MOX-Report No. 54/2026

The lymph 2.0 library: p-adaptive algorithms and parallel assembly strategies for polytopal DG methods

Antonietti, P. F.; Corti, M.; Leimer Saglio, C. B.; Pagani, S.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it https://mox.polimi.it

The lymph 2.0 library: p-adaptive algorithms and parallel assembly strategies for polytopal DG methods*

Paola F. Antonietti[1], Mattia Corti[1], Caterina B. Leimer Saglio[1], and Stefano Pagani[1]

[1] *MOX-Dipartimento di Matematica, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milan, 20133, Italy*

June 23, 2026

Abstract

This work presents a new release of the `lymph 2.0` library [1], an open-source MATLAB framework for high-order discontinuous Galerkin discretizations on general polytopal meshes. The `lymph 2.0` version is extended to support discretizations with element-wise polynomial approximation degrees, which allows the design of p-adaptive strategies based on a posteriori error indicators. In addition, the library introduces a unified assembly framework that abstracts the construction of discrete operators from the underlying physical model, improving code modularity, parallelism, maintainability, and extensibility. Moreover, the proposed approach enables shared-memory parallelism through dedicated parallel tools. Several numerical examples demonstrate the effectiveness of the proposed developments in reducing the computational cost while preserving approximation accuracy.

1 Introduction

This paper discusses recent updates to the `lymph` [1] library (`https://bitbucket.org/lymph/lymph`), an open-source MATLAB framework for the numerical approximation of coupled multi-physics problems exploiting polytopal discontinuous Galerkin (PolyDG) discretizations [4, 11, 10]. The code is specifically designed to address both single-physics and multi-physics problems, offering a modular and extensible structure for mesh handling, finite element space construction, and system assembly. The library supports a variety of applications, including elliptic, parabolic, and hyperbolic problems, as well as coupled systems arising in multi-physics contexts [2, 3, 7, 8, 18].

The present release addresses two important improvements of the library. First, we extend the library to support p-adaptive discretizations [6, 16, 19, 20], allowing for element-wise variation of the polynomial degree, for elliptic and (semilinear) parabolic problems, for which we use the a-posteriori error indicator [9, 15, 18]. The p-adaptive framework reduces the total number of degrees of freedom of the system while retaining a high level of accuracy in the solution approximation. Second, we develop a unified assembly framework that abstracts the construction of discrete operators from the specific physical model, enabling a more modular, reusable, and extensible implementation of multi-physics solvers. These improvements significantly simplify the integration of new physical models within the library. In addition, the new assembly algorithm is explicitly designed to exploit shared-memory parallelism, allowing local contributions to be computed concurrently and thereby improving scalability on modern multicore architectures. Finally, we release two new physics, namely, the Fisher-Kolmogorov equation [13, 17] and the monodomain equation coupled with the FitzHugh-Nagumo ionic model [14, 21], that represent two possible examples of nonlinear partial differential equations treatment within the framework of the novel release of the `lymph 2.0` library.

These new developments significantly improve the flexibility and scalability of the library. The structure of `lymph 2.0` library with the highlighted modifications is reported in Figure 1. The paper is organized as follows. Section 2 presents the implementation of discretizations with element-wise polynomial approximation degrees, including, for

*__Funding__: This work is partially funded by the European Union (ERC SyG, NEMESIS, project number 101115663). Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. CBLS has been funded by the National Recovery and Resilience Plan (NRRP), Mission 4, Component 1 – Investment 3.4 and Investment 4.1, funded by the European Union. MC and CBLS acknowledge "INdAM - GNCS Project", codice CUP E53C25002010001. The present research is part of the activities of the Dipartimento di Eccellenza 2023-2027 grant, funded by MUR. PFA, MC, SP and CBLS are members of INdAM-GNCS.

selected physical models, p-adaptive strategies. Section 4 we resume the main improvements of the library, while Section 3 introduces the new unified assembly framework. Section 5 to demonstrate the effectiveness of the proposed features. Finally, conclusions are drawn in Section 6.

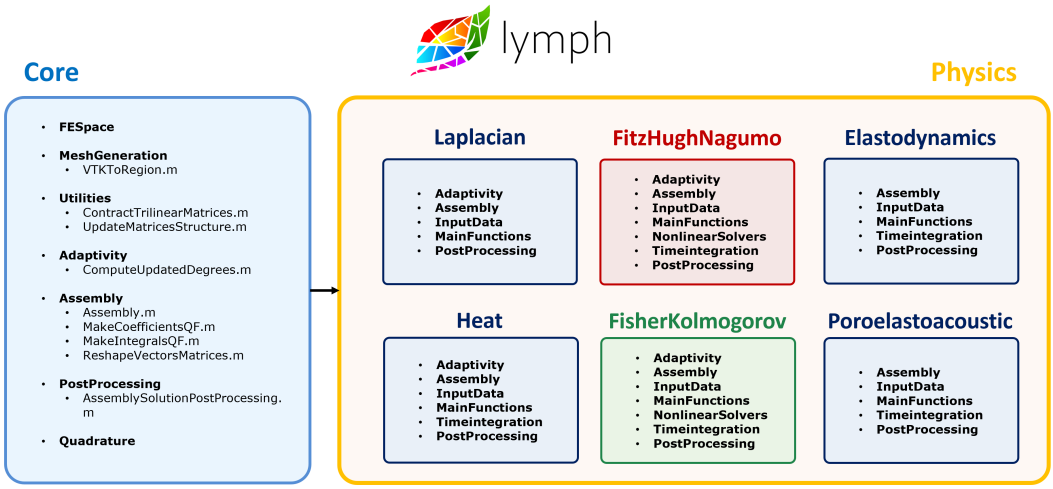


Figure 1: `lymph 2.0` code structure: highlights of the novel modules and logo (top center).

2 The p-adaptive framework

One of the main advantages of PolyDG methods is the possibility of using element-wise polynomial approximation orders, thus naturally supporting p-adaptive discretization strategies. Indeed, let $p_K \geq 1$ denote the polynomial degree associated with the element $K \in \mathscr{T}_h$. The discrete approximation space is then defined as

$$V_h^p = \left\{ v_h \in L^2(\Omega) : v_h|_K \in \mathbb{P}^{p_K}(K) \quad \forall K \in \mathscr{T}_h \right\}.$$

This setting is especially attractive in multiscale and heterogeneous problems, where different regions of the domain may require different levels of approximation accuracy. In such situations, locally increasing the polynomial degree only where needed may lead to a more efficient use of the degrees of freedom than a globally uniform discretization.

