h - \sum_{F\in\mathcal{F}_h^I}\int_F \{\{\nabla_h V_m^{n+1}\}\} \cdot [\![v_h]\!] \\ \displaystyle -\theta \sum_{F\in\mathcal{F}_h^I}\int_F [\![V_m^{n+1}]\!] \cdot \{\{\nabla_h v_h\}\} + \sum_{F\in\mathcal{F}_h^I}\int_F \gamma [\![V_m^{n+1}]\!] \cdot [\![v_h]\!] + \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \chi_m k (V_m^n - 1)(V_m^n - a) V_m^{n+1} \\ \displaystyle + \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \chi_m \omega^{n+1} v_h = \int_\Omega I_{ext}^{n+1} v_h + \sum_{F\in\mathcal{F}_h^B}\int_F g^{n+1} v_h \qquad \forall v_h \in V_h, \\ \\ \displaystyle\sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \frac{\omega^{n+1} - \omega^n}{\Delta t} v_h = \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} \epsilon (V_m^n - \Gamma\omega^{n+1}) v_h \qquad \forall v_h \in V_h. \end{cases}$$

Hence, the linear system to solve at each time step is:

$$\begin{cases} (\frac{1}{\Delta t} + \epsilon\Gamma) M \omega^{n+1} = \epsilon M V_m^n + \frac{M}{\Delta t}\omega^n \quad \Leftrightarrow \quad (\frac{1}{\Delta t} + \epsilon\Gamma)\omega^{n+1} = \epsilon V_m^n + \frac{1}{\Delta t}\omega^n, \\ (\chi_m C_m M + \Sigma V - I^T - \theta I + S + \chi_m k C(V_m^n)) V_m^{n+1} = f^{n+1} + f_{neu}^{n+1} + \chi_m C_m M V_m^n + \chi_m M \omega^{n+1} \end{cases}$$

where:

$$C(V_m^n)_{i,j} = \sum_{\mathcal{K}\in\mathcal{T}_h}\int_{\mathcal{K}} (V_m^n - 1)(V_m^n - a)\varphi_i\varphi_j. \tag{2.1}$$

As usual, it can be solved using the FEM basis (`basis=dealii::FE_SimplexDGP<lifex::dim>`) or the Dubiner basis (`basis=DUBValues<lifex::dim>`). Moreover, the forcing terms and

boundary conditions are assigned knowning that the exact solutions are:

$$\begin{aligned}
d = 2 : V_{m_{\text{ex}}}(x,y) &= \sin(2\pi x)\sin(2\pi y)e^{-5t}, && \Omega = (1,1)^2,\\
w_{\text{ex}}(x,y) &= \frac{\epsilon}{\epsilon\cdot\Gamma - 5}\sin(2\pi x)\sin(2\pi y)e^{-5t}, && \Omega = (1,1)^2,\\
d = 3 : V_{m_{\text{ex}}}(x,y,z) &= \sin\left(2\pi x + \frac{\pi}{4}\right)\sin\left(2\pi y + \frac{\pi}{4}\right)\sin\left(2\pi z + \frac{\pi}{4}\right)e^{-5t}, && \Omega = (1,1)^3,\\
w_{\text{ex}}(x,y,z) &= \frac{\epsilon}{\epsilon\cdot\Gamma - 5}\sin(2\pi x)\sin(2\pi y)\sin(2\pi z)e^{-5t}, && \Omega = (1,1)^3.
\end{aligned}$$

and $t \in [0,T]$, where T, as well as the monodomain and Fitzhugh-Nagumo parameters, are specified in the parameter file (see Section 3.3.1).

Remark 5. The linear system of this model requires the definitions of two new matrices that are not part of the standard core matrices from Section 2.5.1 and, therefore, they have been added to the `AssembleDG` class methods:

- `local_non_linear_fitzhugh`: assembly of the equation (2.1) on the local element $\mathcal{K}$,
- `local_w0_M_rhs`: assembly of the right hand side term associated with the previous time-step gating variable solution, $F(i) = \int_{\mathcal{K}} w^n \varphi_i \, dx$ where w^n is the gating variable solution at a generic previous step n.

Remark 6. The monodomain problem requires the resolution of a system with two unknown functions. This fact is in contrast with the `ModelDG` template setting with a single unknown u. Hence, the inherited functions `run`, `update_time` and `time_initialization` are overridden.

3 | Dependencies, installation and execution

In this section, we aim to show step by step how the user can install the open-source library and use it to solve the implemented models but even how to use DUBeat for user-defined problems.

3.1. Dependencies

Before seeing how to install the library, we need to specify all the dependencies that guarantee the complete backend and the right functioning of DUBeat.
First of all, the library can be used only on a linux machine with CMake $\geq$ 3.22.1 and GNU bash $\geq$ 5.1.16.
Then, DUBeat 1.0.0 relies almost exclusively on the lifex 1.5.0 installation and its dependencies. So, as a first step, the user should visit the lifex installation page[1] to verify he satisfies all the requirements and should then install it on the device.
More precisely, the library requires the libraries included in the lifex mk module, 2022.0 version. This module can be downloaded from here[2] and we refer again to the lifex installation page[1] for more information. In particular, DUBeat uses these packages:

- ADOL-C = 2.7.2
- Boost = 1.76.0
- deal.II = 9.3.1
- p4est = 2.3.2
- PETSc = 3.15.1
- TBB = 2021.3.0
- Trilinos = 13.0.1

[1] https://lifex.gitlab.io/lifex/download-and-install.html
[2] https://github.com/elauksap/mk/releases/download/v2022.0/mk-2022.0-lifex.tar.gz

In addition to the core libraries, other packages need to be installed for supplementary reasons:

- Python $\geq$ 3.9.6 for the creation of convergence plot figures after simulations on different grid sizes (see generate_convergence_plots.py).
- Doxygen $\geq$ 1.9.1 for the generation of the library documentation.
- Graphviz $\geq$ 2.43.0 for the automatic creation of figures included in the documentation.
- Clang-Format $\geq$ 14.0.0 for the automatic indentation of the library codes.
- gmsh $\geq$ 4.0.4 for the generation of mesh files.
- ParaView $\geq$ 5.9.1 for the analysis and view of numerical solutions.

3.2. Installation

If the core and supplementary libraries are installed, the user can proceed to install DUBeat:

1. To download the library, move to the directory where you desire to install DUBeat version 1.0.0 and use the following bash command.

   ```
   git clone git@github.com:teocala/DUBeat.git -branch v1.0.0
   ```

   Notice that the previous operation requires the SSH key authentication, see the GitHub dedicated page[3] for more information.

2. DUBeat is only a header library, so what you have just downloaded is already the library installation.

3. Modify the variable LIFEX_PATH in the Makefile.inc file specifying your local lifex directory.

3.3. Execution

3.3.1. How to run the template simulation

/build/main_dubiner_dg.cpp is a template script to run a simulation using one of the provided models (Laplace, heat or monodomain). To execute it, the user has to:

1. Choose the model and basis functions (between DGFEM and Dubiner) modifying the main_dubiner_dg.cpp script.

[3] https://docs.github.com/en/authentication/connecting-to-github-with-ssh

2. Go back to the main directory and run make to compile the script.
3. Now that the execution file is ready, move to the build folder and type

```
./main_dubiner_dg -g
```

   to generate the `main_dubiner_dg.prm` parameter file.
4. You can now set your parameters in this file without the need to re-compile. In particular, you can specify the mesh refinement that consists in the choice of the mesh file from the `meshes` folder.
5. Finally, write

```
./main_dubiner_dg
```

   to run your simulation.

3.3.2. Visualization of the results

Results can be viewed and analyzed in two ways:

- By default, you should see on the screen the numerical errors with the exact solution. In addition, the `ComputeErrorsDG` class (Section 2.6.1) writes the errors on a .data file. Finally, you can use `Python` to run the `/extra/generate_convergence_plots.py` script to create plots of the errors for different mesh refinements starting from a `.data` file.
- The code generates two solution files, open `solution.xdmf` with `ParaView` to see the contour plots of the numerical and exact solutions.

3.4. Documentation, indentation and cleaning

The `Make` configuration permits to easily perform some operations that are suggested to keep a neat and working environment in `DUBeat`, especially for users that want to contribute to the library or personalize their problems (see Section 3.5). The following operations are to be executed on the `linux` command line from the library main folder:

- Run

```
make doc
```

  to generate the `Doxygen` documentation under the `documentation` folder. We remind that the updated documentation can be always found also here.

