Numerical Algorithms (2025) 99:1393–1428
https://doi.org/10.1007/s11075-024-01919-4

RESEARCH

VEMcomp: a Virtual Elements MATLAB package for bulk-surface PDEs in 2D and 3D

Massimo Frittelli[1] · **Anotida Madzvamuse**[2,3,4,5] · **Ivonne Sgura**[1]

Received: 30 January 2024 / Accepted: 11 August 2024 / Published online: 31 August 2024

Abstract

We present a Virtual Element MATLAB solver for elliptic and parabolic, linear and semilinear Partial Differential Equations (PDEs) in two and three space dimensions, which is coined VEMcomp. Such PDEs are widely applicable to describing problems in material sciences, engineering, cellular and developmental biology, among many other applications. The library covers linear and nonlinear models posed on different simple and complex geometries, involving time-dependent bulk, surface, and bulk-surface PDEs. The solver employs the Virtual Element Method (VEM) of lowest polynomial order $k = 1$ on general polygonal and polyhedral meshes, including the Finite Element Method (FEM) of order $k = 1$ as a special case when the considered mesh is simplicial. VEMcomp has three main purposes. First, VEMcomp generates polygonal and polyhedral meshes optimized for fast matrix assembly. Triangular and tetrahedral meshes are encompassed as special cases. For surface PDEs, VEMcomp is compatible with the well-known Matlab package DistMesh for mesh generation. Second, given a mesh for the considered geometry, possibly generated with an external package, VEMcomp computes all the matrices (mass and stiffness) required by the VEM or FEM method. Third, for multiple classes of stationary and time-dependent bulk, surface and bulk-surface PDEs, VEMcomp solves the considered PDE problem with the VEM or FEM in space and IMEX Euler in time, through a user-friendly interface. As an optional post-processing, VEMcomp comes with its own functions for plotting the numerical solutions and evaluating the error when possible. An extensive set of examples illustrates the usage of the library.

Keywords Virtual element method · Bulk-surface virtual element method · Bulk-surface finite element method · Bulk-surface PDEs · Mesh generation · IMEX Euler Method · MATLAB

Massimo Frittelli, Anotida Madzvamuse and Ivonne Sgura contributed equally to this work

Extended author information available on the last page of the article

1 Introduction

The Virtual Element Method (VEM) was first proposed in [1] for elliptic problems in two space dimensions as a generalization of the Finite Element Method (FEM), where the mesh elements can be general polygons instead of triangles. The usage of polygons with arbitrary number of edges is made possible by enriching the local space of polynomials with suitable non-polynomial functions defined as solutions of an element-wise problem. This elegant idea ensures the optimal polynomial accuracy of the method. The immediate success of VEM is due to the multiple benefits of its geometric flexibility. Among such benefits, we mention: (i) efficient mesh refinement techniques [2, 3], (ii) numerical solutions with high global regularity [4–6], (iii) accurate approximation of boundaries [7–11], (iv) easy mesh pasting [12, 13], and (v) easy handling of complex domain shapes and cuts [10, 14].

Motivated by its multiple advantages, the VEM was quickly extended to numerous Partial Differential Equations (PDE) problems and applications. A non-exhaustive list of models for which the VEM has been employed comprises of (i) linear elliptic problems in two [1, 7] and three [15] space dimensions, (ii) semilinear elliptic problems in two and three space dimensions [16], (iii) linear heat equation in two [17] and three [18] space dimensions, (iv) semilinear parabolic equations [19] and reaction-diffusion systems [20], (v) elasticity [21, 22] and plasticity [23] problems, (vi) phase-field models [6, 24, 25], (vii) fluid dynamics [26, 27], (viii) fracture models [14], (ix) surface [13, 28] and bulk-surface [29–32] PDEs, and recently (x) PDEs on evolving flat domains [33].

Over the last ten years, VEM has established itself as a reliable numerical method with desirable properties for solving PDEs. This has stimulated the development of the first open-source VEM libraries and codes. Here we will recall some of the current state-of-the-art libraries and then outline the key contributions of VEMcomp to existing libraries as well as its major differences and improvements. The work in [34] presents a MATLAB implementation of the baseline VEM application: the lowest order VEM for the Poisson problem in 2D. For the same problem, a high order VEM code in MATLAB is then provided in [35]. An Abaqus-MATLAB VEM code for coupled thermo-elasticity problems in 2D is presented in [36]. VEM libraries for elasticity problems in 2D are available in MATLAB/Octave [37] and C++ [38]. For PDE problems in three space dimensions (3D), we mention the MATLAB library mVEM [39] for the Poisson, Stokes equations, linear elasticity and friction problems, and the C++/Python library VEM3D [40] for the Laplace equation. All the mentioned libraries are dedicated to specific cases or applications and are confined to stationary bulk-only problems. The state of the art of VEM libraries is schematically reviewed in Fig. 1.

To the best of the authors' knowledge, there is no open-source VEM library so far for time-dependent PDE problems in two nor three space dimensions nor for surface or bulk-surface PDE problems in 2D or 3D. In this work, we contribute to the fields of VEM and FEM open-source libraries. We propose a MATLAB library, which we call VEMcomp, for linear elliptic and linear and nonlinear parabolic PDE problems in 2D and 3D, including bulk, surface and bulk-surface PDE models. VEMcomp has four purposes:

CODES FOR VEM: CURRENT STATE OF THE ART

Library	2d	3d	PDE: S=stationary T=evolutionary	Bulk dom	Surface & Bulk/Surface domain	Language	Open source	Mesh	Paper/ Reference
Vem3d (2016)			Poisson BC: Dir			C++/ Python		Own	**MSc Thesis PoliMI** https://github.com/deatinor/VEM3D
50lines (2016)			Poisson BC: Dir			Matlab		Own	**Num Alg** doi.org/10.1007/s11075-016-0235-3
VEM_in_Abaqus (2019)			Coupled Thermo-Elasticity, BC: Dir/Neu			Matlab/ Fortran		Distmesh	**Num Alg** doi.org/10.1007/s11075-018-0516-0
Veamy (2019)			Poisson, linear elastostatic, BC: Dir/Neu			C++		Delynoi	**Num Alg** doi.org/10.1007/s11075-018-00651-0
VEM2d (2021)			Poisson, BC: Dir order $k \geq 1$			Matlab		Own	**Num Alg** doi.org/10.1007/s11075-022-01361-4
mVEM (2022)			Stokes, Mixed Darcy BC: Dir, Neu,Var Ineq			Matlab		Own+ Polymesher	**ArXiv:** 2204.01339v1
VEMLAB (2023)			Poisson, linear elastostatics BC: Dir/Neu			Matlab		Polymesher distmesh	**camlab.cl/software/vemlab/**
VEMcomp (2024)			*S & T*: Reaction-diffusion, BC: Dir/ Neu			Matlab		Own+ distmesh	**Num Alg**

Fig. 1 State of the art of the existing VEM solvers

1. **Mesh generation**: For 2D and 3D domains represented as level sets, the library generates polytopal (i.e. polygonal in 2D and polyhedral in 3D) meshes specifically optimized for fast matrix assembly, following [29, 30]. Regardless of the space dimension, the surface mesh is always taken as the boundary of the bulk mesh.
2. **Matrix assembly**: Given any polygonal or polyhedral mesh – not necessarily generated with VEMcomp itself –, the library generates all the matrices involved in the VEM and FEM methods (e.g. mass, stiffness). For example, for the case of triangulated surfaces in $\mathbb{R}^3$ with an empty boundary, VEMcomp is fully compatible with the triangulated surfaces generated by the well-known Matlab package DistMesh [41];
3. **Black-box solvers**: For multiple classes of PDE problems, the library provides a black-box interface that allows the user to set the problem parameters, and returns the VEM or FEM numerical solution. For time-dependent problems, the time discretization is carried out with the IMplicit-EXplicit (IMEX) Euler method, which has been proven to be simple and effective in combination with FEMs and VEMs for surface [42, 43] and bulk-surface PDEs [29, 30];
4. **Post-processing**: The library can, through inhouse functions, plot the numerical solution and compute the relative error in L^2 norm if the exact solution is known in closed form, without having to resort to external software.

For the sake of clarity, a sketch of the VEMcomp structure together with references to the corresponding sections in the main text of the paper, is reported in Fig. 2.

The structure of our paper is as follows. In Section 2, we state the multiple model problems to be addressed in this work, thereby motivating the functionalities of VEMcomp. In Section 3, we illustrate how VEMcomp generates polytopal (i.e. polygonal in 2D and polyhedral in 3D) meshes, including simplicial (i.e. triangular in 2D and tetrahedral in 3D) meshes. Section 4 showcases VEMcomp's ability to compute local and

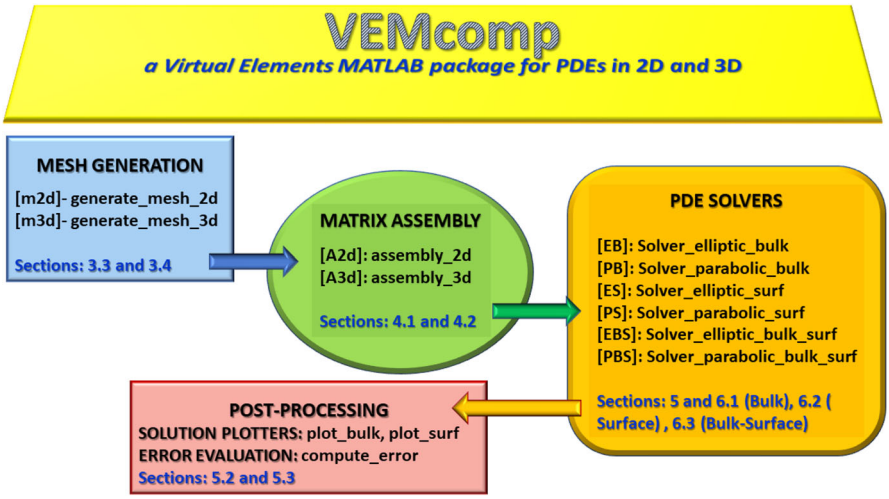

Fig. 2 Schematic representation of VEMcomp's three facilities: (i) polytopal mesh generation, (ii) matrix assembly and (iii) built-in solvers. Each of these facilities is composed of functions that execute specific tasks, as shown in the picture

global VEM and FEM matrices. In Section 5, we present VEMcomp's user-friendly solvers, solution plotters, and error evaluation functions for the model problems outlined in Section 2. Section 6 lists several numerical examples that illustrate at once the usage of VEMcomp. In Section 7, we draw our conclusions and outline future research directions.