The original `lymph` framework adopted a discrete finite element space formulated in terms of a global polynomial degree stored in the variable `femregion.degree`. In this release, we first implement an element-wise discretization degree and a corresponding local number of basis functions, `femregion.nbases`, for each element. Moreover, we introduce a complete workflow for p-adaptivity for the Laplace equation and for semilinear parabolic problems. The routines include a posteriori error indicator computation and efficient updates of the corresponding matrices and finite element structures. From a theoretical point of view, this work is based on the a posteriori error indicator for the polyDG discretization of the Laplacian operator in [9] and on the algorithm proposed by [18] for p-adaptivity for semilinear parabolic problems. The latter equations are discretized in time in $(0, T)$ by means of finite difference time stepping $t_0 = t^{(0)} \leq t^{(1)} \leq ... \leq t^{(N)} = T$. A key aspect of the implementation is that the adaptive workflow is entirely driven by the local evaluation of a suitable (residual-based) indicator $\tau_K^{(n)}$, for any $K \in \mathscr{T}_h$ at time $t^{(n)}$. The routine `ComputeUpdatedDegrees` evaluates the local indicator values and computes a new polynomial distribution according to the adaptive law

$$p_K^{\text{new}} = f_{\text{adapt}}\left(\frac{\tau_K^{(n)}}{\tau_{\text{threshold}}} \right), \quad K \in \mathscr{T}_h,\ n \in \{0, \dots, N\},$$

where f_{adapt} is a user-defined adaptivity function set in `Data.AdaptFunc`, and $\tau_{\text{threshold}}$ is automatically determined during the first adaptive iteration through a k-means clustering procedure applied to the indicator distribution. For a detailed analysis of the algorithm, we refer to [18].

To save computational time, the adaptive indicator is not computed for all elements K by default. Indeed, if the solution at a given time snapshot is nearly stationary on the element $K \in \mathscr{T}_h$, the computation of $\tau_K^{(n)}$ is

skipped on the element K. More precisely, let $U^{(n-1)}$ and $U^{(n)}$ be the expansion coefficients vectors of the time snapshots $t^{(n-1)}$ and $t^{(n)}$, respectively, namely $u_h^{(n)}(\boldsymbol{x}) = \sum_{j=1}^{N_K} U_j^{(n)} \varphi_j(\boldsymbol{x})$, where $\varphi_j(\boldsymbol{x})$ is the set of local basis functions. This choice is driven by the variation of the Euclidean norm $\|U^{(n)} - U^{(n-1)}\|$ at time $t^{(n)}$. The indicator on element K is computed whenever this norm exceeds a (user-defined) tolerance, set by default to 10^{-6} in the function `ComputeAdaptiveIndicatorPhysicsName` for each physics. For the remaining elements, the previously computed indicator values are used, namely $\tau_K^{(n)} = \tau_K^{(n-1)}$.

As a consequence, the p-adaptive routine dynamically redistributes the approximation order over the mesh according to the information on the local approximation quality. The implementation also exploits the hierarchical structure of the polynomial basis [1]. Indeed, if the elementwise polynomial degree is reduced from one adaptive step to the next, the code automatically extract the new matrices associated with volume integrals from the previous higher-order representation, thereby accelerating the reassembly time at each adaptive step.

Example of p-adaptive routine: the heat equation. Starting from an initial discretization associated with a `femregion` structure, the `AdaptivityCycle` routine for each physics performs a sequence of `Data.AdaptIts` adaptive iterations.

Listing 1: Adaptive cycle for the heat problem.

```
%% Physics/Heat/Adaptivity/AdaptivityCycle.m

    AdaptCount = 0;
    while AdaptCount < Data.AdaptIts
        ... % Indicator computation
        [Solution.Indicator] = ComputeAdaptiveIndicatorHeat(.);
        ... % Update the polynomial degrees
        [Data, femregion] = ComputeUpdatedDegrees(.);
        ... % Project the solution onto the new femregion
        [Solution] = ProjectSolutionAssemblyHeat(.);
        ... % Reassemble the matrices
        [Matrices] = MatrixAssemblyHeat(.);
        ...
    end
```

As shown in Listing 1, the a posteriori indicator is first computed through the code `ComputeAdaptiveIndicatorHeat` at each time iteration. The implementation of the indicator is organized within an `Adaptivity` module in each physics and relies on a sequence of dedicated routines based on the novel common `Assembly` function. In particular, the adaptive indicator computation involves the following steps:

- `IndicatorPreallocation` initializes the data structures required for the local indicators.
- `ResidualIndicatorAssembly` assembles the cell residual contributions.
- `FacesIndicatorsAssembly` computes the interface and jump terms.
- `FinalIndicator` combines all local contributions into the element-wise adaptive indicator $\tau_K^{(n)}$.

Based on these indicators, the routine `ComputeUpdatedDegrees` updates the local polynomial distribution over the mesh, producing a new `femregion` structure with element-wise polynomial degrees adapted to the current solution features. Once the approximation spaces have been modified, the previously computed solution is projected onto the new discrete space by means of the function `ProjectSolutionAssemblyHeat`. This projection provides a consistent initialization for the subsequent adaptive iteration. The matrices associated with the updated discretization are then reassembled through `MatrixAssemblyHeat`, accounting for the new local polynomial configuration. The procedure is repeated until the prescribed number of adaptive iterations is reached, progressively refining the distribution of local polynomial degrees and improving the overall approximation efficiency.

3 An unified and parallel assembly framework

The first release of `lymph` [1] already provided efficient strategies for the evaluation of element and face contributions in matrix and vector assembly routines. In particular, the library supported both quadrature-free (QF) [5] and subtriangulation (ST) approaches for polygonal meshes. In this release, we introduce a unified assembly routine that acts as a common driver for the construction of problem-dependent matrices and vectors. The new implementation

decouples the general logic of the assembly process from the specific integrals associated with each physical model, resulting in a modular and reusable framework. Moreover, the implementation leverages MATLAB's `parfor` loops to assemble local matrices in parallel, improving performance in large-scale computations.