- Use instead

```
make indent
```

  to automatically indent all the files using our customized `Clang-Format` setup.

- To conclude, you can run

```
make clean
```

  to remove the previously generated execution files. In addition, you can write

```
make distclean
```

  to perform a complete cleaning, i.e. removing all the generated files such as parameter, documentation and solution files.

3.5. Personalize the models to solve

We dedicate this paragraph to describe the necessary steps to implement a user-defined model, i.e. a model that is different from the problems already described in Section 2.7.

- A new model should follow the same structure as the ones in the `models` folder. This implies that the class should inherit from `ModelDG` if it is a stationary problem, while it should derive from `ModelDG_t` if it is a time-dependent problem.
- The only method that must necessarily be implemented is `assemble_system`, which specifies how the linear system of the problem is defined. However, also all the other methods can be overridden based on the problem requests or preferences (see for instance Remark 6).
- The addition of a new type of discontinuous Galerkin local matrix needs to be supplemented to the `assemble_DG.hpp` methods.
- As for now, simplices meshes can not be built runtime in `lifex`. Therefore, some example meshes are provided in the folder `meshes` and used in the default version. In case it is needed to use other `.msh` files, the user has to adopt the version of the `create_mesh` method in `ModelDG` that accepts a user-defined mesh path.
- To conclude, we remind that the addition of new scripts or folders might require new `Make` and `indent` configurations.

4 | Numerical results

This section is aimed to show some numerical results based on the analysis of the errors between numerical and exact solutions to the models presented in Section 2.7. Convergence rates have here a dual goal: from one side, guarantee that `DUBeat` successfully and correctly solves these model problems and, from the other side, show the performance of the high-order discontinuous Galerkin methods. These two aspects will be shown respectively in Sections 4.1 and 4.2.

4.1. Grid refinement convergence tests

The following convergence rates are shown to guarantee that the numerical methods implemented in `DUBeat` converge to simple problem solutions when we refine the mesh. We opt here to focus only on the results related to the Dubiner basis. Convergence tests with DGFEM are very similar and the following analysis is enough to guarantee the correct resolution of the problems. In this setting, we fix a low polynomial order (1 or 2) and we increase gradually the mesh refinement by picking the different mesh files from the `meshes` folder. Keeping a low polynomial degree, we do not benefit much from the high-order methods, however, the performance of the methods is not the objective of these tests and the lighter computation due to small degrees allows the use of more refined meshes.

4.1.1. Results for LaplaceDG

See Section 2.7.3 for more details about the problem. Choosing the Dubiner basis, the model is, therefore, `LaplaceDG<DUBValues<lifex::dim»`. In the next plots, we define $d = 2, 3$ as the spatial dimension, $p \in \mathbb{N}$ is the polynomial order and $h > 0$ is the average element length of the mesh. Results are shown in Figure 4.1, 4.2, 4.3, 4.4.

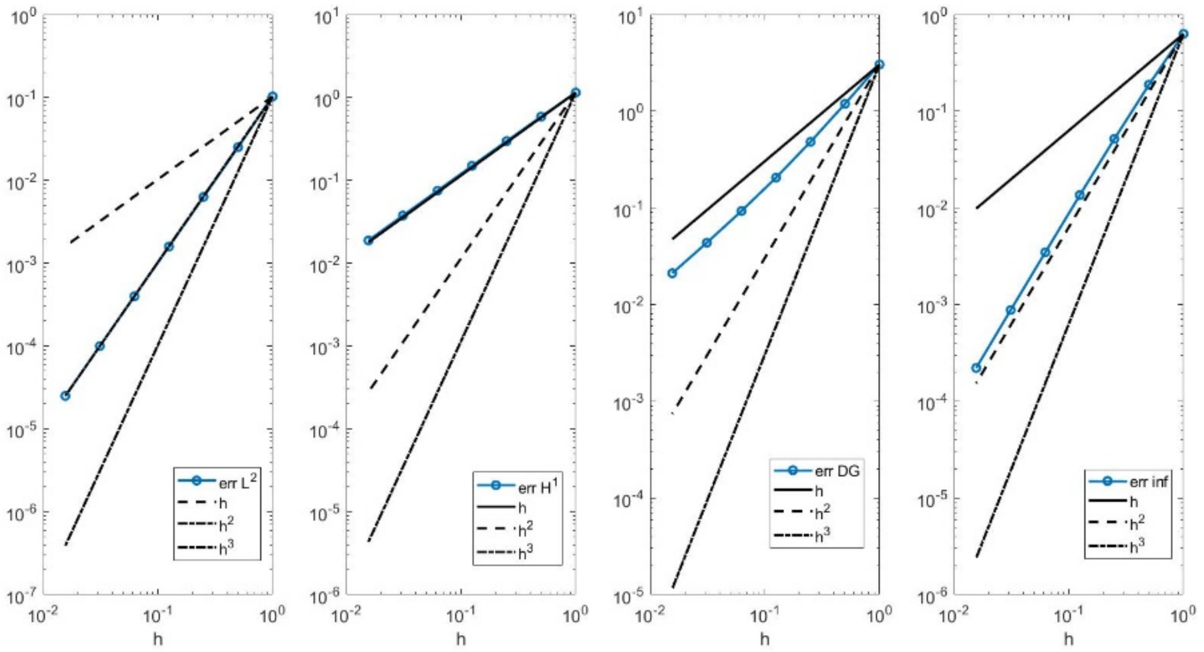


Figure 4.1: LaplaceDG, $d = 2, p = 1$. Expected trends are, in order, h^2, h, h, h^2. The convergence is correct and, for the DG error, even better than the estimated trend.

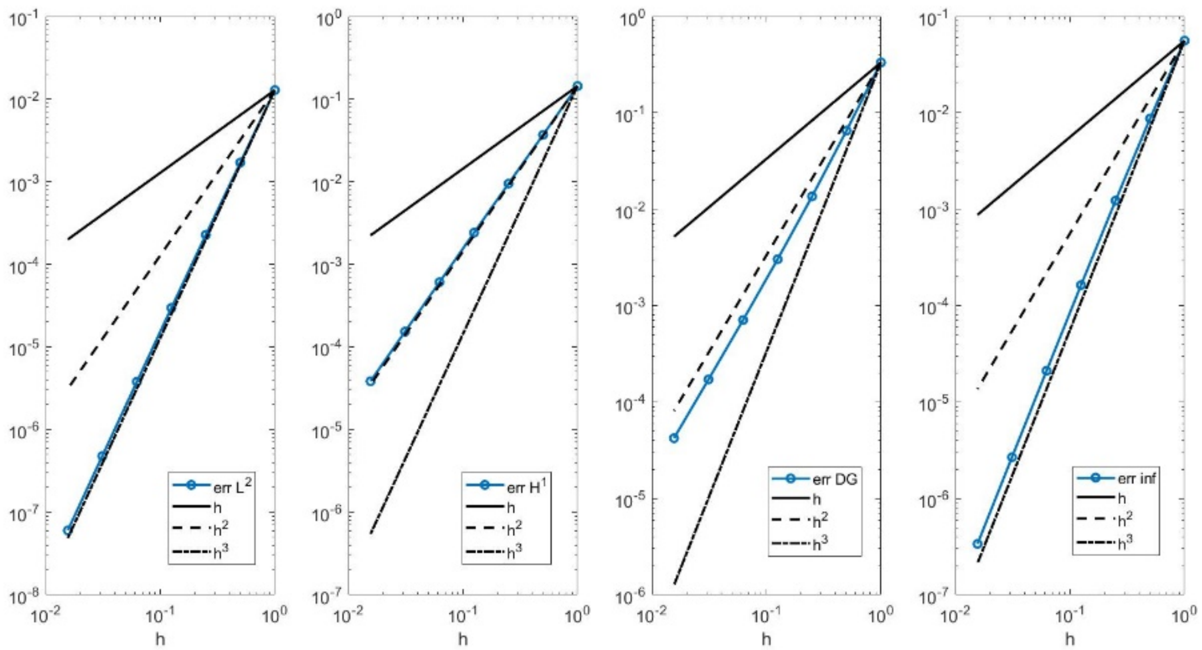


Figure 4.2: LaplaceDG, $d = 2, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. The same comment as in Figure 4.1.