2 Overview

VEMcomp is an object-oriented VEM library written in MATLAB. Compared to other existing VEM libraries, VEMcomp aims to fill the gap for PDE problems (i) with time-dependence and (ii) on complex geometries, where surface or bulk-surface PDEs are posed. In the remainder of this section, let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, be a compact domain in the d-dimensional Euclidean space. The first class of PDE problems covered by VEMcomp comprises linear elliptic bulk-only PDEs, and is given by

$$\begin{cases} -d^{\Omega}\Delta u + \alpha u = f(\boldsymbol{x}), & \boldsymbol{x} \in \Omega; \\ u(\boldsymbol{x}) = 0 \quad \text{or} \quad \dfrac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{1}$$

where Δ denotes the Laplace operator in Ω, $d^{\Omega} > 0$ is a positive diffusion coefficient, $\alpha \geq 0$ is a nonnegative coefficient, and f is a sufficiently smooth source term. As a time-dependent counterpart of (1), VEMcomp covers semilinear parabolic bulk-only

PDE systems of $n \in \mathbb{N}$ equations of the following form

$$\begin{cases} \dfrac{\partial u_i}{\partial t} - d_i^{\Omega} \Delta u_i = f_i(u_1, \dots, u_n, \boldsymbol{x}, t), \quad (\boldsymbol{x}, t) \in \Omega \times [0, T], \\ u_i(\boldsymbol{x}, t) = 0 \quad \text{or} \quad \dfrac{\partial u_i}{\partial \boldsymbol{n}}(\boldsymbol{x}, t) = 0, \quad (\boldsymbol{x}, t) \in \partial\Omega \times [0, T], \\ u_i(\boldsymbol{x}, 0) = u_{i,0}(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega, \end{cases} \tag{2}$$

for $i = 1, \dots, n$, where $d_1^{\Omega}, \dots, d_n^{\Omega} > 0$ are diffusion coefficients, $T > 0$ is the final time, $f_1, \dots, f_n$ are smooth enough linear or nonlinear functions, and $u_{1,0}, \dots, u_{n,0}$ are sufficiently smooth initial data. The general models (1) and (2) comprise several notable bulk-only PDE problems that were solved with VEM in the literature, such as (i) linear [1, 7, 15] and semilinear elliptic problems [16], (ii) linear [17] and semilinear parabolic problems [19] including reaction-diffusion systems [20].

If $\Gamma = \partial\Omega$ is a sufficiently smooth manifold with an empty boundary, a second class of PDEs that fall in VEMcomp's framework is the following class of linear elliptic surface PDEs:

$$-d^{\Gamma} \Delta_{\Gamma} v + \beta v = g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma, \tag{3}$$

where Δ_{Γ} represents the Laplace-Beltrami operator on Γ, $d^{\Gamma} > 0$ is a diffusion coefficient, $\beta > 0$ is a positive coefficient, and g is a regular enough source term. As a time-dependent counterpart of (3), VEMcomp solves the following class of semilinear parabolic surface PDEs or systems of $m \in \mathbb{N}$ surface PDEs:

$$\begin{cases} \dfrac{\partial v_j}{\partial t} - d_j^{\Gamma} \Delta_{\Gamma} v_j = g_j(v_1, \dots, v_n, \boldsymbol{x}, t), \qquad \boldsymbol{x} \in \Gamma, \\ v_j(\boldsymbol{x}, 0) = v_{j,0}(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma, \end{cases} \tag{4}$$

for $j = 1, \dots, m$, where Δ_{Γ} represents the Laplace-Beltrami operator on Γ, $d_1^{\Gamma}, \dots, d_m^{\Gamma} > 0$ are diffusion coefficients, $g_1, \dots, g_m$ are smooth enough linear or nonlinear functions, $T > 0$ is the final time and $v_{1,0}, \dots, v_{m,0}$ are sufficiently smooth initial data. In (3) and (4), no boundary conditions are needed since, as we mentioned, Γ has no boundary. The general models (3) and (4) encompass several surface PDE (SPDE) models of interest, such as elliptic SPDEs [13, 28] and surface reaction-diffusion systems (SRDSs) [44, 45].

The third and most complex class of PDE problems solvable by VEMcomp is given by bulk-surface PDEs. In the stationary case, we consider linear elliptic coupled bulk-surface PDEs of the following form:

$$\begin{cases} -d^{\Omega} \Delta u + \alpha u = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega; \\ -d^{\Gamma} \Delta_{\Gamma} v + \beta v = g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma; \\ \dfrac{\partial u}{\partial \boldsymbol{n}} = \gamma u + \delta v, \qquad \boldsymbol{x} \in \Gamma, \end{cases} \tag{5}$$

where $d^{\Omega}, d^{\Gamma} > 0$ are diffusion coefficients, $\alpha, \beta \geq 0$ are nonnegative coefficients, $\gamma, \delta \geq 0$ are nonnegative coupling coefficients, and f and g are regular enough source terms. As a time-dependent counterpart of (5), VEMcomp solves the following coupled bulk-surface reaction-diffusion system (BS-RDS):

$$\begin{cases} \dfrac{\partial u_i}{\partial t} - d_i^{\Omega} \Delta u_i = f_i(u_1, \ldots, u_n, \boldsymbol{x}, t), & \boldsymbol{x} \in \Omega; \\ \dfrac{\partial v_j}{\partial t} - d_j^{\Gamma} \Delta_{\Gamma} v_j = g_j(v_1, \ldots, v_m, \boldsymbol{x}, t), & \boldsymbol{x} \in \Gamma; \\ \dfrac{\partial u_i}{\partial \boldsymbol{n}} = h_i(u_1, \ldots, u_n, v_1, \ldots, v_m, \boldsymbol{x}, t), & \boldsymbol{x} \in \Gamma; \\ u_i(\boldsymbol{x}, 0) = u_{i,0}(\boldsymbol{x}), & \boldsymbol{x} \in \Omega; \\ v_j(\boldsymbol{x}, 0) = v_{j,0}(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma, \end{cases} \tag{6}$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, where $d_1^{\Omega}, \ldots, d_{\Omega}^{n}, d_1^{\Gamma}, \ldots, d_m^{\Gamma} > 0$ are diffusion coefficients, $f_1, \ldots, f_n, g_1, \ldots, g_m, h_1, \ldots, h_n$ are smooth enough linear or nonlinear functions, $T > 0$ is the final time and $u_{1,0}, \ldots, u_{n,0}, v_{1,0}, \ldots, v_{m,0}$ are sufficiently smooth initial data. We recall that the VEM was extended to bulk-surface reaction-diffusion systems (BS-RDSs) in two [29] and three [30] space dimensions, which fall within the general class (6). These results further motivate our work.

In the next Sections we will illustrate in detail the three main facilities of VEMcomp: (i) polygonal and polyhedral mesh generation, (ii) computation of local VEM matrices and matrix assembly, and (iii) black-box solvers and solution plotters for problems (1)-(6). Each of these three facilities is composed of functions that execute specific tasks. A schematic representation of VEMcomp's facilities and their respective functions is shown in Fig. 2. VEMcomp is compatible with MATLAB R2019a and higher.

3 Element representation and mesh generation

Polygonal and polyhedral mesh generation is a niche topic, and very few off-the-shelf and easy to use software packages are available. For domains in two space dimensions, we mention the software PolyMesher [46] and Delynoi [47]. For domains in three space dimensions, we mention the software Polylla [48] and the MATLAB toolbox voronoi3d [49]. To the best of the authors' knowledge, there is no open-source software for bulk-surface mesh generation in two or three space dimensions. For 2D and 3D domains in level set form, VEMcomp fills this gap. We point out that, when restricted to triangular (in 2D) or tetrahedral meshes (in 3D), the Virtual Element Method (VEM) of low polynomial order $k = 1$ boils down to the Finite Element Method (FEM). This means that VEMcomp can be used as a FEM solver for surface and bulk-surface PDEs, when provided with triangular/tetrahedral meshes, for which several open-source generators exist, for instance the Matlab package DistMesh [41] as illustrated in the numerical example in Section 6.2.2. We start by presenting basic classes that allow to represent single elements in two and three space dimensions.

3.1 The class `element2d`

The class `element2d` represents a polygonal element in 2D. It contains minimal information that uniquely identify the element. To create an instance, say `obj`, of the class `element2d`, use one of the following constructors:

```
obj = element2d(P);
obj = element2d(P, is_square);
obj = element2d(P, is_square, is_boundary);
obj = element2d(P, is_square, is_boundary, Pind);
obj = element2d(P, is_square, is_boundary, Pind, P0);
```

where the inputs are defined as follows:

- `P` is a `NVert` $\times$ 3 array containing the coordinates of the vertexes, ordered clockwise or counterclockwise. The vertexes have three coordinates, because two-dimensional elements are also faces of three-dimensional elements.
- The Boolean `is_square` determines if `obj` is a square (for which the local VEM matrices are known in closed form).
- The Boolean `is_boundary` determines if `obj` is a face of a three-dimensional element lying on the boundary Γ of the bulk domain Ω. This information is necessary for the assembly of global VEM matrices;
- If the element `obj` is part of a mesh, and the coordinates of all the nodes are stored into an array, say `PP`, of size `NMesh` $\times$ 3, then the property `Pind` contains the indexes of `obj.P` in `PP`, i.e. `PP(Pind,:) = obj.P`. This information is crucial for matrix assembly;
- `P0` is a 3×1 array that contains the coordinates of a point w.r.t. which the element `obj` is star-shaped. VEMcomp needs this information later for the computation of local mass and stiffness matrices. However, the condition that each 2D element is star-shaped is not restrictive, as it is a basic assumption in the VEM literature, see [50].

Once an instance `obj` of the class `element2d` has been created, its properties can be accessed using the dot syntax, e.g. `P = obj.P`.

3.2 The class `element3d`

Analogous to the class `element2d`, the class `element3d` represents a polyhedral element in 3D. It contains minimal information that uniquely identify the element. To create an instance, say `obj`, of the class `element3d`, use one of the following constructors:

```
obj = element3d(P,Faces);
obj = element3d(P,Faces,is_cube);
obj = element3d(P,Faces,is_cube,Pind);
obj = element3d(P,Faces,is_cube,Pind,P0);
```

where the inputs are defined as follows:

- `P` is a `NVert` $\times$ 3 array containing the coordinates of the vertexes in any order;
- `Faces` is a `NFaces` $\times$ 1 array of instances of the class `element2d`, representing the faces of `obj`;
- The Boolean `is_cube` determines if `obj` is a cube (for which the local VEM matrices are known in closed form).
- If the element `obj` is part of a mesh, and the coordinates of all the nodes are stored in an array, say `PP`, of size `NMesh` $\times$ 3, then the property `Pind` contains the indexes of `obj.P` in `PP`, i.e. `PP(Pind,:) = obj.P`;
- `P0` is a 3×1 array that contains the coordinates of a point w.r.t. which the element `obj` is star-shaped. As in the 2D case, this information is needed for the computation of local and global mass and stiffness matrices, and the assumption that each 3D element is star-shaped is common in the VEM literature, see [15].