The introduction of a common assembly driver provides several advantages. First, it reduces code duplication by concentrating all quadrature handling, local indexing, and sparse reconstruction in a single routine. As a consequence, it improves maintainability and the improvements to all physical models immediately. Finally, it enhances extensibility, incorporating new physics by defining only the local assembly kernels, without altering the global workflow. Finally, this assembly function is at the basis of both the matrices and forcing terms assembly, as well as the adaptivity indicators and the error computations. In particular, the latter in the first version of the library were computed using the projected solutions instead of the exact ones, potentially leading to approximation issues whenever the underlying solution exhibits sharp fronts.

Listing 2: Common assembly.

```
%% Core/Assembly/Assembly.m

function [Matrices] = Assembly(Data, neighbor, femregion, AssembInfo, Funcs)
    ... % Preallocation of local matrices for a single element and repetition for all the mesh ones
    Matrices = Funcs.Preallocation(GenMatrices);
    Matrices = repmat({Matrices}, femregion.nel, 1);
    ...
    parfor ie = 1:femregion.nel
        ...
        if AssembInfo.assemblyvolumes
            if AssembInfo.ass_vol_vec(ie)
                ... % Assembly the volume
                switch AssembInfo.quadrature
                    case 'QF'
                        ...
                        [Matrices{ie}.Volume]   = Funcs.VolumeAssemblyQF(.);
                        [Matrices{ie}.Volume3L] = Funcs.Volume3LAssemblyQF(.);
                    case 'ST'
                        ...
                        [Matrices{ie}.Volume]   = Funcs.VolumeAssemblyST(.);
                        [Matrices{ie}.Volume3L] = Funcs.Volume3LAssemblyST(.);
                end
            else
                ... % Reuse and eventual cut of previously assembled matrices
            end
        end

        if AssembInfo.assemblyfaces
            ... % Loop over faces
            for iedg = 1 : neighbor.nedges(ie)
                ... % Computation of the matrices related to face integrals
                [Matrices{ie}.Faces] = Funcs.FacesAssembly(.);
            end
        end
    end
    ... % Construction of final global matrices
    [Matrices] = Funcs.FinalMatrices(Matrices);
end
```

General structure of the `Assembly` routine. The new framework is built around a single `Assembly` function (whose structure is reported in Listing 2 that has as inputs, besides the mesh and finite element data structures, two problem-dependent ones: `AssembInfo` and `Funcs`. The first collects all the information needed to control the assembly workflow. This includes the choice of the quadrature strategy for volume terms, logical flags determining which types of integrals must be assembled (i.e., volume, external/internal face, trilinear terms), and additional options related to basis-function derivatives and adaptive updates. These flags allow the code to skip unnecessary computations in specific assembly calls, thereby preserving the original efficiency. Additionally, the structure contains the current time or solution values for the assembly of nonlinear or forcing terms.

The `Funcs` structure instead contains the problem-specific routines that define the contributions for the specific physical problem. Namely, the structure requires a handle for matrix preallocation, volume, face, and trilinear form assembly, as well as final construction of the global matrices. As a consequence, the same assembly engine can be

reused for different applications by simply changing the local assembly kernels collected in `Funcs`.

The unified routine preserves at the volume level the support of both QF and ST strategies, consistently with the original release [1]. The two different implementations for the specific physics need to be separately provided through the handles associated with `Funcs.VolumeAssemblyQF` and `Funcs.VolumeAssemblyST`, respectively. In any case, only the strategy chosen by the flag `AssembInfo.quadrature` will be used in practice. As an additional advantage, in the subtriangulation assembly of the volume terms, we eliminated an unnecessary loop over the resulting triangles, improving the overall computational time. On the contrary, the face terms are always computed using a subtriangulation of the segment, without making use of QF strategies. For this reason, only a function `Funcs.FacesAssembly` needs to be constructed. As in the original release, the assembly strategy is based on the construction of local matrices and index structures used to map local degrees of freedom to the global algebraic system. This process can now be performed in an element-wise parallel approach by using the `parfor` construct to distribute the assembly strategy across multiple cores, by setting `Setup.isParallel` equal to 1. Once all local contributions have been computed, the data are reshaped, filtered to remove unused entries, and passed to the final reconstruction stage. In the common assembly framework, only the assembly of local matrices is under the control of the single physics, while the construction of the global ones is automated and contained in the `ReshapeVectorMatrices` function. The final matrices associated with the discretization method are then constructed through the problem-specific `Funcs.FinalMatrices` routine.

Integration with p-local workflows. The `Assembly` function has been adapted to account for element-wise varying polynomial degrees. The design extends to adaptive settings, where local polynomial degrees may be updated at different stages. To avoid unnecessary computations, in the p-adaptivity steps, the assembly process considers only the elements contained in the vectors `AssembInfo.ass_vol_vec` and `AssembInfo.ass_face_vec`, based on the local degree changes. On the contrary, for the remaining elements the assembly process can be skipped by reusing the previously computed matrices or by extracting them in case of reduction of the polynomial degree p.

Nonlinear PDEs. Another important feature of the new assembly routine is the possibility of handling, within the same framework, nonlinear contributions as well. Concerning general forms of nonlinearities, the assembly is typically called inside the nonlinear solver. To handle the presence of previous iteration solutions, we manage the passage of the information within the `AssembInfo` structure. The assembly of the specific structures in this case is performed within the classical `VolumeAssembly` and `FacesAssembly` functions. On the contrary, we provide a separate treatment for trilinear contributions. Indeed, the quadratic reaction terms of semilinear equations are controlled by the logical flag `assemblytrilinearforms`, which activates `VolumeAssembly3L` for the construction of three-dimensional tensors $[\mathbf{M}]_{ijk} = \int_\Omega \varphi_i(\boldsymbol{x})\varphi_j(\boldsymbol{x})\varphi_k(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$. For these terms (e.g., in the Fisher-Kolmogorov equation), the use of the block-diagonal structure of the volume matrices allows contracting the resulting tensor into a matrix using the tool `ContractTrilinearMatrix`, which is more efficient than reassembling it at every iteration.