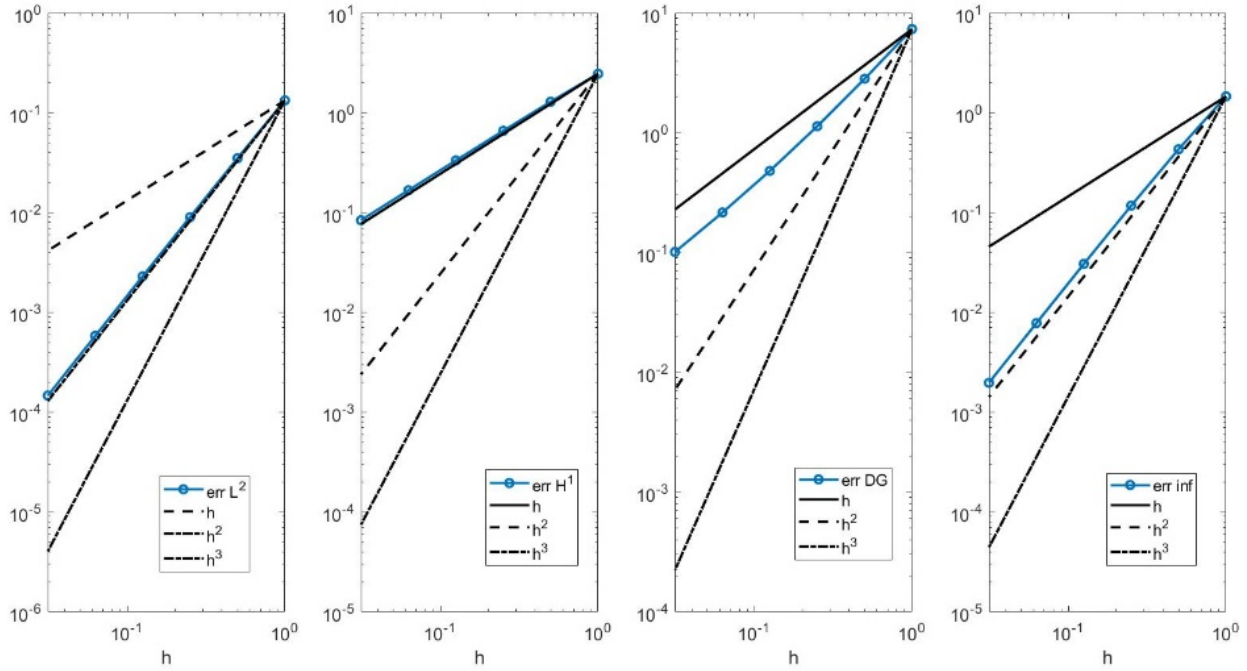


Figure 4.3: LaplaceDG, $d = 3, p = 1$. Expected trends are, in order, h^2, h, h, h^2. The convergence is correct and, for the DG error, even better than the estimated trend.

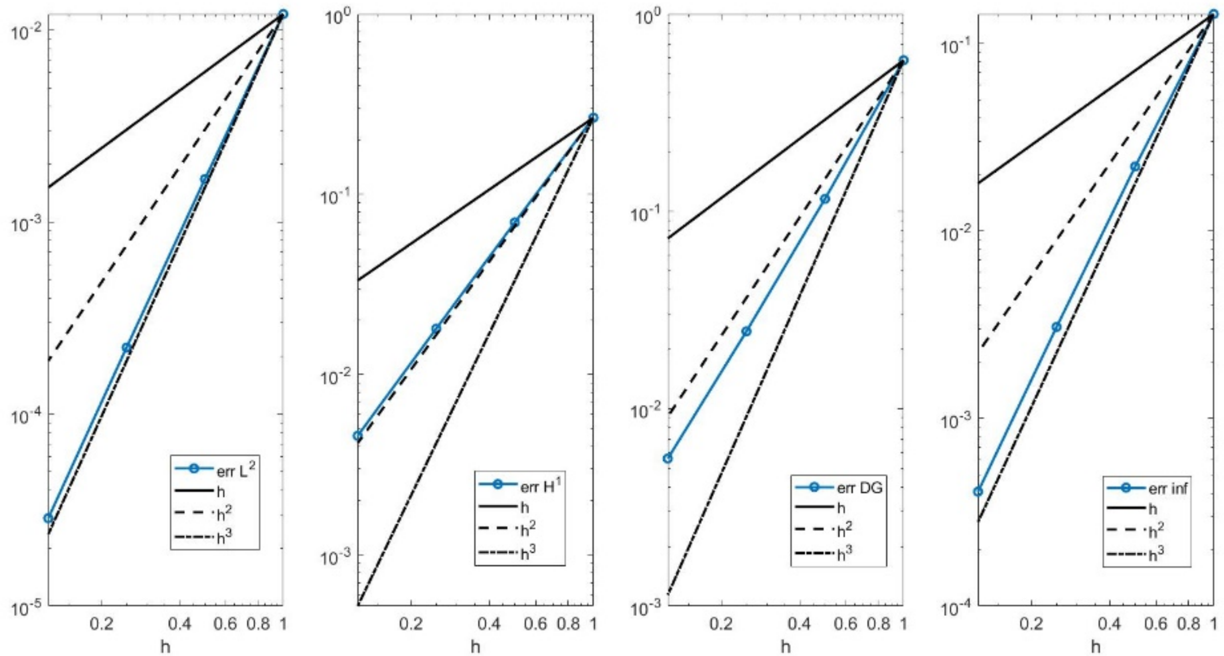


Figure 4.4: LaplaceDG, $d = 3, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. The same comment as in Figure 4.3.

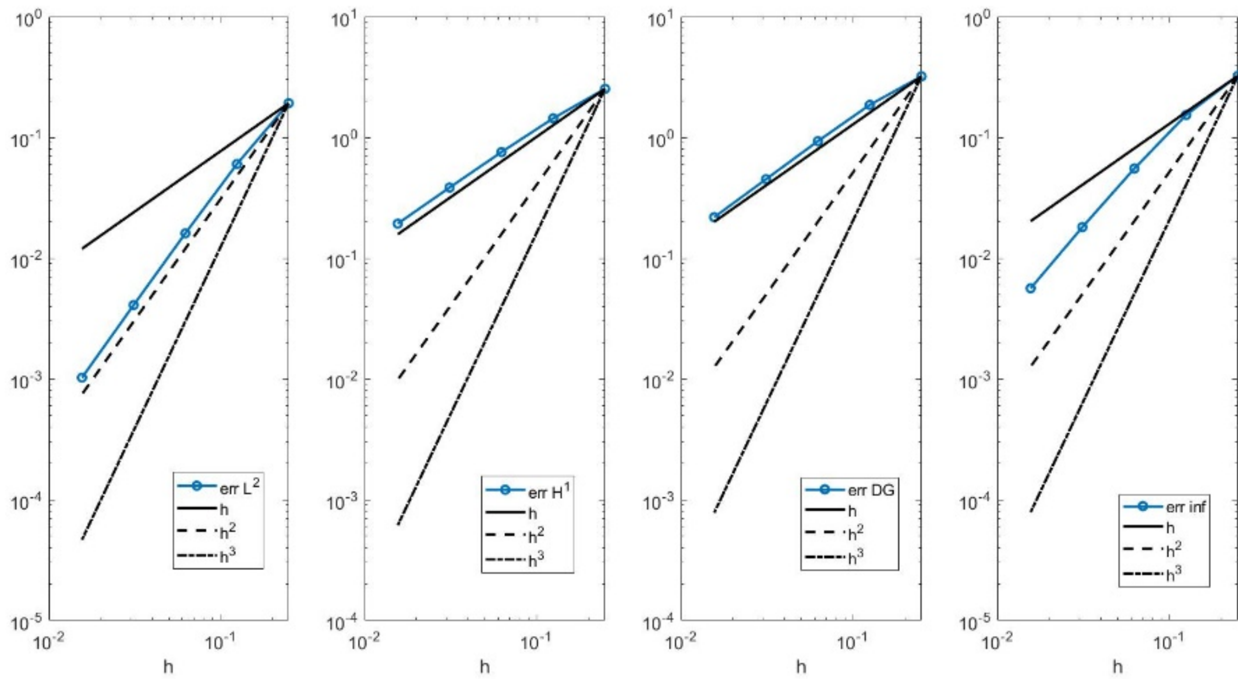


Figure 4.5: HeatDG, $d = 2, p = 1$. Expected trends are h^2, h, h, h^2. The convergence is correct.