3.3 Mesh generation in 2D

We can now state that any polygonal mesh in 2D can be represented as a collection of `element2d`. Even if the user is free to write custom code to generate meshes as collections of `element2d`, VEMcomp comes with a built-in function for the generation of meshes in this format, for domains that are defined as level sets of Lipschitz functions as demonstrated next. Let $Q := [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \subset \mathbb{R}^2$ be a compact rectangle and let $f : Q \to \mathbb{R}$ be a Lipschitz function. Let $\Omega \subset \mathbb{R}^2$ and $\Gamma = \partial\Omega$ be defined respectively as

$$\Omega = \{\boldsymbol{x} \in Q \mid f(\boldsymbol{x}) \leq 0\}, \quad \text{and} \quad \Gamma = \{\boldsymbol{x} \in Q \mid f(\boldsymbol{x}) = 0\}. \tag{7}$$

The rectangle Q is subdivided with a Cartesian grid composed of rectangular elements. Then, the rectangles that intersect the boundary Γ are cut. This well-known algorithm, called *marching squares*, produces a piecewise linear approximation Γ_h of the boundary Γ [51]. However, our purpose is to produce a mesh for both the bulk Ω and the surface Γ. To this end, the rectangles and the cut rectangles produced as a by-product of the marching squares algorithm, constitute a polygonal approximation Ω_h of the bulk Ω, such that the approximate boundary Γ_h is exactly the boundary of Ω_h, i.e. $\Gamma_h = \partial\Omega_h$, see Fig. 3 for an illustration. This property is typical of simplicial (i.e. triangular and tetrahedral) meshes generated with well-assessed packages like FEniCS [52] or deal.II [53]. The approach proposed here generalizes the strategy proposed in our previous work [29], which was confined to spherical domains, to general level-set domains. In fact, this new approach does not require the projection of nodes onto the surface Γ. VEMcomp generates conforming bulk-surface meshes in 2D as described above through the function `generate_mesh2d`, whose syntax is as follows:

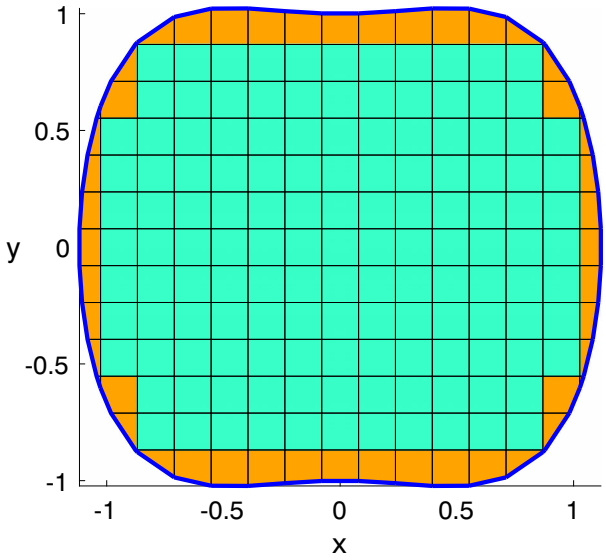

Fig. 3 A conforming bulk-surface mesh in 2D generated with the VEMcomp function `generate_mesh2d`. The bulk domain Ω is approximated by a polygonal mesh Ω_h composed of square elements (green) and more general polygonal elements (orange) close to the surface Γ. The surface Γ is approximated by a piecewise straight line Γ_h (blue)

```
[P, h, BulkElements, SurfElements] = ...
    generate_mesh2d(fun, Q, Nx, tol);
```

In the above, the inputs and outputs are defined as follows. In input:

- `fun` is the level function used in (7), represented as an inline function of the type `fun = @(P) <expression>`, where `P` is any $n \times 2$ array of $n \in \mathbb{N}$ points in $\mathbb{R}^2$;
- `Q` is the 2×2 array introduced in (7), which in Matlab can be created using the syntax $Q = [x_{\min}, x_{\max}; y_{\min}, y_{\max}]$;
- `Nx` $>= 2$ is the required amount of gridpoints along each dimension. If Q is not a square, then the shortest side of Q is discretised with `Nx` gridpoints, such that the resulting grid is equally spaced;
- `tol` > 0 is the minimum distance between distinct nodes. Any two nodes distant from each other less than `tol` will be merged into a unique node, in order to avoid elements with excessively short edges.

In output:

- `P` is a $N \times 2$ array containing the $N \in \mathbb{N}$ nodes of the bulk mesh Ω_h. We remark that the nodes of the surface mesh Γ_h constitute a subset of `P`;
- `h` > 0 is the meshsize of the bulk mesh Ω_h. Since the surface mesh Γ_h is the boundary of the bulk mesh Ω_h, then h is also an upper bound of the meshsize of the surface mesh Γ_h;
- `BulkElements` is a list of the bulk elements in `element2d` format;
- `SurfElements` is a $M \times 2$ array, where $M \in \mathbb{N}$ is the number of surface elements and, for each $i = 1, \ldots, M$, the nodes of the i-th surface element (a segment) are `P(SurfElements(i,:),:)`.

We remark that the marching squares algorithm does not guarantee shape regularity [51]. This means that shape regularity of the obtained mesh does not depend monotonically on the discretisation level N_x. This unpredictable behavior also depends on the shape of the domain under consideration. As a workaround, the user can try different values of the discretisation level N_x and different sizes for the bounding box Q. To the best of our knowledge, regular mesh generation for general level-set domains is an open problem. We also remark that VEMcomp can be used in combination with an externally generated mesh, if converted to the format returned by the function `generate_mesh2d` described above. Finally, it must be noted that `generate_mesh2d` is guaranteed to generate star-shaped elements, which is essential later for matrix assembly, see Section 3.1.

3.4 Mesh generation in 3D

In a similar way, we address mesh generation in 3D. Let $Q := [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}] \subset \mathbb{R}^3$ be a compact cuboid and let $f : Q \to \mathbb{R}$ be a Lipschitz function. Let $\Omega \subset \mathbb{R}^3$ and $\Gamma = \partial\Omega$ be defined respectively as

$$\Omega = \{\boldsymbol{x} \in Q \mid f(\boldsymbol{x}) \leq 0\}, \quad \text{and} \quad \Gamma = \{\boldsymbol{x} \in Q \mid f(\boldsymbol{x}) = 0\}. \tag{8}$$

The well-known *marching cubes* algorithm [51] is often used to generate triangulated surface meshes. Similarly to the 2D case, the algorithm starts with a cubic 3D mesh and then produces a triangulated mesh Γ_h composed of triangular faces obtained by cutting the cubic elements that intersect the surface Γ, see [51]. In our work [31] we have shown that the cubes and cut cubes produced as a by-product of the the marching cubes algorithm naturally compose a polyhedral bulk mesh Ω_h that approximates Ω, such that $\Gamma_h = \partial\Omega_h$, see Fig. 4 for an illustration. This bulk-surface variant of the marching cubes algorithm was proposed in our work [31] and is now available in VEMcomp through the function `generate_mesh3d`, whose syntax is as follows:

```
[P, h, BulkElements, SurfElements] ...
    = generate_mesh3d(fun, Q, Nx, tol);
[P, h, BulkElements, SurfElements, ElementsPlot] ...
    = generate_mesh3d(fun, Q, Nx, tol, xcut);
```

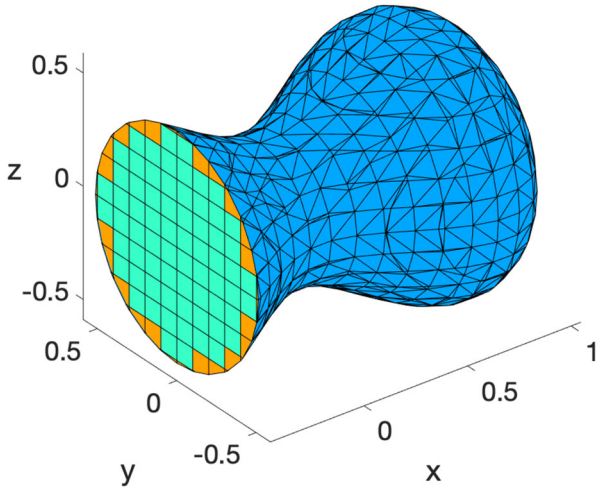

Fig. 4 A conforming bulk-surface mesh in 3D generated with the VEMcomp function `generate_mesh3d`. The bulk domain Ω is approximated by a polyhedral mesh Ω_h composed of cubic elements (green) and more general polyhedral elements (orange) close to the surface Γ. The surface Γ is approximated by a triangulated surface Γ_h (blue) taken as the boundary of the bulk mesh Ω_h

In the above, inputs and outputs are analogous to the 2D case. Specifically, in input:

- `fun` is the level function used in (8), represented as an inline function of the type `fun = @(P) <expression>`, where `P` is any $n \times 3$ array of $n \in \mathbb{N}$ points in $\mathbb{R}^3$;
- `Q` is the 2×3 array introduced in (8), which in Matlab syntax can be created using the syntax $Q = [x_{\min}, x_{\max}; y_{\min}, y_{\max}; z_{\min}, z_{\max}]$;
- `Nx` $>= 2$ is the required amount of gridpoints along each dimension. If Q is not a cube, then the shortest side of Q is discretised with `Nx` gridpoints, such that the resulting grid is equally spaced;
- `tol` > 0 is the minimum distance between distinct nodes. Any two nodes distant from each other less than `tol` will be merged into a unique node;
- If the optional input `xcut`$\in \mathbb{R}$ is provided, then VEMcomp considers the auxiliary mesh

$$\Omega_{\text{cut}} := \{(x, y, z) \in \Omega \mid x \leq x_{\text{cut}}\}, \qquad \Gamma_{\text{cut}} := \partial\Omega_{\text{cut}}, \tag{9}$$

  which will be useful at a later stage to plot numerical solutions inside of Ω, as explained below in the list of outputs.

In output:

- `P` is a $N \times 3$ array containing the $N \in \mathbb{N}$ nodes of the bulk mesh $\Omega_h \subset \mathbb{R}^3$. The nodes of the surface mesh Γ_h constitute a subset of `P`;
- `h` > 0 is the meshsize of the bulk mesh Ω_h;

- `BulkElements` is a list of the bulk elements in `element3d` format;
- `SurfElements` is a $M \times 3$ array, where $M \in \mathbb{N}$ is the number of surface elements and, for each $i = 1, \ldots, M$, the nodes of the i-th surface element (a triangle) are `P(SurfElements(i,:),:)`;
- The optional output `ElementsPlot` is a polygonal mesh, represented as a list of objects of the class `element2d`, that approximates the surface Γ_{cut} defined in (9). The surface mesh `ElementsPlot` can be used to plot numerical solutions inside of Ω.

Similarly to the 2D case, we remark that the marching cubes algorithm does not guarantee mesh regularity [51]. For a workaround, we redirect the reader to the discussion at the end of Section 3.3. Finally, it must be noted that `generate_mesh3d` is guaranteed to generate star-shaped elements with star-shaped faces, which is essential for matrix assembly, see Section 4.3.