Multiphysics problems. This unified perspective is particularly advantageous for multiphysics problems. Different operators, potentially associated with distinct fields or submodels, can be assembled by combining appropriate local routines while relying on a common high-level driver. As a result, the implementation becomes more robust, easier to maintain, and naturally extensible to new coupled formulations. Moreover, the approach enables the assembly of all physical contributions within a single loop, avoiding repeated preallocation and quadrature steps.

4 Additional improvements

In addition to multiple bug fixes and performance improvements, the new release includes the following novel improvements:

- **New models released in Physics:** In this release, we introduce the Fisher-Kolmogorov equation and monodomain equation coupled with the FitzHugh-Nagumo ionic model as examples of semilinear PDEs stemming, for example, in the fields of biological population dynamics and electrophysiology.
- **Improved postprocessing assembly:** As done for the matrices, the postprocessing is now based on a common function `AssemblySolutionPostProcessing` that minimizes code duplication in loops.

- **Distributed matrices in parallel:** In case of large computations, it can be useful to distribute the matrices between different cores to save computational time. An example of the use of MATLAB's `distribute` function is provided in the `Elastodynamics` module.
- **Background postprocessing in parallel framework:** Whenever the parallel flag `Setup.isParallel` is activated, the library automatically makes use of the `parfeval` function for postprocessing solutions, performing it in the background without slowing the evaluation of subsequent time steps.
- **Input VTK [22] meshes:** The library can now read meshes in the `.vtk` format. The function `VTKtoRegion` automatically handles the generation of the `region` and `neighbor` structures and saves them in a `.mat` file for future use.

5 Numerical examples

In the following, we show some paradigmatic examples to illustrate the new capabilities of `lymph` in handling both steady-state and time-dependent, as well as scalar and vector-valued PDEs.

5.1 The Poisson problem: p-adaptive solver

We start by considering the Poisson problem in a polygonal domain $\Omega \subset \mathbb{R}^2$:

$$\begin{aligned} -\nabla \cdot (\mu \nabla u) &= f, \qquad && \text{in } \Omega, \\ u &= g, && \text{on } \partial\Omega. \end{aligned} \tag{1}$$

The numerical discretization of problem (1) is performed by means of the PolyDG method described in [1]. As a first test case, we consider the domain $\Omega = (0,1)^2$ with homogeneous diffusion coefficient $\mu = 1$. The exact solution is $u(x,y) = \tanh\big(-20(x^2+y^2-0.8)\big)$, and the boundary condition and the forcing term are computed accordingly. To solve this problem, we use the routines contained in the `Laplacian` module, and the data are set in the file `DataComparisonpAdaptive.m`. We focus on the comparison between adaptive and uniform simulations using an adaptive procedure to dynamically update the local polynomial degree distribution up to $p_{\max} = 10$. We exploit different element meshes ($N_{\text{el}} = 180, 300, 800, 1500, 2200, 3000$). The p-adaptive indicator used in the simulation is characterized by the sum of a residual component and one associated with the jumps of the numerical solution and its gradient across the mesh edges (see [9] for details).

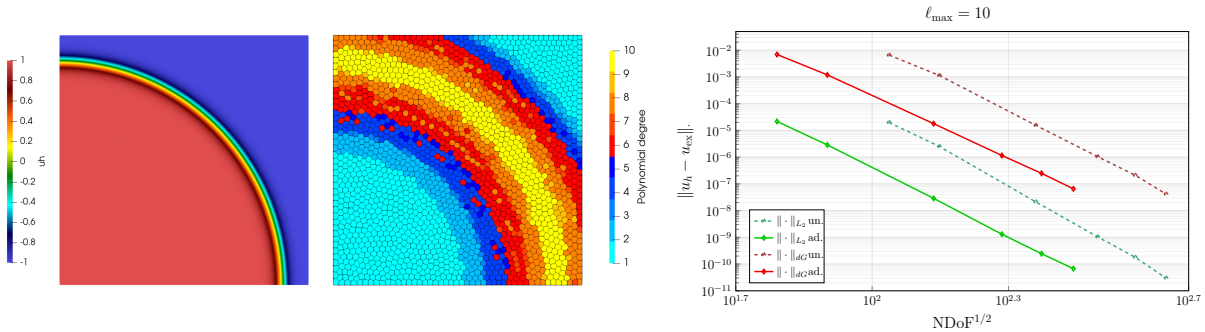


Figure 2: Poisson problem. Left: Numerical solution u_h and polynomial degree distribution with $p_{\max} = 10$. Right: Corresponding computer errors in the $\|u - u_h\|_{dG}$ and $\|u - u_h\|_{L^2(\Omega)}$ as a function of the total number of degrees of freedom of the system. The adaptive approach (solid line) achieves a reduction of approximately 38%–43% degrees of freedom compared to the uniform refinement (dashed line).

Figure 2 illustrates the behavior of the p-adaptive strategy compared with the uniform one. On the left, we show the computed numerical solution together with the computed element-wise polynomial degree distribution obtained with the adaptive algorithm compared to the uniform refinement (dashed line), which is obtained by exploiting a uniform polynomial degree. On the right, we show the computed errors as a function of the total number of degrees of freedom (NDoF) of the system in the L^2 dG-norms, where the latter is defined as:

$$\|u\|_{dG}^2 = \|\mu \nabla_h u\|_{\mathbf{L}^2(\Omega)}^2 + \|\sqrt{\eta}[\![u]\!]\|_{L^2(\mathscr{F})}^2. \tag{2}$$

Figure 2 shows that high polynomial degrees are automatically assigned along the transition layer. The convergence behavior is preserved. Here, the same level of accuracy is achieved with significantly fewer NDoF. This confirms that the p-adaptive approach is able to maintain the approximation properties of the uniform high-order discretization while substantially reducing computational cost.

5.2 Time-dependent problems

As a second example, we consider a general semilinear parabolic problem. Given an open, bounded, polygonal domain $\Omega \subset \mathbb{R}^d$, $(d = 2, 3)$ and a final time $T > 0$, we consider the solution $u : \Omega \times [0, T] \to \mathbb{R}$. The model reads as follows: for any time $t \in (0, T]$, find $u = u(\boldsymbol{x}, t)$ such that:

$$\frac{\partial u}{\partial t} - \nabla \cdot (\mu \nabla u) + f(u) = g_u, \qquad \text{in } \Omega, \tag{3}$$

complemented with suitable boundary and initial conditions $u(0) = u^0$. Moreover, $f(u)$ is a possibly nonlinear function of the solution, and g_u is the forcing term.