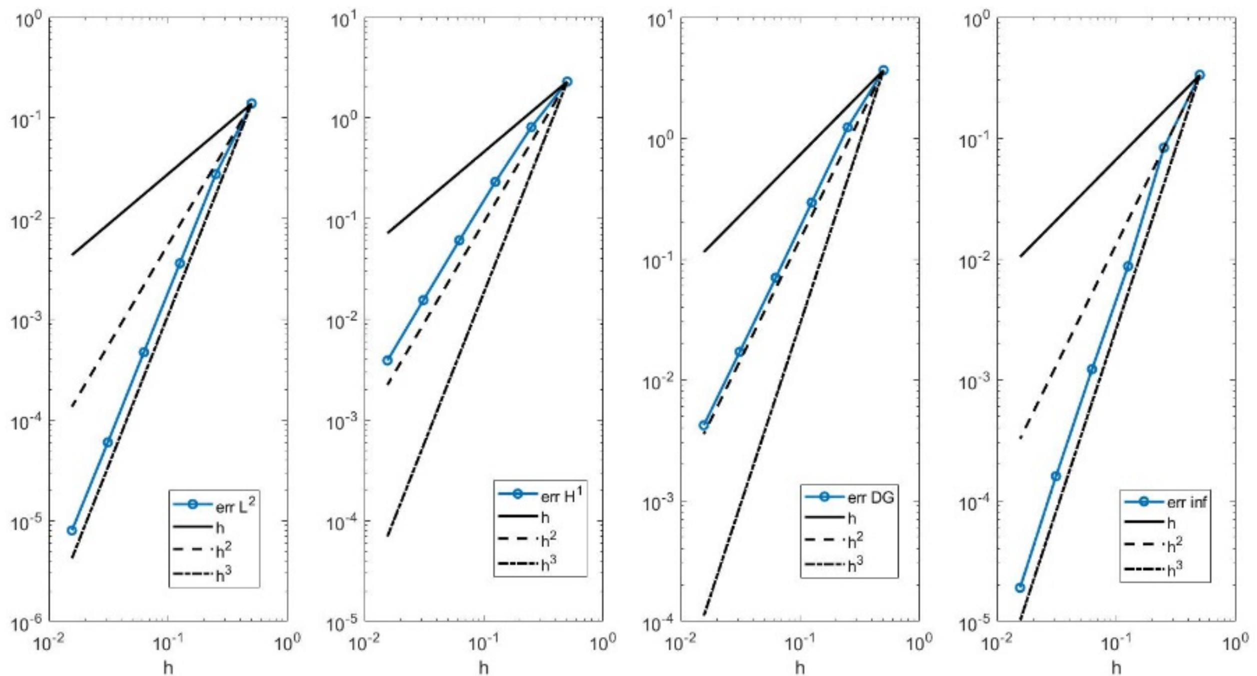


Figure 4.6: HeatDG, $d = 2, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. The same comment as in Figure 4.5.

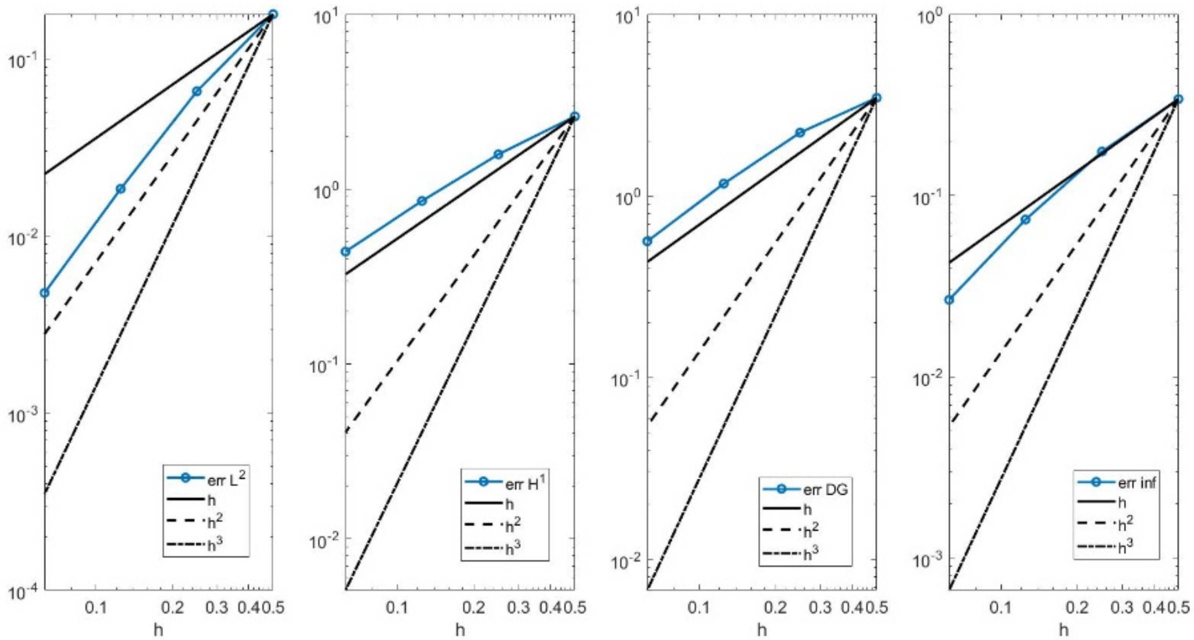


Figure 4.7: HeatDG, $d = 3, p = 1$. Expected trends are, in order, h^2, h, h, h^2. The convergence is correct even if the L^∞ error reaches gradually the right trend.

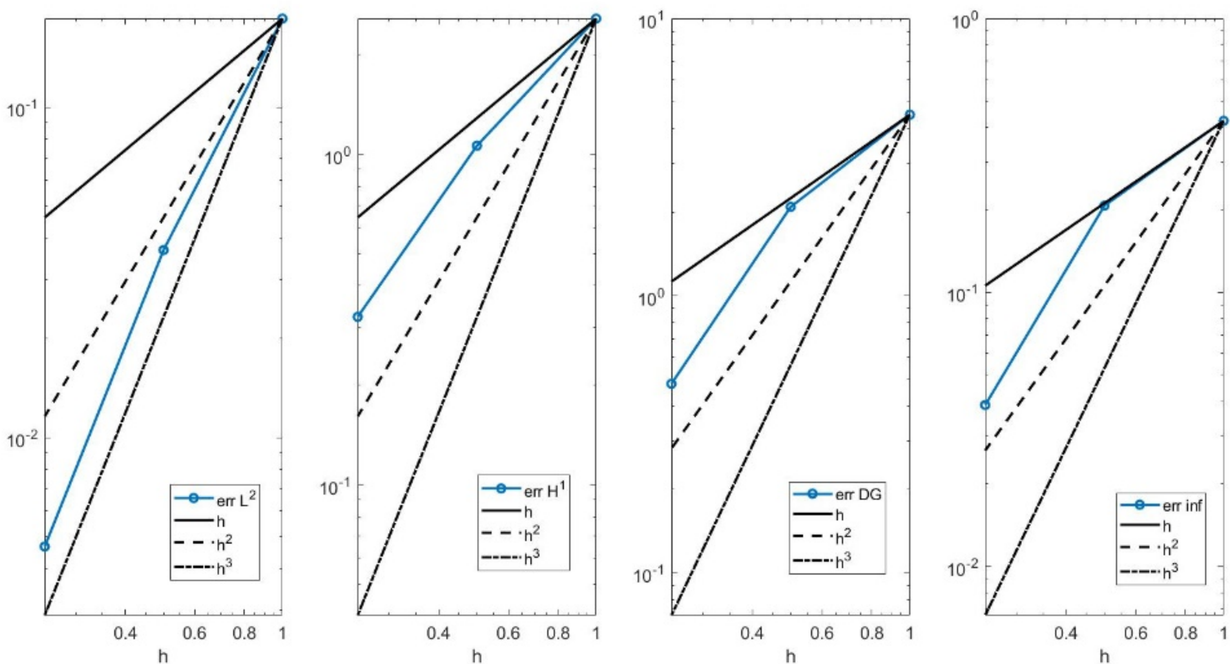


Figure 4.8: HeatDG, $d = 3, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. Trends seem correct but, due to the large time of execution, it is not possible to refine more.