Remark 1 (Compatibility with third-party mesh generators) For triangulated surface meshes in 3D, the format (`P`, `SurfElements`) returned by the function `generate_mesh3d` is commonly used in finite element coding. For instance, the well-known Matlab package DistMesh [41] generates triangulated surface meshes in the same format. This implies that VEMcomp can solve surface PDEs using triangulated surface meshes generated with DistMesh, as we will show in the numerical example in Section 6.2.2. For bulk and bulk-surface meshes, the meshes generated by DistMesh or other packages would require a preliminary conversion step before being usable in VEMcomp.

4 Computation of local and global matrices

In this Section we will address the computation of (i) local and (ii) global mass and stiffness matrices, specifically:

- We will show that the classes `element2d` and `element3d` introduced in Sections 3.3-3.4 will also allow to determine the local mass and stiffness matrices. To test the correctness of VEMcomp, we present for the first time the closed-form local VEM matrices for a square and a cubic element and compare the results with those obtained numerically by VEMcomp;
- We present the built-in functions `assembly2d` and `assembly3d`, which allow for the assembly of the global mass and stiffness matrices in 2D and 3D, respectively.

We remark that, following [30], the mass matrix is sufficient for the approximation of the nonlinearities in the time-dependent problems (2), (4) and (6).

4.1 A worked example in 2D: the unit square

Here, we show the usage of `element2d` to compute the local matrices of the unit square and will compare the results with the local matrices in closed-form. In this regard, we remark that, while the closed-form stiffness matrix for the unit square was presented in [54], the closed-form mass matrix is a novel result. This result is further motivated by the marching squares mesh generation algorithm mentioned in Section 3.3. Consider the unit square $F = [0, 1]^2$ contained in the xy-plane. Notice that vertex ordering affects the resulting matrices, so we order the vertexes as (0, 0, 0), (0, 1, 0), (1, 1, 0), and (1, 0, 0). With this choice, the closed-form local stiffness, mass and consistency matrices of F for the VEM of lowest order $k = 1$ are as follows:

$$K = \frac{1}{4}\begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}; \; M = \frac{1}{48}\begin{bmatrix} 17 & -9 & 13 & -9 \\ -9 & 17 & -9 & 13 \\ 13 & -9 & 17 & -9 \\ -9 & 13 & -9 & 17 \end{bmatrix}; \; C = \frac{1}{48}\begin{bmatrix} 5 & 3 & 1 & 3 \\ 3 & 5 & 3 & 1 \\ 1 & 3 & 5 & 3 \\ 3 & 1 & 3 & 5 \end{bmatrix}. \quad (10)$$

If we compute numerically such matrices using VEMcomp as shown in Appendix A, the elementwise errors are: 0 for the stiffness matrix K, 5.5511e-17 for the mass matrix M, and 2.7756e-17 for the consistency matrix C. The numerically computed matrices are thus exact up to machine precision.

4.2 A worked example in 3D: the unit cube

Here we show the usage of `element3d` to compute the local matrices of the unit cube $E = [0, 1]^3$ and will compare the results with the corresponding closed-form matrices. In this regard, we remark that the closed-form VEM local matrices for the unit cube, which we present here, are a novel result in the literature to the best of our knowledge. This result is further motivated by the marching cubes mesh generation algorithm mentioned in Section 3.4. Because vertex ordering is reflected in the resulting matrices, we order the vertexes as follows:

$$(0, 0, 0)\ (0, 0, 1)\ (0, 1, 0)\ (0, 1, 1)\ (1, 0, 0)\ (1, 0, 1)\ (1, 1, 0)\ (1, 1, 1). \quad (11)$$

The local stiffness, consistency and mass matrices for the lowest order $k = 1$ in closed-form are as follows. The interested reader is referred to Appendix A for the

derivation.

$$K=\frac{1}{16}\begin{bmatrix} 3 & 1 & 1 & -1 & 1 & -1 & -1 & -3 \\ 1 & 3 & -1 & 1 & -1 & 1 & -3 & -1 \\ 1 & -1 & 3 & 1 & -1 & -3 & 1 & -1 \\ -1 & 1 & 1 & 3 & -3 & -1 & -1 & 1 \\ 1 & -1 & -1 & -3 & 3 & 1 & 1 & -1 \\ -1 & 1 & -3 & -1 & 1 & 3 & -1 & 1 \\ -1 & -3 & 1 & -1 & 1 & -1 & 3 & 1 \\ -3 & -1 & -1 & 1 & -1 & 1 & 1 & 3 \end{bmatrix} + \frac{\sqrt{3}}{4}\begin{bmatrix} 2 & -1 & -1 & 0 & -1 & 0 & 0 & 1 \\ -1 & 2 & 0 & -1 & 0 & -1 & 1 & 0 \\ -1 & 0 & 2 & -1 & 0 & 1 & -1 & 0 \\ 0 & -1 & -1 & 2 & 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 2 & -1 & -1 & 0 \\ 0 & -1 & 1 & 0 & -1 & 2 & 0 & -1 \\ 0 & 1 & -1 & 0 & -1 & 0 & 2 & -1 \\ 1 & 0 & 0 & -1 & 0 & -1 & -1 & 2 \end{bmatrix}; \tag{12}$$

$$C = \frac{1}{96}\begin{bmatrix} 3 & 2 & 2 & 1 & 2 & 1 & 1 & 0 \\ 2 & 3 & 1 & 2 & 1 & 2 & 0 & 1 \\ 2 & 1 & 3 & 2 & 1 & 0 & 2 & 1 \\ 1 & 2 & 2 & 3 & 0 & 1 & 1 & 2 \\ 2 & 1 & 1 & 0 & 3 & 2 & 2 & 1 \\ 1 & 2 & 0 & 1 & 2 & 3 & 1 & 2 \\ 1 & 0 & 2 & 1 & 2 & 1 & 3 & 2 \\ 0 & 1 & 1 & 2 & 1 & 2 & 2 & 3 \end{bmatrix}; \tag{13}$$

$$M = \frac{1}{96}\begin{bmatrix} 51 & -22 & -22 & 1 & -22 & 1 & 1 & 24 \\ -22 & 51 & 1 & -22 & 1 & -22 & 24 & 1 \\ -22 & 1 & 51 & -22 & 1 & 24 & -22 & 1 \\ 1 & -22 & -22 & 51 & 24 & 1 & 1 & -22 \\ -22 & 1 & 1 & 24 & 51 & -22 & -22 & 1 \\ 1 & -22 & 24 & 1 & -22 & 51 & 1 & -22 \\ 1 & 24 & -22 & 1 & -22 & 1 & 51 & -22 \\ 24 & 1 & 1 & -22 & 1 & -22 & -22 & 51 \end{bmatrix}. \tag{14}$$

If we compute numerically such matrices using VEMcomp as shown in Appendix A, the elementwise errors are: 2.2204e-16 for the stiffness matrix K, and 1.5543e-15 for both the mass matrix M and the consistency matrix C. The numerically computed matrices are thus exact up to machine precision.

4.3 Global matrix assembly

Once a mesh has been generated in the format returned by the functions `generate_mesh2d` or `generate_mesh3d` in Section 3, VEMcomp provides two functions for global matrix assembly in 2D and 3D. The 2D case is covered by the `assembly2d` function, whose syntax is as follows:

```
[K,C,M,KS,MS,R]=assembly2d(P,BulkElements,SurfElements);
```

In the above, the inputs are as returned by the function `generate_mesh2d` in Section 3.3, while the outputs are defined as follows:

- `K`, `C` and `M` are the stiffness, consistency and mass matrices in the bulk, stored as sparse matrices;

- `KS` and `MS` are the stiffness and mass matrices on the surface, stored as sparse matrices. We observe that the surface here is a curve, so the "1D surface VEM" coincides with the 1D surface FEM, see [29]. This implies that the consistency matrix on the surface, say `CS`, coincides with the mass matrix on the surface `MS` [29], hence the lack of the additional output `CS`;
- `R` is the reduction matrix, see [29].

For full definitions of the above matrices, see [29]. Analogously, for matrix assembly in 3D, VEMcomp provides the function `assembly3d`, whose syntax is

```
[K,C,M,KS,CS,MS,R]=assembly3d(P,BulkElements,SurfElements);
```

The inputs `P`, `ElementsBulk`, `ElementsSurface` are as returned by the function `generate_mesh3d` in Section 3.4. The outputs `K,C,M,KS,CS,MS,R` are defined as in the 2D case, with the addition of the consistency matrix `CS` for the surface, since in the 3D case it is no longer true that `CS = MS`, see [30]. It is worth noting that the `PDEtool` in Matlab has a similar syntax in the built-in function `assempde` for the FEM, see [55].

5 Black-box solvers and post-processing

In this Section we will present VEMcomp's functions for (i) solving the PDE models (1)-(6), (ii) plotting the numerical solutions and (iii) computing the numerical errors when a closed-form solution is known. The usage of all these functions will be demonstrated in several numerical examples in Section 6.

5.1 Black-box solvers

VEMcomp provides black-box functions for the numerical approximation, via (BS)-FEM or (BS)-VEM, of the (BS)-PDE problems considered in Section 2. Here, we will state the syntax of such black-box solvers. Full examples will be shown in Section 6. VEMcomp solves the elliptic bulk-only problem (1) through the function

```
u = solver_elliptic_bulk(D, alpha, f, P, M, K, R, bcond);
```

where, in input:

- `D`, `alpha` are two real numbers (`double`) containing the diffusion coefficient $d^\Omega > 0$ and the coefficient $\alpha \geq 0$ as in (1);
- `f` is an inline function representing the load term f in (1);
- `P`, `M`, `K`, `R` are as in the function `assembly2d` or `assembly3d` introduced in Section 4.3;
- `bcond` is a string indicating the kind of boundary conditions. The available options for `bcond` are `'dir'` and `'neu'`, which correspond to homogeneous Dirichlet or Neumann boundary conditions, respectively.