5.2.1 The heat equation

First, we consider the heat equation in (3) setting $f = 0$. Moreover, we consider $\Omega = (-1, 1)^2$ and $\mu = 1$. The exact solution is given by

$$u(x, y, t) = \exp\left(\frac{1 - \exp\left(\frac{x-ty}{\varepsilon}\right)}{1 - \exp\left(-\frac{1}{\varepsilon}\right)}\right),$$

where $\varepsilon = 8e{-}2$ is the parameter controlling the thickness of the resulting layer in the solution. The forcing term and the Dirichlet boundary conditions are set accordingly. To solve this problem, we use the routines contained in the `Heat` module using the data in `DataTestCasepAdaptive.m`. For this simulation, we impose $T = 3$ and $\Delta t = 10^{-2}$, with implicit Euler scheme discretization. The mesh is composed of 1000 polygonal elements ($h = 0.121$) with $p_{\max} = 5$. The numerical simulations have been performed as a serial job using the `Kami` cluster (40 computing nodes configured as follows: **CPU** 2× AMD EPYC 7413 24-Core Processor, **RAM** 512 GB) at the Department of Mathematics, Politecnico di Milano.

	Matrices Reassembly	Indicator Construction	RHS Assembly	Degree Distribution Computation	Numerical Solution Projection	Algebraic System Solving	Forcing Term Computation	**Total Time**
Uniform	-	-	0.54 s	-	-	193.59 s	143.93 s	**338.06 s**
% of total	-	-	*0.16%*	-	-	*57.27%*	*42.57%*	
Adaptive	27.54 s	30.47 s	0.23 s	0.29 s	11.47 s	62.07 s	83.93 s	**216.00 s**
% of total	12.75%	*14.11%*	*0.11%*	*0.13%*	*5.31%*	*28.74%*	*38.86%*	

Table 1: Heat equation. Computational times for uniform and adaptive strategies. Percentages are computed with respect to the total time of each strategy.

Table 1 reports the computational times for the uniform and p-adaptive strategies for this simulation. Although the adaptive approach introduces additional costs related to matrix reassembly, indicator construction, and degree distribution within the p-adaptive routine, it significantly reduces the time required to solve the linear system and to compute the forcing term of the problem. As a result, the overall computational time is reduced from 338.06 s to 216.00 s, corresponding to a gain of approximately 36.11%. Figure 3 shows the evolution of the total NDoFs during the simulation. The evolution highlights the effect of the adaptive strategy on the total number of degrees of freedom of the system. In particular, the adaptive approach provides a reduction of approximately 54% in the number of degrees of freedom with respect to the initial configuration with $p_{\max} = 5$.

5.2.2 The Fisher-Kologorov model

In this section, we consider the Fisher-Kolmogorov equation obtained from problem (3) by choosing $f(u) = -u(1 - u)$. This model is widely employed in biological wave propagation and represents an example of a nonlinear parabolic equation exhibiting traveling fronts and sharp transition layers.

For details about the discretization of the problem, we refer to [12]. The computational domain is $\Omega = [-1, 3] \times [0, 1]$, with final time $T = 4$. The coefficients are $\mu = 10^{-3}$, the polynomial degree is $p_{\max} = 3$, and the time step is $\Delta t = 10^{-2}$. The exact solution of the problem is set to $u(x, y, t) = 0.25[1 + \tanh(8 - \sqrt{(24\mu)^{-1}}\, x)]^2$. We exploit homogeneous Neumann boundary conditions as in [3], and the nonlinear term is treated by Picard iterations

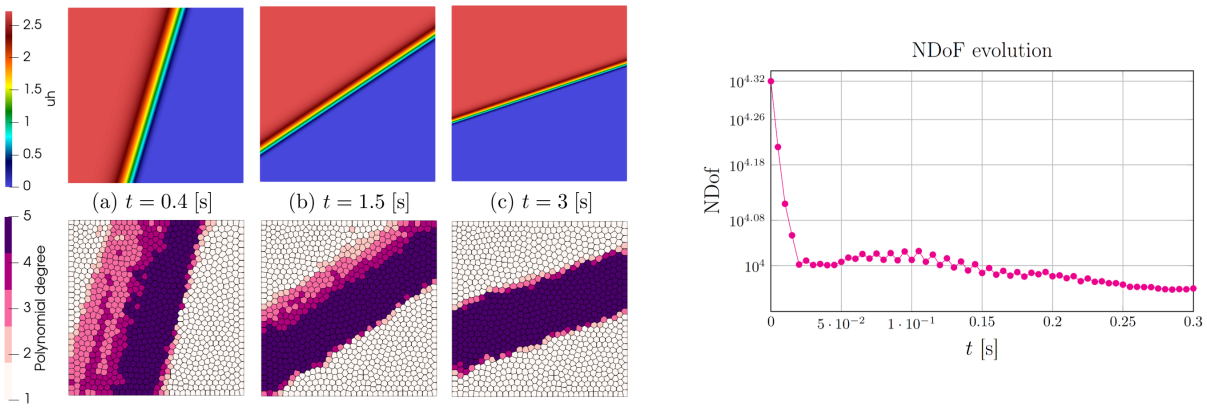


Figure 3: Heat equation. Evolution of the numerical solution and corresponding polynomial degree distribution at different time instants $t = 0.4\,\mathrm{s}$, $t = 1.5\,\mathrm{s}$, and $t = 3\,\mathrm{s}$.

with tolerance 10^{-10} and maximum number of iterations equal to 1000. Finally, adaptive polynomial refinement is employed. The adaptive algorithm is performed every 10 time steps, and at most two adaptation iterations are allowed. To solve this problem, we use the functions contained in `FisherKolmogorov`, and we set the data in `DataWavesFKPP.m`. Figure 4 (right) shows the numerical solution at different time instants $t = 5.0$ and $t = 20$ s, the corresponding polynomial degree distribution obtained through the adaptive procedure, and the distribution of the complete a posteriori indicator $\tau_K^{(n)}$. As expected, the indicator attains its largest values in correspondence with the sharp transition layer of the traveling wave, while remaining small in the smooth regions of the solution, so that the highest approximation orders are concentrated along the moving front. In Figure 4 (left), we report a convergence