4.1.2. Results for HeatDG

See Section 2.7.4 for more details about the problem. Again, we only focus on the Dubiner basis and, so, we select `HeatDG<DUBValues<lifex::dim»`. In the following figures, we set the final time $T = 0.003$, the time step $\Delta t = 0.0001$ and BDF order $\alpha = 1$. $d = 2, 3$, $p \in \mathbb{N}$ and $h > 0$ have the same meaning as before. Results are shown in Figure 4.5, 4.6, 4.7, 4.8.

4.1.3. Results for MonodomainFHNDG

See Section 2.7.5 for more details about the problem. The model is `MonodomainFHNDG<DUBValues<lifex::dim»`, therefore we have two unknowns that are the trans-membrane potential V_m and the gating variable ω. In particular, the following figures will regard only V_m, the main solution. We pose the final time $T = 0.003$, the time step $\Delta t = 0.0001$ and BDF order $\alpha = 1$. $d = 2, 3$, $p \in \mathbb{N}$ and $h > 0$ have the same meaning as before. Results are shown in Figure 4.9, 4.10, 4.11, 4.12.
In particular, in Figure 4.12, the final time T is decreased due to the larger time of execution. In this case, we set $T = 0.001$.
As for the model parameters, we used the typical values:

- $\chi_m = 10^5 \ m^{-1}$,
- $\Sigma = 0.12 \ Sm^{-1}$,
- $C_m = 10^{-2} \ Fm^{-2}$,
- $k = 19.5$,
- $\epsilon = 1.2$,
- $\Gamma = 0.1$,
- $a = 13 \cdot 10^{-3}$.

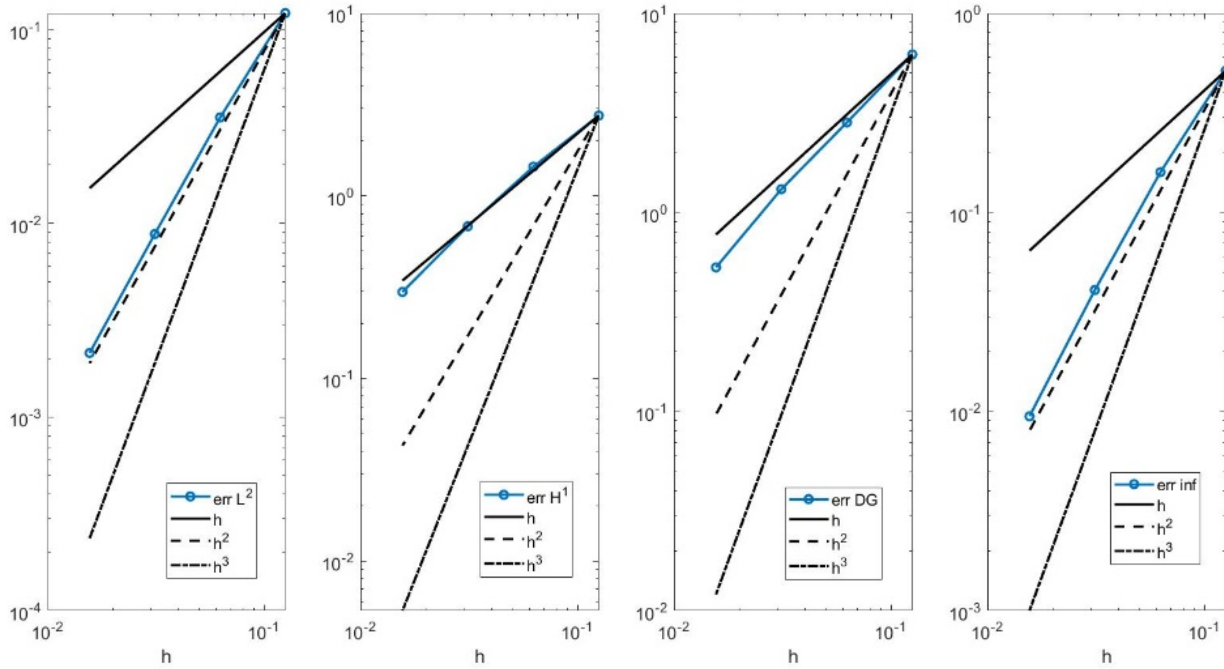


Figure 4.9: MonodomainFHNDG, $d = 2, p = 1$. Expected trends are, in order, h^2, h, h, h^2. The convergence is correct.

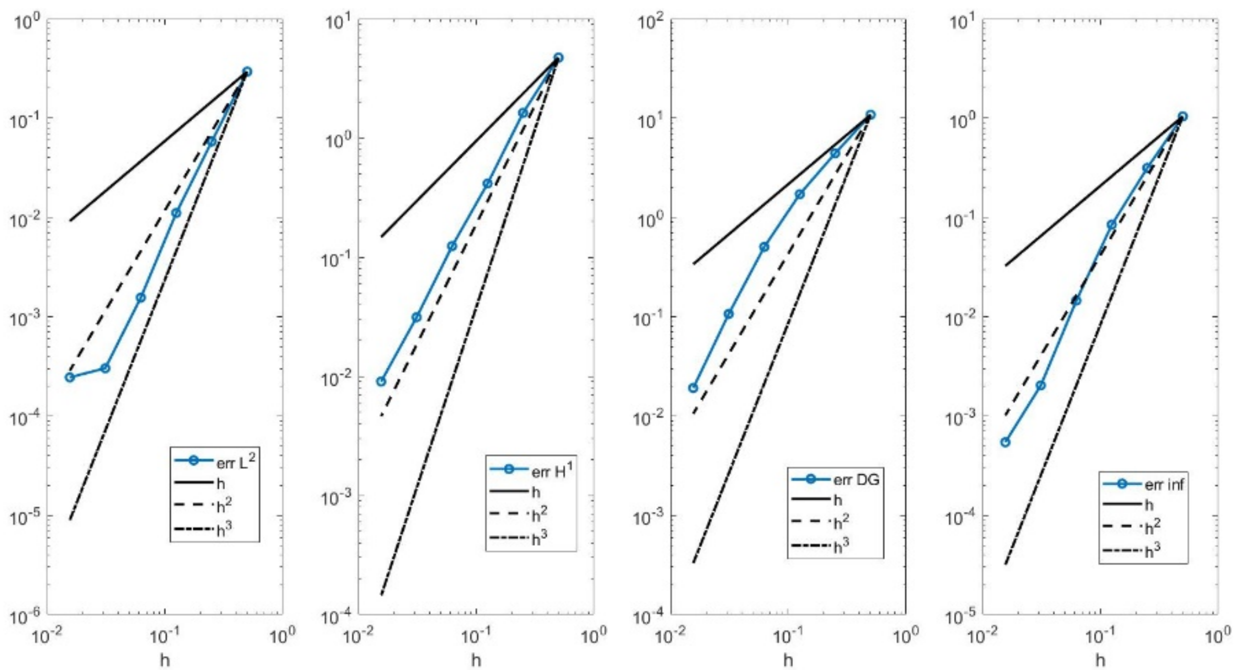


Figure 4.10: MonodomainFHNDG, $d = 2, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. The convergence is correct except for the L^∞ error which shows a h^2 trend.