In output, `u` is a `size(P,1)×1` array containing the nodal values of the numerical solution. The semilinear parabolic system (2) is solved through the function

```
[u,t,uprime_norm,u_average] = solver_parabolic_bulk(D, f, ...
    P, M, K, R, bcond,T,tau,u0);
```

where, in input:

- `D` is a `n×1` array containing the diffusion coefficients d_i^{Ω}, $i = 1, \dots, n$;
- `f` is a `n×1` cell array containing the kinetics f_i as inline functions;
- `P, M, K, R, bcond` are as in the previous function `solver_elliptic_bulk`;
- `T` is the final time and `tau` is the timestep used in the IMEX Euler timestepping scheme [29];
- `u0` is a `size(P,1)×n` array where the i-th column contains the nodal values of the i-th component of the initial condition, for all $i = 1, \dots, n$;

In output:

- `u` is a `size(P,1)×1` array, where the i-th column contains the nodal values of the component u_i of the numerical solution evaluated at the final time T, for all $i = 1, \dots, n$.
- the optional output `t` is a $1 \times N_T$ array, where $N_T := \lceil \frac{T}{\tau} \rceil$, containing the time steps $t_k := k\tau$, $k = 1, \dots, N_T$.
- the optional output `uprime_norm` is also a $1 \times N_T$ array containing the L^2 norm of the discrete time derivative of the first component of `u` evaluated at all time nodes:

$$\texttt{uprime\_norm}(k) = \frac{\|u_1^{(k)} - u_1^{(k-1)}\|_{L^2(\Omega_h)}}{\tau} := \frac{1}{\tau}\left(\int_{\Omega_h}\left(u_1^{(k)} - u_1^{(k-1)}\right)^2\right)^{\frac{1}{2}},\ k = 1, \dots, N_T.$$

- the optional output `u_average` is a $1 \times N_T$ array containing the spatial average of the first component of `u` evaluated at all time nodes:

$$\texttt{u\_average}(k) = < u_1^{(k)} > := \frac{1}{\text{measure}(\Omega_h)}\int_{\Omega_h} u_1^{(k)},\ k = 1, \dots, N_T.$$

The optional outputs above are usually used in the context of RDSs to assess whether a stationary steady state (Turing pattern) has been reached, see for instance [56, 57]. With a similar and self-explanatory syntax, the surface and bulk-surface model problems (3)-(6) are solved through the functions

```
v = solver_elliptic_surf(D, alpha, g, P, MS, KS, R);
[v,t,vprime_norm,v_average] = solver_parabolic_surf(D, f, ...
    P, MS, KS, R, T,tau, v0);
[u,v] = solver_elliptic_bulk_surf(dOmega, dGamma, alpha, ...
    beta, f, g, P, M, MS, K, KS, R, gamma,delta);
[u,v,t,uprime_norm,vprime_norm,u_average,v_average] = ...
    solver_parabolic_bulk_surf(dOmega, dGamma, f, g, ...
    h, P, M, MS, K, KS, R,T,tau,u0,v0);
```

All the mentioned black-box solvers rely on Matlab's backslash \ direct solver, accelerated through the `symamd` reordering, for the linear systems appearing in the fully discrete problems.

5.2 Solution plotters

We now present VEMcomp's inhouse functions for plotting numerical solutions. In the 2D case, when Ω is a flat domain and $\Gamma = \partial\Omega$ is a curve, each bulk- and surface component of a numerical solution is plotted via the functions

```
plot_bulk2d(BulkElements, u_i, titlestring);
plot_surf2d(P,SurfaceElements,v_j,titlestring);
```

respectively. In the above, in input:

- `P`, `BulkElements` and `SurfaceElements` are outputs of the function `generate_mesh2d` in Section 3.3;
- `u_i` is a `size(P,1)` $\times$ 1 vector containing the nodal values of the bulk component u_i of the numerical solution, and corresponds to the i-th column of the array `u` returned by any of the black-box solvers in Section 5.1;
- `v_j` is a `size(R,2)` $\times$ 1 vector containing the nodal values of the surface component v_j of the numerical solution, and corresponds to the j-th column of the array `v` returned by any of the black-box solvers in Section 5.1;
- the optional input `titlestring` is the title of the plot in `string` format.

Similarly, for the 3D case, the plots are generated by the following functions:

```
plot_bulk3d(ElementsPlot, u_i, titlestring);
plot_surf3d(P,R,SurfaceElements,v_j,titlestring);
```

where, in input,

- `P`, `ElementsPlot` and `SurfaceElements` are generated by the function `generate_mesh3d` in Section 3.4;
- `R` is the reduction matrix generated by the function `assembly3d` in Section 4.3;
- `u_i`, `v_j` and `titlestring` are defined as in the 2D case above.

The presence of inhouse functions for solution plotting avoids the necessity to export the numerical solution to external plotters such as ParaView [58].

5.3 Error evaluation

Being based on the Virtual Element Method, VEMcomp is backed up by extensive literature on stability and convergence. The interested reader is referred to the literature review in Section 2. When a closed-form solution is known, VEMcomp allows to compute the relative error of the numerical solution in (i) $L^2(\Omega)$ norm for bulk-only problems, (ii) $L^2(\Gamma)$ norm for surface-only problems and (iii) $L^2(\Omega) \times L^2(\Gamma)$ norm for bulk-surface problems. It is worth recalling that the VEM basis functions are not known in closed form. For this reason it is not possible to evaluate the error of the

numerical solution exactly. The VEM literature typically faces this issue by evaluating the error of the *piecewise polynomial projection of the numerical solution*, see for instance [15, Section 3.1.2], and VEMcomp follows this approach. Such error evaluation is performed by the function `compute_error` whose syntax is as follows

```
L2_relative_err = compute_error(C,[],u,u_exact,[],[]);
L2_relative_err = compute_error([],MS,[],[],v,v_exact);
L2_relative_err = compute_error(C,MS,u,u_exact,v,v_exact);
```

for bulk, surface and bulk-surface problems, respectively. In input:

- the bulk consistency matrix `C` and the surface mass matrix `MS` are outputs of `assembly2d` or `assembly3d` depending on the space dimension of the problem;
- `u` and/or `v` are outputs of any of the black-box solvers in Section 5.1;
- `u_exact` and/or `v_exact` are arrays of the same sizes of `u` and `v`, respectively, containing the nodal values of the exact solution.

In the next section, we present six numerical examples that illustrate the applicability, versatility and generality of VEMcomp.

6 Numerical examples

In this Section, we present six numerical experiments carried out in VEMcomp to showcase the usage of our Matlab package.

6.1 Bulk-only problems

This first set of examples showcases the application of VEMcomp to bulk-only PDE problems. In Example 6.1.1 we consider a Poisson problem in 2D on a circular domain with Neumann boundary conditions. In Example 6.1.2 we show the solution of a Poisson problem in 3D on the spherical domain with Dirichlet boundary conditions. The main aim here is demonstrate the generality of VEMcomp in dealing with different PDEs and different types of boundary conditions.

6.1.1 Poisson problem in 2D on a circular domain

We consider the following Poisson problem on the unit circle $\Omega = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$ with homogeneous Neumann boundary conditions:

$$\begin{cases} -\Delta u + u = 8(1 - 2(x^2 + y^2)) + (1 - (x^2 + y^2))^2, & (x, y) \in \Omega; \\ \nabla u \cdot \boldsymbol{n} = 0, \quad (x, y) \in \partial\Omega, \end{cases} \tag{15}$$

whose exact solution is given by $u(x, y) = (1-(x^2+y^2))^2$ for all $(x, y) \in \Omega$. We solve the considered problem, plot the numerical solution and evaluate the relative error in $L^2(\Omega)$ norm using the following VEMcomp code:

```
% STEP 1: Generate mesh
level_fun = @(P) P(:,1).^2 + P(:,2).^2 - 1;
range = [-1,1;-1,1]; Nx = 40; tol = 1e-6;
[P,h,BulkElements,SurfElements] = ...
        generate_mesh2d(level_fun,range,Nx,tol);

% STEP 2: Matrix assembly
[K,C,M,KS,MS,R] = assembly2d(P,BulkElements,SurfElements);

% STEP 3: Solve PDE
D = 1; alpha = 1; bcond = 'neu';
f = @(P) 8*(1-2*(P(:,1).^2+P(:,2).^2)) + ...
        (1-(P(:,1).^2+P(:,2).^2)).^2;
u = solver_elliptic_bulk(D, alpha, f, P, M, K, R, bcond);

% STEP 4: Post-processing
figure, set(gcf,'color','white')
plot_bulk2d(BulkElements, u, '$u$')
u_exact = (1- (P(:,1).^2 + P(:,2).^2)).^2;
L2_relative_err = compute_error(C,[],u,u_exact,[],[]);
```

The above code generates a mesh with $N = \texttt{size(P,1)} = 1336$ nodes and meshsize $h = 0.0725$. The obtained relative error in $L^2(\Omega)$ norm is $2.3734e\text{-}2$. The numerical solution and the mesh are shown in Fig. 5.

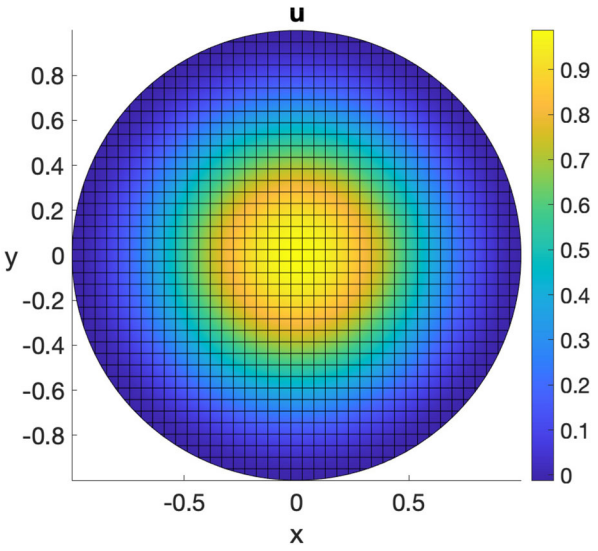

Fig. 5 Numerical solution of the elliptic bulk problem (15) obtained in VEMcomp on a mesh with $N = 1336$ nodes and meshsize $h = 0.0725$

6.1.2 Poisson equation in 3D on a spherical domain

We now consider on the unit sphere $\Omega := \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 \leq 1\}$ the following Poisson problem with homogeneous Dirichlet boundary conditions:

$$\begin{cases} -\Delta u + u = 7 - (x^2 + y^2 + z^2) & (x, y, z) \in \Omega; \\ u = 0 & (x, y, z) \in \partial\Omega, \end{cases} \tag{16}$$

whose exact solution is given by $u(x, y, z) = 1 - (x^2 + y^2 + z^2)$ for all $(x, y, z) \in \Omega$. We solve problem (16) in VEMcomp by running the following script code

```
% STEP 1: Generate mesh
level_fun = @(P) P(:,1).^2 + P(:,2).^2 + P(:,3).^2 -1;
range = [-1,1; -1,1; -1,1];
Nx = 30; tol = 1e-6; xcut = -0.3;
[P,h,BulkElements,SurfElements,ElementsPlot] = ...
    generate_mesh3d(level_fun,range,Nx,tol,xcut);

% STEP 2: Matrix assembly
[K,M,C,KS,MS,CS,R] = assembly3d(P,BulkElements,SurfElements);

% STEP 3: Solve PDE
D = 1;   alpha = 1; bcond = 'dir';
f = @(P) 7 - (P(:,1).^2 + P(:,2).^2 + P(:,3).^2);
u = solver_elliptic_bulk(D, alpha, f, P, M, K, R, bcond);

% STEP 4: Post-processing
figure, set(gcf,'color','white')
plot_bulk3d(ElementsPlot, u, '$u$')
colormap cool % Default colormap is parula
u_exact = 1- (P(:,1).^2 + P(:,2).^2 + P(:,3).^2);
L2_relative_err = compute_error(C,[],u,u_exact,[],[]);
```

The above code generates a mesh and computes its meshsize `h`, which is equal to 0.1195. By exploiting the definitions of the matrices produced by `assembly3d`, the number of nodes is given e.g. by `size(P,1)` and is equal to 16600. The relative error in $L^2(\Omega)$ norm is 1.7372e-4. The numerical solution and the mesh are shown in Fig. 6.