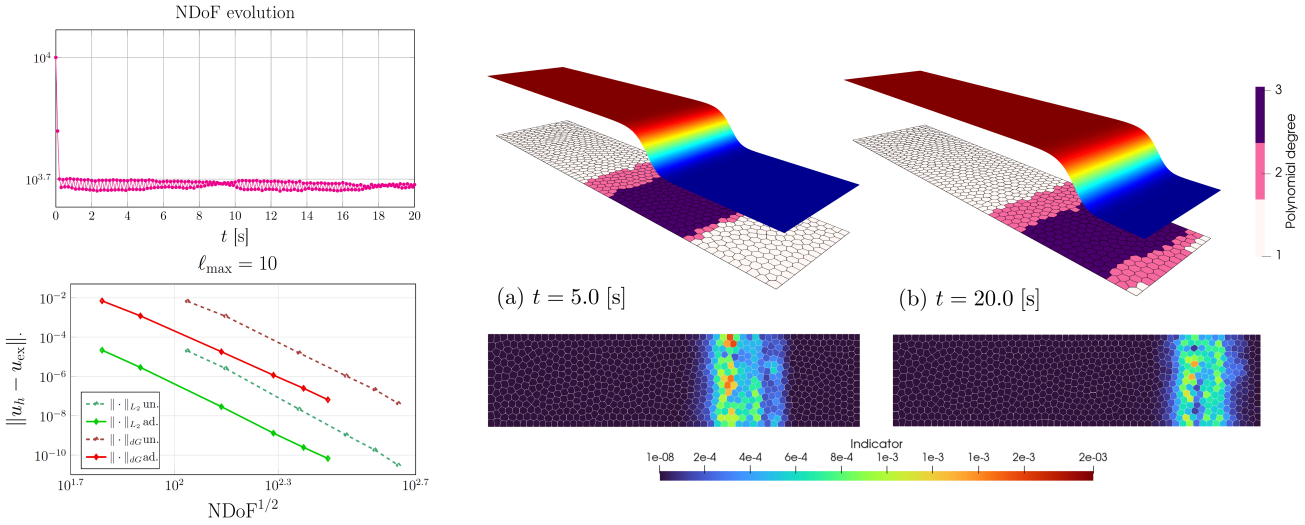


Figure 4: Fisher-Kolmogorov model. Left: temporal evolution of the total number of NDoFs during the p-adaptive simulation and convergence history in the L_2 and dG-norms with respect to NDoF$^{1/2}$. Right: numerical solution and spatial distribution of the local polynomial degree and a-posteriori indicator for each cell.

analysis for the uniform refinement with final time $T = 1$ and time step $\Delta t = 0.01$. Different mesh resolutions are considered, with a polynomial degree $p_{\max} = 3$. The convergence behavior of the uniformly enriched discretization is compared with that obtained through the adaptive strategy. As expected, the adaptive approach preserves the same level of accuracy achieved by the uniform approximation while significantly reducing the NDoFs.

5.2.3 The monodomain coupled with FizHugh-Nagumo model

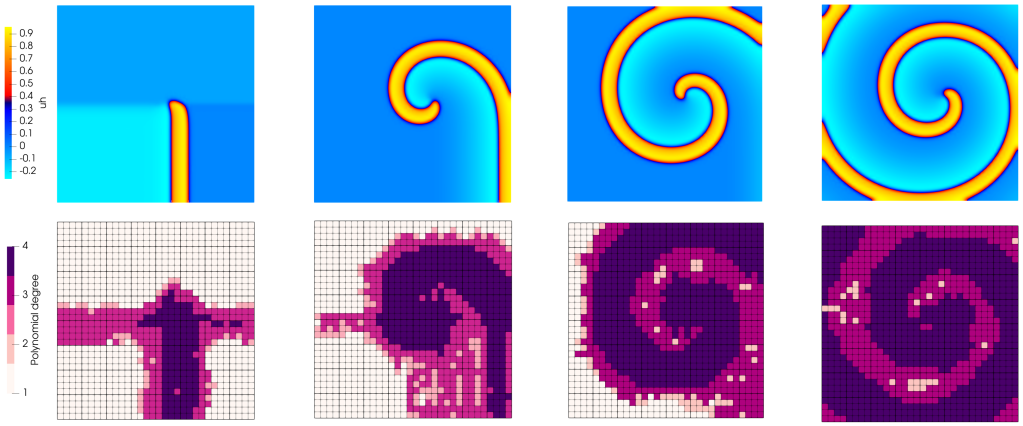
In this section, we consider the monodomain equation coupled with FitzHugh-Nagumo ionic model, which provides a simplified description of the electrophysiological behavior of biological tissues [14, 21]. Given an open, bounded,

polygonal domain $\Omega \subset \mathbb{R}^d$, $(d = 2, 3)$ and a final time $T > 0$, we consider the solution (u, w) with $u : \Omega \times [0, T] \to \mathbb{R}$, and $w = w(\boldsymbol{x}, t)$ with $w : \Omega \times [0, T] \to \mathbb{R}$. The model reads as follows: for any time $t \in (0, T]$, find $u = u(\boldsymbol{x}, t)$ and $w = w(\boldsymbol{x}, t)$ such that:

$$\begin{aligned} C\chi_m \frac{\partial u}{\partial t} - \nabla \cdot (\mu \nabla u) + \chi_m k u(u - a)(u - 1) &= g_u, \qquad \text{in } \Omega, \\ \frac{\partial w}{\partial t} + \varepsilon(\beta u - \gamma w) &= g_w, \qquad \text{in } \Omega. \end{aligned} \tag{4}$$

complemented with suitable boundary and initial conditions $u(0) = u^0$ and $w(0) = w^0$. With parameters $a \in (0, 1)$, $\varepsilon \ll 1$, and $\gamma > 0$, controlling the excitability and recovery dynamics of the system. In this test case, we consider $\Omega = (0, 2.5)^2$ with homogeneous Neumann boundary conditions. We set the model parameters as $\mu = 10^{-4}$, $\kappa = \gamma = 1$, $\varepsilon = 0.003$, $\beta = 0.5$, and $a = 0.1$. We adopt a Crank–Nicolson time-stepping scheme with final time $T = 3000$ and time step $\Delta t = 0.5$. We consider a Cartesian mesh of 1 024 elements and employ the p-adaptive DG approximation with a polynomial degree $p_{\max} = 4$. Adaptive updates are performed every 10 time steps, with at most 2 adaptive iterations. The resulting nonlinear algebraic system is solved through Picard iterations with tolerance 10^{-10} and maximum number of iterations equal to 1000. The initial conditions are chosen in order to trigger the formation and propagation of spiral-wave dynamics, as follows

$$u_0(x, y) = \begin{cases} 1, & x \le 1.25 \text{ and } y \le 1.25, \\ 0, & \text{otherwise}, \end{cases} \qquad w_0(x, y) = \frac{1}{20}\left(1 + \tanh\left(\frac{y - 1.25}{0.02}\right)\right).$$