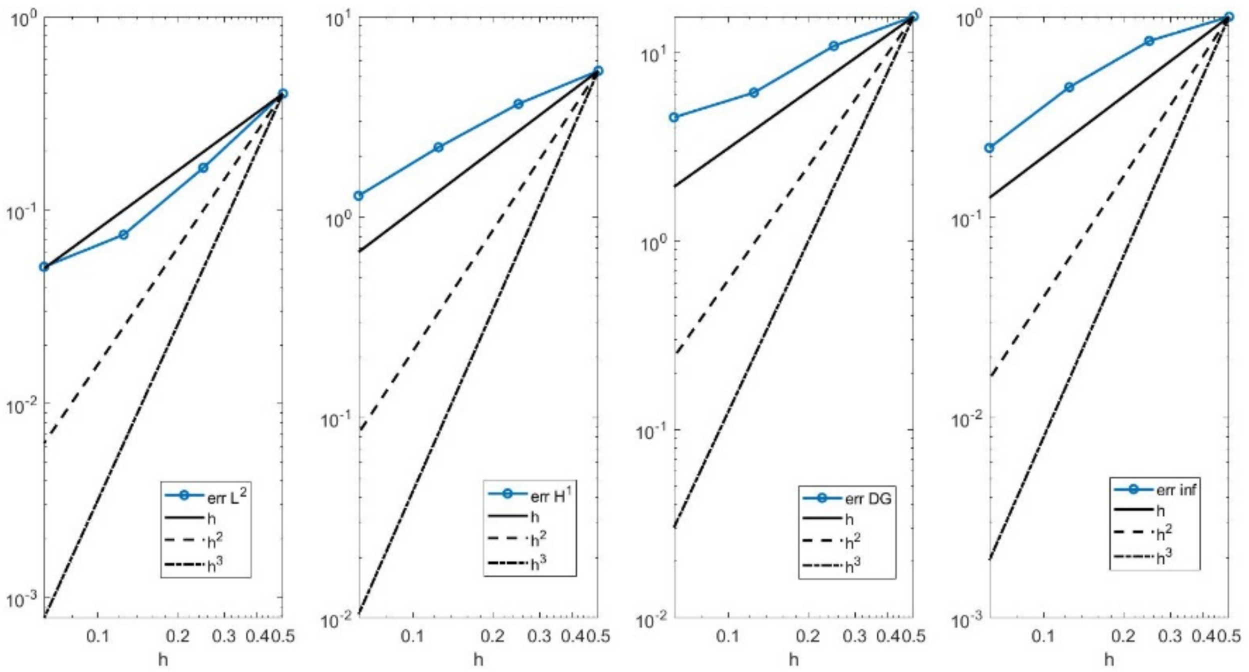


Figure 4.11: MonodomainFHNDG, $d = 3, p = 1$. Expected trends are h^2, h, h, h^2. The convergence is achieved, but the rates are not completely defined.

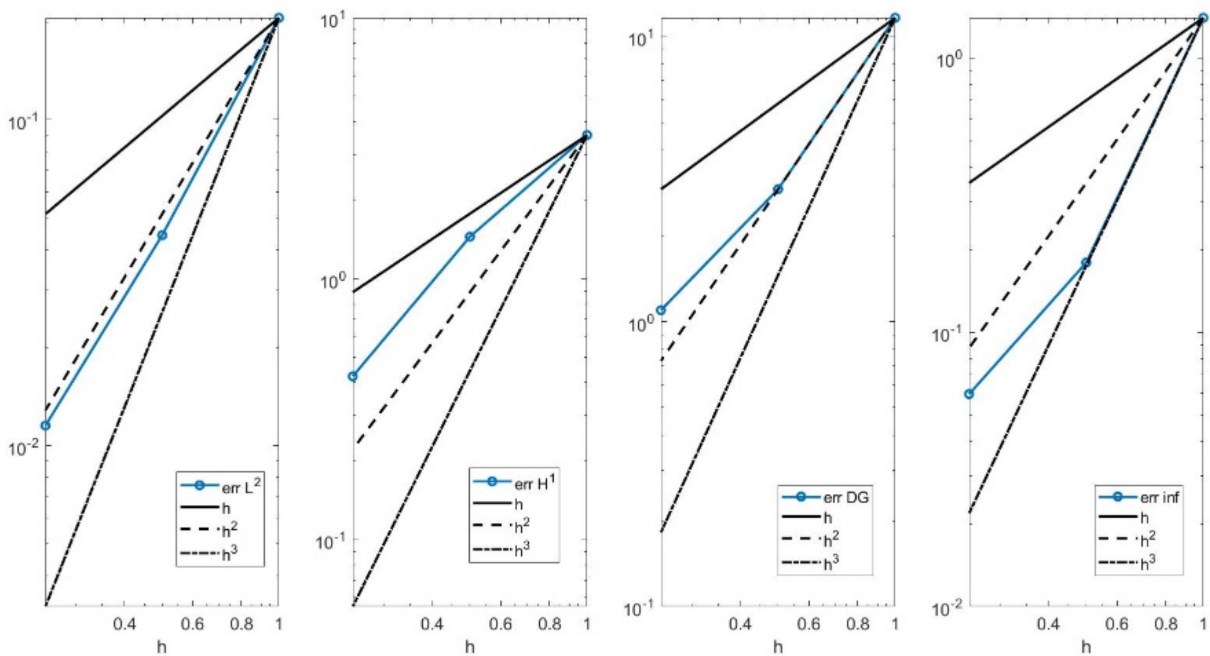


Figure 4.12: MonodomainFHNDG, $d = 3, p = 2$. Expected trends are h^3, h^2, h^2, h^3. The convergence is partially correct, but it is not possible to refine more due to computation times.

So, to conclude this section, it seems evident that most of the simulations correctly converge to the exact solutions with the expected rates. The only exception is the monodomain problem in three dimensions because, due to the computational effort, it cannot be analyzed with finer temporal discretizations and more grid refinements. In particular, one can easily spot the influence of the temporal discretization error for small grid sizes. The overall small error is, therefore, disturbed and this is the cause for the change of slopes for small h.
Besides this, we can confirm that the implemented numerical schemes are correct and `DUBeat` successfully employs the Dubiner method to solve differential problems.

4.2. Higher order convergence rates

The following study has been performed to assure that the numerical methods implemented in `DUBeat` converge at an exponential rate when we increase the polynomial order $p \in \mathbb{N}$. We remind that we are still focusing only on the Dubiner basis. In this setting, we take a coarse mesh (length of the element $\approx$ length of the domain) and we visualize the convergence when we increase the polynomial order. Results are shown in Figure 4.13, 4.14.

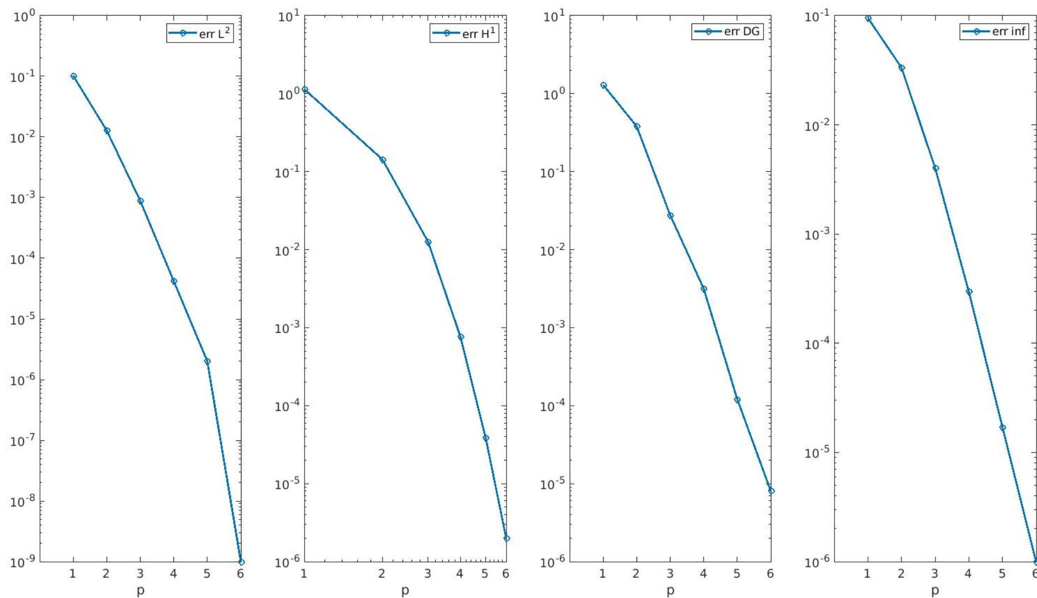


Figure 4.13: LaplaceDG, $d = 2$. Lines are straights, in a `semilogy` plot this means that the convergence is exponential ($\propto e^{-\alpha}$).