6.2 Surface-only problems

This second set of examples showcases the application of VEMcomp to surface-only problems. In Example 6.2.1 we consider a linear parabolic problem on a spherical surface. In Example 6.2.2 we solve a reaction-diffusion system on an ellipsoidal surface

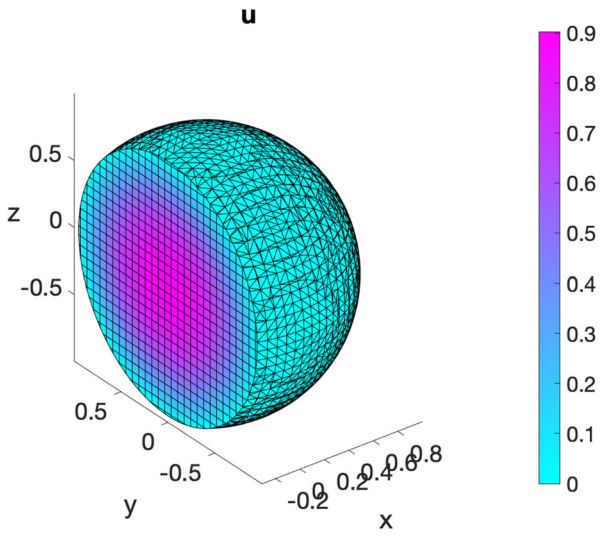

Fig. 6 Numerical solution of the elliptic bulk problem (16) obtained in VEMcomp on a mesh with $N = 16600$ nodes and meshsize $h = 0.1195$. The colormap is different than in the other experiments, in order to better highlight the structure of the mesh

and we demonstrate the compatibility of VEMcomp with the triangulated surfaces generated by DistMesh.

6.2.1 Linear parabolic problem

In this Section we show VEMcomp's ability to solve surface PDEs. For illustrative purposes, we choose a linear parabolic problem on a spherical surface in 3D. We remark, however, that VEMcomp can solve surface PDE problems in 2D as well, i.e. when the spatial domain Γ is a curve in $\mathbb{R}^2$. Let us then consider the following linear parabolic problem on the unit spherical surface Γ:

$$\begin{cases} \dfrac{\partial u}{\partial t} - \Delta_\Gamma u = 13xyze^t, & (x, y, z, t) \in \Gamma \times [0, T]; \\ u(x, y, z, 0) = xyz, & (x, y, z) \in \Gamma, \end{cases} \tag{17}$$

whose exact solution is given by $u(x, y, z, t) = xyze^t$ for all $(x, y, z, t) \in \Gamma \times [0, T]$. The surface Γ is approximated with the same surface mesh `SurfElements` generated -but not used- in the previous example in Section 6.1.2 with `Nx` = 30, while the bulk mesh `BulkElements` is now unused. By using the definition of the matrices produced by `assembly3d`, the number of nodes of the surface mesh is given e.g. by `length(KS)` or `size(R,2)` and is equal to 3888. Furthermore, as discussed in Section 6.1.2, the meshsize `h` is 0.1195. The final time is $T = 1$ and the timestep is $1e$-4. The code for mesh generation and matrix assembly is the same as in the previous experiment in Section 6.1.2. Therefore we report here only the code for the solver and plotter:

```
% STEP 3: Solve PDE
g = {@(u,P,t) 13*P(:,1).*P(:,2).*P(:,3)*exp(t)};
v0 = R'*(P(:,1).*P(:,2).*P(:,3));
D = 1; T = 1; tau = 1e-4;
v = solver_parabolic_surf(D, g, P, MS, KS, R, T, tau, v0);

% STEP 4: Post-processing
figure, set(gcf,'Color','white')
plot_surf3d(P,R,SurfElements,v,'$v$')
xlim([-0.5,1]) % to cut the surface
v_exact = R'*(P(:,1).*P(:,2).*P(:,3)*exp(T));
L2_relative_err = compute_error([],MS,[],[],v,v_exact);
```

The relative error, measured at the final time, is 6.5565e-3. The numerical solution and the surface mesh are shown in Fig. 7.

6.2.2 Surface RDS on the ellipsoid using DistMesh

To showcase the compatibility between DistMesh and VEMcomp in the case of surface PDEs, we solve the following surface reaction-diffusion system (RDS) with activator-depleted kinetics:

$$\begin{cases} \dfrac{\partial v_1}{\partial t} - \Delta_\Gamma v_1 = \gamma_\Gamma g_1(v_1, v_2), & \text{in } \Gamma \times [0, T]; \\ \dfrac{\partial v_2}{\partial t} - d_2^\Gamma \Delta_\Gamma v_2 = \gamma_\Gamma g_2(v_1, v_2), & \text{in } \Gamma \times [0, T], \end{cases} \tag{18}$$

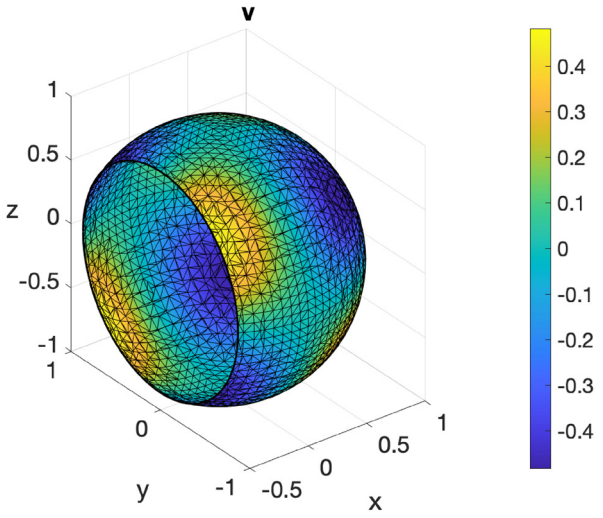

Fig. 7 Numerical solution of the parabolic surface PDE (17) obtained in VEMcomp on a mesh with $N = 3888$ nodes and meshsize $h = 0.1195$

where $g_1(v_1, v_2) = a - v_1 + v_1^2 v_2$, $g_2(v_1, v_2) = b - v_1^2 v_2$, and a, b are positive reaction parameters. The model was considered in [59] and is an instance of the general surface parabolic problem (4). As shown in [59], the following is a spatially uniform steady state:

$$(v_1^*, v_2^*) := \left(a + b, \frac{b}{(a+b)^2}\right). \tag{19}$$

Suitable conditions on a, b, d_2^Γ ensure that the steady state (19) is Turing unstable, i.e. in the presence of diffusion it destabilizes and for long time integration the solution tends towards a Turing pattern inhomogeneous in space. Following [59] we choose the following parameters:

$$a = 0.1;\ b = 0.9;\ d_2^\Gamma = 10. \tag{20}$$

Finally, for illustrative purposes, we choose $\gamma_\Gamma = 300$. For the spatial domain Γ, we consider the ellipsoid

$$\Gamma := \left\{(x, y, z) \in \mathbb{R}^3 \;\middle|\; \frac{x^2}{4} + y^2 + \frac{z^2}{1.5^2} - 1 = 0\right\}, \tag{21}$$

considered in the DistMesh manual [41]. To show pattern formation, the initial data for (18) are chosen as small spatially random perturbations of amplitude 1e-3 around the equilibrium (19). The final time is $T = 10$ and the timestep is $\tau = 1e\text{-}5$. As expected, the solution at the final time, shown in Fig. 8, exhibits spatial patterns. Such solution is an asymptotic steady state since, as we can see in Fig. 8, at the final time, the L^2 norm of the discrete time derivative $\dot{v}_1$ is neglibible and the spatial average of v_1 has reached an asymptotic value. The code for generating the surface mesh with DistMesh, for solving the model and plotting the solution and the post-processing indicators in VEMcomp is as follows:

```
% STEP 1: Generate mesh using DistMesh
h = 0.1; % Upper bound of the meshsize
level_fun = @(P) P(:,1).^2/4+P(:,2).^2/1+P(:,3).^2/1.5^2−1;
[P,SurfElements] = distmeshsurface(level_fun,@huniform,h,...
        [−2.1,−1.1,−1.6; 2.1,1.1,1.6]);

% STEP 2: Matrix assembly
[∼, ∼, ∼, KS, MS,∼,R] = assembly3d(P, [], SurfElements);
% There are no bulk elements and no bulk matrices

% STEP 3: Set model and solve PDE
a = 0.1; b = 0.9; dGamma = [1;10]; gammaGamma = 300;
g = [{@(u,P,t) gammaGamma∗(a−u(:,1)+u(:,1).^2.∗u(:,2))}; ...
     {@(u,P,t) gammaGamma∗(b−u(:,1).^2.∗u(:,2))}];
rng(0); % initialize random number generator
```

```
v0 = [a + b + 1e−3∗(2∗rand(size(P,1),1)−1), ...
      b/(a+b)^2 + 1e−3∗(2∗rand(size(P,1),1)−1)];
T = 10; tau = 1e−5;
v = solver_parabolic_surf(dGamma,g,P,MS,KS,R,T,tau,v0);

% STEP 4: Post−processing
figure, set(gcf, 'color', 'white')
sgtitle('Reaction−diffusion system on ellipsoidal surface')
subplot(2,2,1) % plot component v1
plot_surf3d(P,R,SurfElements,v(:,1),'$v_1$')
subplot(2,2,2) % plot component v2
plot_surf3d(P,R,SurfElements,v(:,2),'$v_2$')
subplot(2,2,3) % plot L2 norm of time derivative of v1
semilogy(t, vprime_norm, 'LineWidth',2)
legend('$\\dot{v}_1\_{L^2(\Gamma_h)}$','interpreter','latex'), xlabel('t'),
      set(gca, 'FontSize', 14)
subplot(2,2,4) % plot spatial average of v1
plot(t, v_average, 'LineWidth',2)
legend('$<v_1>$','interpreter','latex')
xlabel('t'), set(gca,'FontSize', 14)
```