(a) $t = 80$ [s] (c) $t = 250.0$ [s] (b) $t = 350$ [s] (d) $t = 2300.0$ [s]

Figure 5: FitzHugh-Nagumo model. Snapshots of the numerical solution u_h (top row) and corresponding polynomial-degree distribution (bottom row) at different time snapshot $t = 80,\ 250,\ 350,\ 2300\ s$.

To solve this problem, we use the `FitzHughNagumo` module and the data file `DataFHNSpiral.m`. Figure 5 shows the numerical solution of the FitzHugh–Nagumo system at four representative time instants. The first row shows the evolution of the activator variable u_h, while the second row reports the corresponding distribution of the local polynomial degree selected by the adaptive strategy. We observe that the highest polynomial degrees are located in proximity to the spiral wave.

5.3 The elastodynamic system

We now consider the linear elastodynamic equation: find the displacement field $\boldsymbol{u} : \Omega \times [0, T] \to \mathbb{R}^2$ such that

$$\rho \partial_{tt} \boldsymbol{u} - \nabla \cdot \boldsymbol{\sigma}(\boldsymbol{u}) = \boldsymbol{f} \qquad \text{in } \Omega \times (0, T],$$

together with proper boundary and initial conditions $\boldsymbol{u}(\cdot, 0) = \boldsymbol{u}_0$, $\partial_t \boldsymbol{u}(\cdot, 0) = \boldsymbol{u}_0$. The stress tensor is defined by the isotropic linear elastic constitutive relation $\boldsymbol{\sigma}(\boldsymbol{u}) = 2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}) + \lambda\, \mathrm{tr}(\boldsymbol{\varepsilon}(\boldsymbol{u}))\mathbf{I}$, where $\boldsymbol{\varepsilon}(\boldsymbol{u})$ is the symmetric strain tensor, $\mathbf{I}$ denotes the identity tensor, and $\lambda, \mu \in L^\infty(\Omega)$ are the Lamé coefficients. All the details about material parameters, model coefficients, boundary conditions and numerical discretization adopted in this work are reported

in [1, §5.3]. The numerical simulations have been performed as a serial job using the `Nemesis` cluster (**CPU** 2× AMD EPYC 9634 84-Core Processor (336 threads), **RAM** 1.5 TB) at the Department of Mathematics, Politecnico di Milano.

	Matrix Assembly (QF)	Matrix Assembly (ST)	RHS Assembly	Linear System Solver
Serial: 1 core	9.51 s	10.23 s	1.93 s	60.06 s
(% time saving)	*(-2.36%)*	*(-4.57%)*	*(-4.46%)*	*(-1.54%)*
Parallel: 4 cores	5.05 s	5.39 s	0.72 s	26.20 s
(% time saving)	*(-48.15%)*	*(-49.72%)*	*(-64.36%)*	*(-57.05%)*
Parallel: 8 cores	4.31 s	4.58 s	0.47 s	22.13 s
(% time saving)	*(-55.75%)*	*(-57.28%)*	*(-76.73%)*	*(-63.74%)*
`lymph` **1.0** [1]	9.74 s	10.72 s	2.02 s	61.00 s

Table 2: Elastodynamics equation. Computational times serial and parallel solvers and comparison with the `lymph` 1.0 version. With total number of dregrees of freedom of the system equal to 102270.

Table 2 reports the computational times for the different phases of the simulation, comparing serial and parallel runs with the new assembly framework against the previous `lymph` 1.0 implementation. The results show that the unified assembly routine benefits significantly from shared-memory parallelism: moving from a serial run to 4 and 8 cores yields a substantial reduction in the matrix assembly times, for both QF and ST strategies, and for the linear system solver time. Finally, the row corresponding to `lymph` 1.0 highlights that the new release achieves comparable times in serial, confirming the effectiveness of the redesigned driver, with important gains obtained when parallelization is enabled.

6 Conclusions

The new version of the `lymph` library introduces a set of advances that turn the original framework into a more powerful environment for high-order discontinuous Galerkin simulations on polygonal meshes. This release aims to improve the generality, modularity, parallelism, and efficiency of the solver by combining a novel assembly engine with an adaptive workflow. The combination of a unified assembly driver with enhanced parallel capabilities improves efficiency for large-scale and long-time simulations, while the reorganization of the postprocessing and input/output layers further streamlines the overall workflow. Moreover, the support for element-wise polynomial degrees and the p-adaptive strategies introduces flexible tools to tackle multiscale and strongly heterogeneous problems with a tightly controlled computational cost. Altogether, these developments significantly extend the scope of `lymph` and strengthen its role as a reliable platform for the development and prototyping of high-order discontinuous Galerkin discretizations on polytopal grids.

Acknowledgments

We gratefully acknowledge Stefano Bonetti, Michele Botti, Ivan Fumagalli and Ilario Mazzieri for their careful review and testing of the code released alongside this paper. Their feedback was invaluable in improving the quality, reliability, and usability of the software.

References

[1] P. F. Antonietti, S. Bonetti, M. Botti, M. Corti, I. Fumagalli, and I. Mazzieri. lymph: discontinuous poLYtopal methods for Multi-PHysics differential problems. *ACM Trans. Math. Softw.*, 51(1):1–22, 2025.

[2] P. F. Antonietti, M. Botti, A. Cancrini, and I. Mazzieri. A polytopal discontinuous Galerkin method for the pseudo-stress formulation of the unsteady Stokes problem. *Comput. Methods Appl. Mech. Engrg.*, 447:118404, 2025.