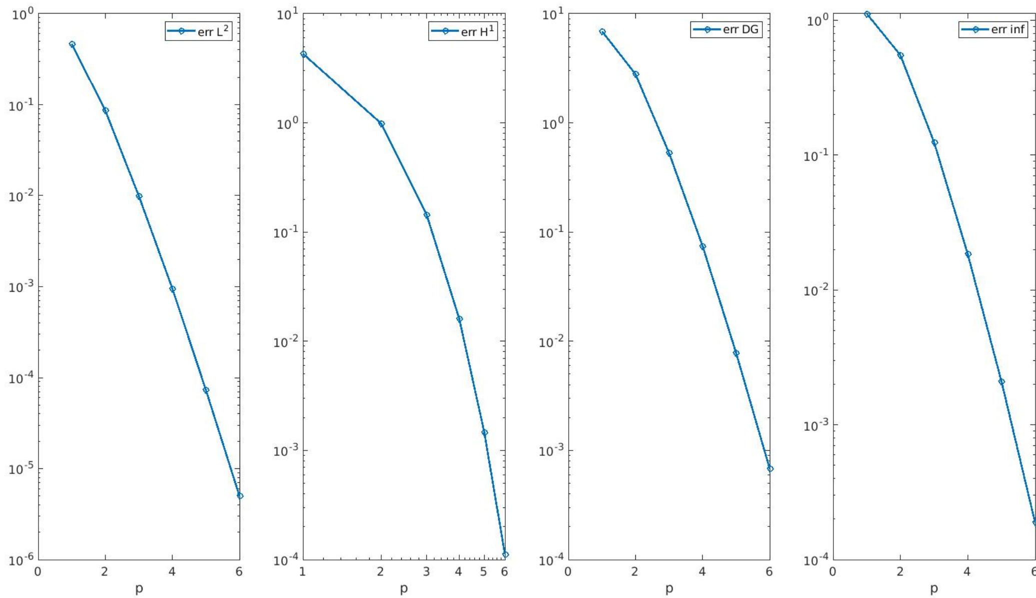


Figure 4.14: LaplaceDG, $d = 3$. Same comment as in Figure 4.13.

We clearly see that the exponential rate is achieved in both two and three dimensions. This is another guarantee of the correct behaviour of the methods.
Let us focus now on what this implies in the performance of the high-order methods.
We performed the same computations shown in Figure 4.3 and 4.14 using the HPC cluster from the MOX department at PoliMi[1] keeping track of the computation times and the number of GMRES iterations for the resolution of the problem linear system. Therefore, starting from the same configuration ($p = 1$ and same coarse mesh), we performed in one case a grid refinement convergence analysis and in the other study a convergence analysis based on the increment of the polynomial order. Data are shown in Table 4.1 and in Table 4.2.
A comparison between the two tables proves the outstanding performance of the Dubiner method when the high order is exploited. The choice of a coarse mesh with high orders leads to a fast simulation with very high accuracy. For instance, the 5th order gives in 9 minutes better accuracy than the 6-times refined mesh in 20 hours. The discrepancy is also evident from the number of GMRES iterations for the system solving. We also notice that the results in Table 4.1 are very similar to the errors when using DGFEM and, probably, also the continuous FEM.
The definitions of the errors are the same as Section 2.6.1.

[1] https://mox.polimi.it/research-areas/hpcmox/

Table 4.1: LaplaceDG, $d = 3$. Grid refinement with $p = 1$.

n° ref	1	2	3	4	5	6
time (h:m:s)	00:00:20	00:00:43	00:03:26	00:26:26	02:16:29	20:16:22
GMRES iter	1	1	132	253	1004	1000
L^∞ error	1.118910	0.432081	0.117463	0.030616	0.007814	0.001974
L^2 error	0.465414	0.035014	0.009014	0.002300	0.000582716	0.000147
H^1 error	4.301522	1.291193	0.659523	0.333049	0.167319	0.083858
DG error	6.917900	2.821673	1.132224	0.480651	0.215676	0.101097

Table 4.2: LaplaceDG, $d = 3$. Polynomial order increment with n° ref = 1.

p	1	2	3	4	5	6
time (h:m:s)	00:00:19	00:00:24	00:00:49	00:02:44	00:09:21	00:29:40
GMRES iter	1	1	1	1	2	2
L^∞ error	1.118910	0.549955	0.124414	0.018430	0.002132	0.000188
L^2 error	0.465414	0.086423	0.009940	0.000954	0.000073	0.000005
H^1 error	4.301522	0.989008	0.143108	0.016139	0.001453	0.000112
DG error	6.917900	2.797505	0.531981	0.074641	0.007822	0.000682

Hence, high-order methods such as the Dubiner method are really promising for the fast and accurate resolution of differential problems. We expect this performance to be even enhanced when dealing with steep waves in electrophysiology problems and this is the main motivation to the next section.

4.2.1. Pseudo-realistic simulation

To conclude, we present the solution of a very simple pseudo-realistic simulation for the class `MonodomainFHNDG<DUBValues<lifex::dim»` from Section 2.7.5. It is noteworthy to warn that no exact solutions are known for these kinds of problems.

In this particular test, the domain concretely represents a square section of the surface of the heart that is electrically isolated (i.e., the Neumann boundary conditions are homogeneous) and an external current is applied in a central region for a limited interval of time. From a concrete perspective, it simulates the action of a defibrillator.

The main parameters are set in table 4.3.

Table 4.3: Parameters for the pseudo-realistic simulation.

Dimension	$d = 2$
Domain (m)	$\Omega = [-0.025, 0.035]^2$
Time-step (s)	$\Delta t = 0.0001$
Applied current (Am^{-3})	$I_{ext}(x, y, t) = 150 \cdot 10^3 \cdot \left(I_{t\in[0.001,0.002)} \cdot I_{x\in(0.004,0.006)} \cdot I_{y\in(0.004,0.006)}\right)$
Neumann BC (Am^{-2})	$g(x, y, t) = 0$
Initial conditions	$V_{m_{ex}} = 0$ and $\omega_{ex} = 0$
Polynomial degree	$p = 6$
Number of elements	8

Moreover, the model parameters are the same as in Section 4.1.3.
The last two parameters in Table 4.3 are motivated by the fact that a resolution with high polynomial order coupled with a coarse mesh, as proved in Section 4.2, is a very effective approach for its time of execution and its accuracy.
The following figures, from Figure 4.15 to Figure 4.22, represent a visualization of the trans-membrane variable V_m at different time-steps. The highest value is achieved with $V_m = 1$ (red) and the lowest is approximately zero (blue).
It is possible to follow the propagation of the steep wave until the entire domain is activated (i.e., $V_m \simeq 1$ everywhere). This is coherent with the theory and the realistic phenomena, except for the dynamics that follows after the activation: after the depolarization, the cardiac cells should repolarize returning to their initial value at rest. Instead, the potential here stabilizes at the peak value. This discrepancy was already pointed out in [6] and it is probably due to the ineffectiveness of the simple Fitzhugh-Nagumo model to fully describe the realistic phenomenon.
Despite the very simplicity of this test, we have encountered once again confirmations for the correct functioning of the library and its methods. Furthermore, the dynamics of the solution has an analogy with the same result in [6].

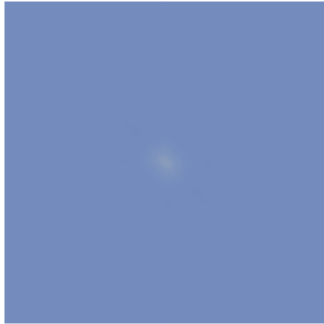


Figure 4.15: V_m at time 0.01 s.

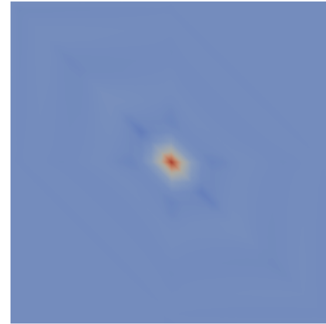


Figure 4.16: V_m at time 0.02 s.

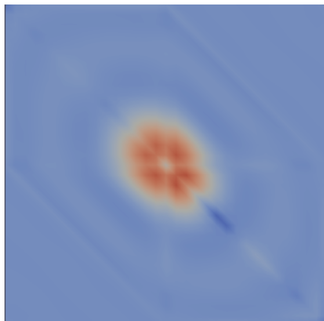


Figure 4.17: V_m at time 0.03 s.

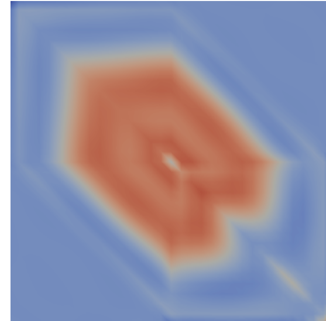


Figure 4.18: V_m at time 0.04 s.

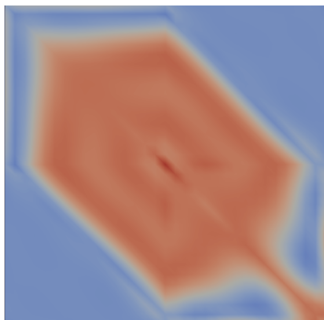


Figure 4.19: V_m at time 0.045 s.