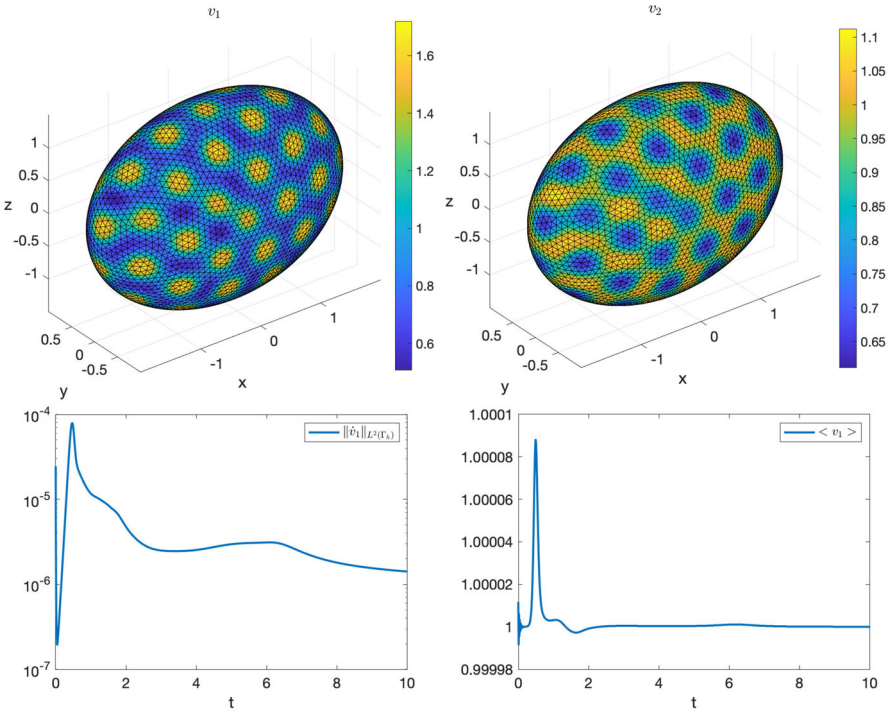

Fig. 8 Numerical solution at the final time $T = 10$ of the S-RDS (18) on the ellipsoid Γ defined in (21), solved in VEMcomp on a mesh with 3990 nodes and meshsize $h \leq 0.1$, whose exact value is not provided by DistMesh. Top row: component v_1 (left) and component v_2 (right). Bottom row: L^2 norm of time derivative of v_1 over time (left) and spatial average of v_1 over time (right)

In the above code, the meshsize, not provided exactly by DistMesh, is upper-bounded by `h` = 0.1. The number of nodes, given by `size(P,1)`, is 3990.

6.3 Numerical examples for bulk-surface problems in 3D

We now apply VEMcomp to two different BS-PDEs in 3D. In Example 6.3.1 we show a baseline problem given by a bulk-surface linear elliptic problem on the sphere. In Example 6.3.2, we use VEMcomp to solve the top-end PDE problem considered in this work: a bulk-surface reaction-diffusion system (BS-RDS) on the sphere. We remark that VEMcomp can solve bulk-surface problems in 2D as well.

6.3.1 Elliptic bulk-surface problem in 3D on the sphere

We numerically solve the following elliptic bulk-surface problem, from [60], on the unit sphere Ω in 3D:

$$\begin{cases} -\Delta u + u = xyz, & \text{in } \Omega; \\ -\Delta_\Gamma v + v + \nabla u \cdot \boldsymbol{n} = 29xyz, & \text{on } \partial\Omega; \\ \nabla u \cdot \boldsymbol{n} = -u + 2v, & \text{on } \partial\Omega, \end{cases} \tag{22}$$

whose exact solution is given by

$$\begin{aligned} u(x, y, z) &= xyz, \qquad (x, y, z) \in \Omega; \\ v(x, y, z) &= 2xyz, \qquad (x, y, z) \in \partial\Omega. \end{aligned}$$

We use the same mesh considered in Experiment 6.1.2. The obtained relative error in $L^2(\Omega) \times L^2(\Gamma)$ norm is 6.9535e-3. The components of the numerical solution and

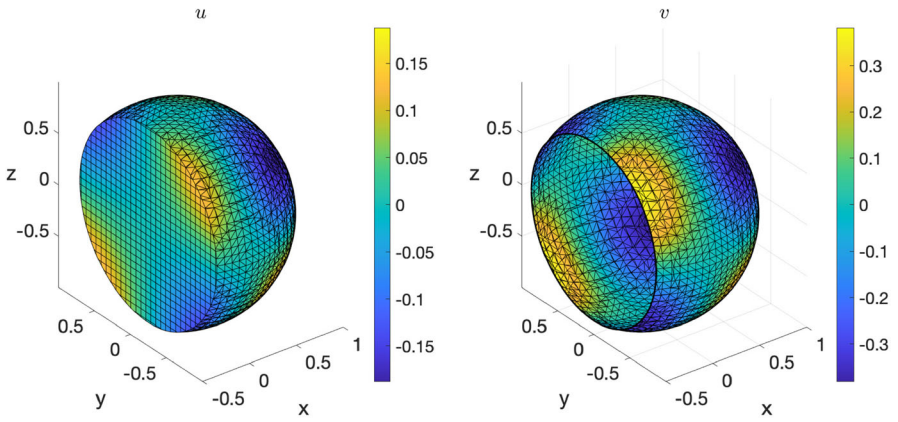

Fig. 9 Elliptic bulk-surface problem (22) on the unit sphere Ω in 3D, solved in VEMcomp on a mesh with 16600 nodes and meshsize 0.1195. Left: bulk component u. Right: surface component v

the bulk-surface mesh are plotted in Fig. 9. Omitting the code for mesh generation and matrix assembly, which is the same as in the experiment in Section 6.1.2, the relevant piece of code for the dedicated solver and post-processing is:

```
% STEP 3: Solve PDE
dOmega = 1; dGamma = 1;
alpha = 1; beta = 1; gamma = -1; delta = 2;
% right-hand-side of 1st eq. in (35):
f = @(P) P(:,1).*P(:,2).*P(:,3);
% right-hand-side of 2nd eq. in (35):
g = @(P) 29*P(:,1).*P(:,2).*P(:,3);
[u,v] = solver_elliptic_bulk_surf(dOmega, dGamma, alpha,...
        beta, f, g, P, M, MS, K, KS, R, gamma,delta);

% STEP 4: Post-processing
figure, set(gcf, 'Color','white')
subplot(121) % plot bulk component u
plot_bulk3d(ElementsPlot, u, '$u$')
subplot(122) % plot surface component v
plot_surf3d(P,R,SurfElements,v,'$v$')
xlim([-0.5,1]) % to cut the surface
u_exact = P(:,1).*P(:,2).*P(:,3);
v_exact = 2*R'*(P(:,1).*P(:,2).*P(:,3));
L2_err_rel = compute_error(C,MS,u,u_exact,v,v_exact);
```

6.3.2 Bulk-surface reaction-diffusion system on the torus

In this final example we solve the following non-dimensionalised BS-RDS introduced in [61]:

$$\begin{cases} \dfrac{\partial u_1}{\partial t} - \Delta u_1 = \gamma_\Omega f_1(u_1,u_2), & \text{in } \Omega\times[0,T]; \\ \dfrac{\partial u_2}{\partial t} - d_2^\Omega \Delta u_2 = \gamma_\Omega f_2(u_1,u_2), & \text{in } \Omega\times[0,T]; \\ \dfrac{\partial v_1}{\partial t} - \Delta_\Gamma v_1 = \gamma_\Gamma (f_1(v_1,v_2) - h_1(u_1,u_2,v_1)), & \text{in } \Gamma\times[0,T]; \\ \dfrac{\partial v_2}{\partial t} - d_2^\Gamma \Delta_\Gamma v_2 = \gamma_\Gamma (f_2(v_1,v_2) - h_2(u_1,u_2,v_2)), & \text{in } \Gamma\times[0,T]; \\ \nabla u_1\cdot \boldsymbol{n} = \gamma_\Gamma h_1(u_1,u_2,v_1), & \text{in } \Gamma\times[0,T]; \\ d_\Omega \nabla u_2\cdot \boldsymbol{n} = \gamma_\Gamma h_2(u_1,u_2,v_2), & \text{in } \Gamma\times[0,T], \end{cases} \tag{23}$$

where $f_1(u_1,u_2) = a - u_1 + u_1^2 u_2$, $g(u_1,u_2) = b - u_1^2 u_2$, $h_1(u_1,u_2,v_1) = \alpha_1 v_1 - \beta_1 u_1 - \kappa_1 u_1$, $h_2(u_1,u_2,v_2) = \alpha_2 v_2 - \beta_2 u_1 - \kappa_2 u_2$, and $a, b, \alpha_1, \alpha_2, \beta_1, \beta_2, \kappa_1, \kappa_2$ are positive reaction parameters. As shown in [61], the following is a spatially uniform

steady state, under suitable conditions on the parameters $\alpha_1, \alpha_2, \beta_1, \beta_2, \kappa_1, \kappa_2$:

$$(u_1^*, u_2^*, v_1^*, v_2^*) := \left(a + b, \frac{b}{(a+b)^2}, a + b, \frac{b}{(a+b)^2}\right). \tag{24}$$

Further conditions on all the parameters except γ_Ω and γ_Γ ensure that the steady state (24) is Turing unstable, i.e. in the presence of diffusion it destabilizes and for long time integration the solution tends towards a Turing pattern inhomogeneous in space. Following [61] we choose the following parameters:

$$\begin{aligned} &a = 0.1;\ b = 0.9;\ \alpha_1 = 5/12;\ \alpha_2 = 5; \beta_1 = 5/12;\ \beta_2 = 0;\ \kappa_1 = 0;\ \kappa_2 = 5; \\ &d_2^\Omega = 10;\ d_2^\Gamma = 10. \end{aligned} \tag{25}$$

Finally, for illustrative purposes, we choose $\gamma_\Omega = \gamma_\Gamma = 300$. In [30], we had solved the BS-RDS (23) via BS-VEM on the unit sphere. Here, to showcase the generality of the VEMcomp library, we choose the torus

$$\Omega := \left\{ (x, y, z) \in \mathbb{R}^3 \;\middle|\; \left(\sqrt{x^2 + y^2} - \frac{7}{10}\right)^2 + z^2 - \frac{9}{100} \leq 0 \right\}, \tag{26}$$

as the bulk domain, and the toroidal surface $\Gamma := \partial\Omega$ as the surface domain. This specific toroidal surface Γ was used in our previous work [13] as a benchmark domain for the Surface Virtual Element Method (SVEM) for surface PDEs. It can be shown that Ω is bounded as follows:

$$\Omega \subset [-1, 1] \times [-1, 1] \times [-0.31, 0.31], \tag{27}$$

which is useful for mesh generation as discussed in Section 3.4. To show pattern formation, the initial data in (23) are chosen as small spatially random perturbations of amplitude 1e-3 around the equilibrium (24). Here we solve the model (23) by VEMcomp on a mesh generated by the `generate_mesh3d` function as shown below. The final time is $T = 5$ and the timestep is $\tau = 1e\text{-}5$. As expected, the solution at the final time, shown in Fig. 10, is an asymptotic steady state that exhibits spatial patterns. We solve the considered problem by VEMcomp using the following code:

```
% STEP 1: Assign domain and generate mesh
level_fun = @(P) (sqrt(P(:,1).^2 + P(:,2).^2) − 7/10).^2 ...
        + P(:,3).^2 − 9/100;
range = [−1,1;−1,1;−0.31,0.31]; %bounds of Omega as in (40)
Nx = 11; tol = 1e−10; xcut = −0.5;
[P,h,BulkElements,SurfElements, ElementsPlot] = ...
        generate_mesh3d(level_fun,range,Nx,tol,xcut);

% STEP 2: Matrix assembly
[K,C,M,KS,CS,MS,R]=assembly3d(P,BulkElements,SurfElements);