[3] P. F. Antonietti, M. Corti, S. Gómez, and I. Perugia. A structure-preserving LDG discretization of the Fisher-Kolmogorov equation for modeling neurodegenerative diseases. *Math. Comput. Simul.*, 241:351–366, 2026.

[4] P. F. Antonietti, S. Giani, and P. Houston. hp-version composite discontinuous Galerkin methods for elliptic problems on complicated domains. *SIAM J. Sci. Comput.*, 35(3):A1417–A1439, 2013.

[5] P. F. Antonietti, P. Houston, and G. Pennesi. Fast numerical integration on polytopic meshes with applications to discontinuous Galerkin finite element methods. *J. Sci. Comput.*, 77(3):1339–1370, Dec. 2018.

[6] L. Beirão da Veiga, G. Manzini, and L. Mascotto. A posteriori error estimation and adaptivity in *hp* virtual elements. *Numer. Math.*, 143(1):139–175, 2019.

[7] S. Bonetti and M. Corti. Unified discontinuous Galerkin analysis of a thermo/poro-viscoelasticity model. *J. Sci. Comput.*, 105:11, 2025.

[8] M. Botti, I. Fumagalli, and I. Mazzieri. Polytopal discontinuous Galerkin methods for low-frequency poroelasticity coupled to unsteady Stokes flow. *Eng. Comput.*, 41:4173–4189, 2025.

[9] A. Cangiani, Z. Dong, and E. H. Georgoulis. A posteriori error estimates for discontinuous Galerkin methods on polygonal and polyhedral meshes. *SIAM J. Numer. Anal.*, 61(5):2352–2380, 2023.

[10] A. Cangiani, Z. Dong, E. H. Georgoulis, and P. Houston. *hp-Version Discontinuous Galerkin Methods on Polygonal and Polyhedral Meshes*. Springer, 2017.

[11] A. Cangiani, E. H. Georgoulis, and P. Houston. hp-version discontinuous Galerkin methods on polygonal and polyhedral meshes. *Math. Models Methods Appl. Sci.*, 24(10):2009–2041, 2014.

[12] M. Corti, F. Bonizzoni, L. Dede', A. M. Quarteroni, and P. F. Antonietti. Discontinuous Galerkin methods for Fisher–Kolmogorov equation with application to α-synuclein spreading in Parkinson's disease. *Comput. Methods Appl. Mech. Eng.*, 417:116450, 2023.

[13] R. A. Fisher. The wave of advance of advantageous genes. *Ann. Eugen.*, 7:355–369, 1937.

[14] R. FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophys. J.*, 1(6):445–466, 1961.

[15] P. Houston, B. Senior, and E. Süli. A posteriori error analysis for hp-version discontinuous Galerkin methods for second-order quasilinear elliptic problems. *IMA J. Numer. Anal.*, 22(4):547–573, 2002.

[16] P. Houston and T. P. Wihler. An *hp*-adaptive Newton-discontinuous-Galerkin finite element approach for semilinear elliptic boundary value problems. *Math. Comp.*, 87(314):2641–2674, 2018.

[17] A. Kolmogorov, I. Petrovsky, and N. Piscounov. Study of the diffusion equation with growth of the quantity of matter and its application to a biological problem. *Bull. Moscow Univ. Math. Mech.*, pages 1–25, 1937.

[18] C. B. Leimer Saglio, S. Pagani, and P. F. Antonietti. A p-adaptive polytopal discontinuous Galerkin method for high-order approximation of brain electrophysiology. *Comput. Methods Appl. Mech. Eng.*, 446:118249, 2025.

[19] C. B. Leimer Saglio, S. Pagani, M. Corti, and P. F. Antonietti. A high-order discontinuous Galerkin method for the numerical modeling of epileptic seizures. *Comput. Math. Appl.*, 205:112–131, 2026.

[20] J. M. Melenk and B. I. Wohlmuth. On residual-based a posteriori error estimation in *hp*-FEM. *Adv. Comput. Math.*, 15(1–4):311–331, 2001.

[21] J. Nagumo, S. Arimoto, and S. Yoshizawa. An active pulse transmission line simulating nerve axon. *Proc. IRE*, 50(10):2061–2070, 1962.

[22] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit (4th ed.)*. Kitware, 2006.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

51/2026 Bellezza P.; Ciaramella G.; Macchini C.; Mazzieri I.; Verani M.
ParaFlow: Parareal Acceleration of Gradient-Flow Minimization

52/2026 Bonazzoli M.; Ciaramella G.; Mazzieri I.
On the Unmapped Tent Pitching for the Heterogeneous Wave Equation

53/2026 Dong Z., Jiang Y., Ng M., Ciaramella G., Yin J.
Chebyshev-Filtered Randomized Low-rank Preconditioners for Symmetric Positive Definite Linear Systems

50/2026 Donnarumma, A.; Guagliardi, O.; Di Stazio F.; Mazza E.; Tanelli M.; Paganoni A.M.
Modelling Well-Being and Psychological Risk in Doctoral Education: An Integrated Latent Trait Approach

49/2026 Bortolotti, T.; Troilo, R.; Casu, F.; Vantini, S.; Menafoglio, A.
Regularized covariance estimation from partially observed interferometric data

48/2026 Antonietti, P.F.; Corti, M.; Orlando, G.
Optimized high-order IMEX-RK schemes for degenerate diffusion-reaction problems with application to travelling waves phenomena

46/2026 Cancrini, A.; Ciaramella, G.; Antonietti, P.F.
A Scalable Deflated Conjugate Gradient Solver for the Time-Dependent Pseudo-Stress Stokes Problem

47/2026 Torri, V.; Barbieri, E.; Cantarutti, A.; Giaquinto, C.; Ieva, F.
Automatic identification of diagnosis from hospital discharge letters via weakly supervised Natural Language Processing

45/2026 Antonietti, P.F.; Botti, M.; Parolini, N.; Pederzoli, V.; Verani, M.
Polytopal Discontinuous Galerkin Discretizations of Coupled Non-Newtonian Stokes-Darcy Systems

44/2026 Bonetti, S; Botti, M.; Antonietti, P.F.
Splitting strategies for the fully-coupled nonlinear thermo-hydro-mechanical problem