Figure 4.20: V_m at time 0.05 s.

Figure 4.21: V_m at time 0.06 s.

Figure 4.22: V_m at time 0.07 s.

5 | Conclusion

DUBeat is a new and promising project that attempts to exploit non-standard methods with the goal to correctly solve problems that are otherwise challenging when using normal techniques. The initial and most demanding task was the construction of an efficient, clear and general structure that permits to easily employ the two DG methods to both standard and realistic problems. The numerical and computational outcomes from Section 4 are encouraging and certainly show how the initial idea as well as the demanding work to create DUBeat are really successful. We also remind that the project started from the initial motivation and results of [6] that turned out to be optimistic and lead to a more widespread project such as DUBeat.
With this article, we officially publish the DUBeat 1.0.0 version while many improvements will certainly be considered in the next versions. The main goals are the parallelization of certain operations and the applications to more realistic and interesting problems. The authors are indeed already working on a problem that exploits the more realistic ten Tusscher Panfilov ionic model ([9]) applied to the monodomain model. We hope that the initial excellent results of the methods will lead to the same level of performance when applied to realistic and complex simulations. Moreover, we hope that the community could contribute in the future to the development of the open-source project and concretely enhance its features.

Acknowledgements

The authors wish to thank the MOX advisors who helped us in this amazing project and accompanied us for a very long time. First of all, we are grateful to prof. Paola Antonietti for her knowledge of high-order DG methods and her assistance in the correct implementation and analysis of such schemes. Then, we thank prof. Christian Vergara for his support in the field of electrophysiology modelling and the realistic interpretations of our work. The original intuition for this project was suggested by them and we are really delighted for the year and half that we spent working together to make it possible. Finally, we thank doctor Pasquale Africa for his recent support in the development of `DUBeat`. It would not have been possible to finalize this project without his assistance during the implementation of the library and his support for the `lifex` integration.

Bibliography

[1] P. C. Africa. lifex: a flexible, high performance library for the numerical solution of complex finite element problems. 2022. doi: 10.48550/ARXIV.2207.14668.

[2] F. Andreotti and D. Uboldi. Discontinuous galerkin approximation of the monodomain problem. *Politecnico di Milano*, 2021.

[3] P. F. Antonietti and P. Houston. A class of domain decomposition preconditioners for hp-discontinuous galerkin finite element methods. *Journal of Scientific Computing*, 46, 2011. ISSN 0885-7474.

[4] D. Arndt, W. Bangerth, M. Feder, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, S. Sticko, B. Turcksin, and D. Wells. The `deal.II` library, version 9.4. *Journal of Numerical Mathematics*, 2022. doi: 10.1515/jnma-2022-0054. Accepted.

[5] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified analysis of discontinuous galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis*, 39(5):1749–1779, 2002. doi: 10.1137/S0036142901384162.

[6] F. Botta and M. Calafà. A high-order discontinuous galerkin method for the bidomain problem of cardiac electrophysiology. *Politecnico di Milano*, 2021.

[7] P. Colli Franzone, L. F. Pavarino, and S. Scacchi. *Mathematical Cardiac Electrophysiology*. Springer-Verlag, Cham, 2014.

[8] M. Dubiner. Spectral methods on triangles and other domains. *Journal of Scientific Computing*, 6:345–390, 1991.

[9] T. T. K. H. and P. A. V. Alternans and spiral breakup in a human ventricular tissue model. *American journal of physiology. Heart and circulatory physiology*, 291(3), 2006. doi: 10.1152/ajpheart.00109.2006.

[10] T. Koornwinder. Two-variable analogues of the classical orthogonal polynomials. *Theory and Application of Special Functions*, pages 435–495, 1975. doi: https://doi.org/10.1016/B978-0-12-064850-4.50015-X.

[11] A. Quarteroni. *Numerical Models for Differential Problems*. MS&A. Springer Milan, 2014. ISBN 9788847055223.

[12] A. Quarteroni, A. Manzoni, and C. Vergara. The cardiovascular system: Mathematical modelling, numerical algorithms and clinical applications. *Acta Numerica*, pages 365–590, 2017.

[13] F. R. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961. doi: https://doi.org/10.1016/s0006-3495(61)86902-6.

[14] S. J. Sherwin and G. E. Karniadakis. A new triangular and tetrahedral basis for high-order (hp) finite element methods. *International Journal for Numerical Methods in Engineering*, 38(22):3775–3802, 1995. doi: https://doi.org/10.1002/nme.1620382204.

[15] G. Szego. *Orthogonal Polynomials*. American Math. Soc: Colloquium publ. American Mathematical Society, 1939. ISBN 9780821810231.

[16] K. P. Vincent, M. J. Gonzales, A. K. Gillette, C. T. Villongco, S. Pezzuto, J. H. Omens, M. J. Holst, and A. D. McCulloch. High-order finite element methods for cardiac monodomain simulations. *Frontiers in physiology*, 6, 217, 2015. doi: https://doi.org/10.3389/fphys.2015.00217.

List of Figures

1.1 2D transformation from the reference square $\hat{Q}$ to the reference triangle $\hat{K}$ (courtesy of Paola F. Antonietti). 5
1.2 Dubiner basis functions that generate $\mathbb{P}^5(\hat{K})$ (courtesy of G. Steiner). 6

4.1 LaplaceDG, $d = 2, p = 1$. Expected trends are, in order, h^2, h, h, h^2. The convergence is correct and, for the DG error, even better than the estimated trend. 30
4.2 LaplaceDG, $d = 2, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. The same comment as in Figure 4.1. 30
4.3 LaplaceDG, $d = 3, p = 1$. Expected trends are, in order, h^2, h, h, h^2. The convergence is correct and, for the DG error, even better than the estimated trend. 31
4.4 LaplaceDG, $d = 3, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. The same comment as in Figure 4.3. 31
4.5 HeatDG, $d = 2, p = 1$. Expected trends are h^2, h, h, h^2. The convergence is correct. 32
4.6 HeatDG, $d = 2, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. The same comment as in Figure 4.5. 32
4.7 HeatDG, $d = 3, p = 1$. Expected trends are, in order, h^2, h, h, h^2. The convergence is correct even if the L^∞ error reaches gradually the right trend. 33
4.8 HeatDG, $d = 3, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. Trends seem correct but, due to the large time of execution, it is not possible to refine more. . 33
4.9 MonodomainFHNDG, $d = 2, p = 1$. Expected trends are, in order, h^2, h, h, h^2. The convergence is correct. 35
4.10 MonodomainFHNDG, $d = 2, p = 2$. Expected trends are, in order, h^3, h^2, h^2, h^3. The convergence is correct except for the L^∞ error which shows a h^2 trend. . . . 35
4.11 MonodomainFHNDG, $d = 3, p = 1$. Expected trends are h^2, h, h, h^2. The convergence is achieved, but the rates are not completely defined. 36
4.12 MonodomainFHNDG, $d = 3, p = 2$. Expected trends are h^3, h^2, h^2, h^3. The convergence is partially correct, but it is not possible to refine more due to computation times. 36
4.13 LaplaceDG, $d = 2$. Lines are straights, in a `semilogy` plot this means that the convergence is exponential ($\propto e^{-\alpha}$). 37

4.14 LaplaceDG, $d = 3$. Same comment as in Figure 4.13. 38
4.15 V_m at time 0.01 s. 41
4.16 V_m at time 0.02 s. 41
4.17 V_m at time 0.03 s. 41
4.18 V_m at time 0.04 s. 41
4.19 V_m at time 0.045 s. 41
4.20 V_m at time 0.05 s. 41
4.21 V_m at time 0.06 s. 41
4.22 V_m at time 0.07 s. 41

List of Tables

4.1 LaplaceDG, $d = 3$. Grid refinement with $p = 1$. 39
4.2 LaplaceDG, $d = 3$. Polynomial order increment with n° ref $= 1$. 39
4.3 Parameters for the pseudo-realistic simulation. 40

List of Acronyms

BDF Backward Differentiation Formula.

DG Discontinuous Galerkin Method.

DGFEM Discontinuous Galerkin Method with FE basis.

DOF Degrees of Freedom.

FE Finite Element.

FEM Finite Element Method.

GMRES Generalized Minimal Residual Method.

PDE Partial Differential Equation.