```

```
% STEP 3: Set model and solve PDE
a=0.1; b=0.9; alpha1=5/12; alpha2=5; beta1=5/12; beta2=0;
kappa1 = 0; kappa2 = 5; dOmega = [1;10]; dGamma = [1;10];
gammaOmega = 300; gammaGamma = 300;
f1 = @(u) a − u(:,1) + u(:,1).^2.∗u(:,2);
f2 = @(u) b − u(:,1).^2.∗u(:,2);
h1 = @(u,v) alpha1∗v(:,1) − beta1∗u(:,1) − kappa1∗u(:,2);
h2 = @(u,v) alpha2∗v(:,1) − beta2∗u(:,1) − kappa2∗u(:,2);
f = [{@(u,P,t) gammaOmega∗f1(u)}; ...
     {@(u,P,t) gammaOmega∗f2(u)}];
g = [{@(u,v,P,t) gammaGamma∗(f1(u) − h1(u,v))}; ...
     {@(u,v,P,t) gammaGamma∗(f2(u) − h2(u,v))}];
h = [{@(u,v,P,t) h1(u,v)}; {@(u,v,P,t) h2(u,v)}];
rng(0); % initialize random number generator
u0 = [a + b + 1e−3∗(2∗rand(size(P,1),1)−1), ...
      b/(a+b)^2 + 1e−3∗(2∗rand(size(P,1),1)−1)];
v0 = [a + b + 1e−3∗(2∗rand(size(R,2),1)−1), ...
      b/(a+b)^2 + 1e−3∗(2∗rand(size(R,2),1)−1)];
T = 5; tau = 1e−5;
[u,v] = solver_parabolic_bulk_surf(dOmega, dGamma, f, g, ...
        h, P, M, MS, K, KS, R, T, tau, u0, v0);

% STEP 4: Plot all components of numerical solution
figure, set(gcf, 'color', 'white')
subplot(2,2,1) % plot bulk Component u
plot_bulk3d(ElementsPlot, u(:,1), '$u_1$');
subplot(2,2,2) % plot bulk Component v
plot_bulk3d(ElementsPlot, u(:,2), '$u_2$')
subplot(2,2,3) % plot surface component r
plot_surf3d(P,R,SurfElements,v(:,1),'$v_1$')
xlim([xcut, 1]) % to cut the surface
subplot(2,2,4) % plot surface Component s
plot_surf3d(P,R,SurfElements,v(:,2),'$v_2$')
xlim([xcut, 1]) % to cut the surface
```

Similarly with the previous experiments, in the above code, the meshsize `h` is 0.1074, the overall number of nodes, given by `size(P,1)`, is 8144, while the number of boundary nodes, given by `length(KS)` or `size(R,2)`, is 3008.

7 Conclusions

We have introduced VEMcomp, a user-friendly MATLAB library for (i) polytopal bulk-surface mesh generation in 2D and 3D, (ii) matrix assembly for the lowest-order FEM and VEM, (iii) solving bulk, surface, and bulk-surface, elliptic and parabolic,

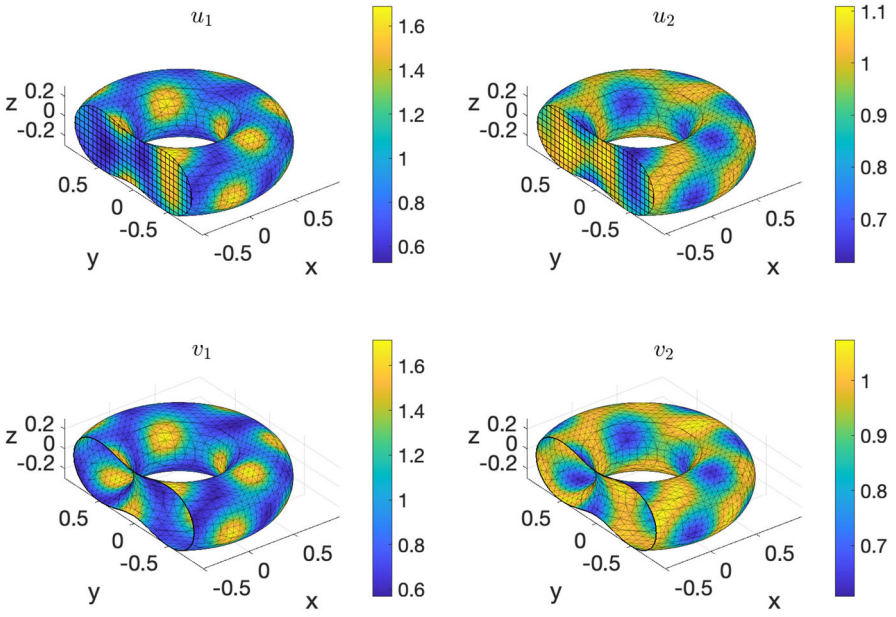

Fig. 10 Numerical solution at the final time $T = 5$ of the BS-RDS (23) on the torus Ω defined in (26), solved in VEMcomp on a mesh with 8144 nodes and meshsize 0.1074. Top row: bulk components (u_1, u_2). Bottom row: surface components (v_1, v_2)

linear and semilinear PDEs or systems of PDEs and (iv) post-processing the numerical solution, in terms of plotting and error evaluation. The present VEMcomp package is intended as a proof-of-concept tool, with the main goal of being user-friendly and self-explicative to solve a large selection of PDE models 2D and 3D. For this reason, VEMcomp has room for improvement in terms of computational performances. To this end, some main challenges are (i) devising mesh generation strategies that are cheaper and guarantee mesh regularity, (ii) devising cheaper quadrature formulas in 2D and 3D, (iii) including higher order VEM, (iv) solving PDEs on evolving domains and surfaces, (v) devising higher order schemes for time integration and finally (vi) adopting fast iterative solvers (e.g. conjugate gradient type) for the linear systems involved in the fully discrete problems.

Appendix A: Local matrices in closed form

As shown in [54, Sections 3.2 and 5.1], the computation of the local VEM matrices relies on three fundamental matrices $B \in \mathbb{R}^{(d+1)\times \mathtt{NVert}}$, $D \in \mathbb{R}^{\mathtt{NVert}\times(d+1)}$, $H \in \mathbb{R}^{(d+1)\times(d+1)}$ (where $d \in \{2, 3\}$ is the dimension of the problem and $\mathtt{NVert}$ is the number of vertexes of the given element) which are uniquely determined by the

vertexes and whose lengthy definitions we do not report here. With the matrices B, D in hand, the following matrices can be obtained as shown in [54, Sections 3.2 and 3.3]:

- $G := BD \in \mathbb{R}^{(d+1)\times(d+1)}$;
- $\widetilde{G} := \begin{bmatrix} 0 & 0 \\ 0 & I_d \end{bmatrix} G \in \mathbb{R}^{(d+1)\times(d+1)}$, where I_d is the $d \times d$ identity matrix;
- $\Pi_*^\nabla := G^{-1}B \in \mathbb{R}^{(d+1)\times \mathtt{NVert}}$;
- $\Pi^\nabla := D\Pi_*^\nabla \in \mathbb{R}^{\mathtt{NVert}\times\mathtt{NVert}}$.

Finally, as shown in [54, Sections 3.3 and 6.2] the local stiffness, consistency and mass matrices are given by

$$K = (\Pi_*^\nabla)^T \widetilde{G} \Pi_*^\nabla + (I - \Pi^\nabla)^T (I - \Pi^\nabla); \tag{A1}$$

$$C = (\Pi_*^\nabla)^T H \Pi_*^\nabla; \tag{A2}$$

$$M = C + \mathrm{Area(F)}(I - \Pi^\nabla)^T (I - \Pi^\nabla). \tag{A3}$$

For simplicity, in (A1)-(A3), the simplest form of VEM stabilization was used, the so-called *dofi-dofi* stabilization, see [62] for a review on VEM stabilization techniques.

A.1 The unit square in 2D

It is possible to show that

$$B = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ -\sqrt{2} & \sqrt{2} & \sqrt{2} & -\sqrt{2} \\ -\sqrt{2} & -\sqrt{2} & \sqrt{2} & \sqrt{2} \end{bmatrix}; \; D = \frac{1}{4}\begin{bmatrix} 4 & -\sqrt{2} & -\sqrt{2} \\ 4 & \sqrt{2} & -\sqrt{2} \\ 4 & \sqrt{2} & \sqrt{2} \\ 4 & -\sqrt{2} & \sqrt{2} \end{bmatrix}; \; H = \frac{1}{24}\begin{bmatrix} 24 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{A4}$$

In particular, the matrices B, D in (A4) were already computed in [54]. It follows that

$$G = \frac{1}{2}\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \qquad \widetilde{G} = \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \tag{A5}$$

$$\Pi_*^\nabla = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ -2\sqrt{2} & 2\sqrt{2} & 2\sqrt{2} & -2\sqrt{2} \\ -2\sqrt{2} & -2\sqrt{2} & 2\sqrt{2} & 2\sqrt{2} \end{bmatrix}; \quad \Pi^\nabla = \frac{1}{4}\begin{bmatrix} 3 & 1 & -1 & 1 \\ 1 & 3 & 1 & -1 \\ -1 & 1 & 3 & 1 \\ 1 & -1 & 1 & 3 \end{bmatrix}. \tag{A6}$$

The final result (10) then follows. To test the correctness of the numerically computed local matrices for the unit square, we use the following script, whose output is commented at the end of Section 4.1.

```
%% Create unit square F and compute local VEM matrices
% Define array P containing coordinates of vertexes of F
P = [0 0 0; 0 1 0; 1 1 0; 1 0 0];
% F is star−shaped w.r.t. the following point P0
P0 = [.5 .5 0];
F = getLocalMatrices(element2d(P, false, false, [], P0));
%% Closed−form local VEM matrices
% Define K,M,C as in equation (10)−full script in the library
%% Evaluate maximum error of numerical local matrices
norm(K(:) − F.K(:),'inf')
norm(C(:) − F.C(:),'inf')
norm(M(:) − F.M(:),'inf')
```

In the above script, in the constructor of `element_2d`, we have set the property `is_square` to `false`. In this way, VEMcomp does not know that `F` is a square and therefore actually computes the local matrices numerically.

A.2 The unit cube in 3D

Using lengthy but simple computations it is possible to show that

$$B = \frac{1}{8\sqrt{3}} \begin{bmatrix} \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} \\ -2 & -2 & -2 & -2 & 2 & 2 & 2 & 2 \\ -2 & -2 & 2 & 2 & -2 & -2 & 2 & 2 \\ -2 & 2 & -2 & 2 & -2 & 2 & -2 & 2 \end{bmatrix}; \tag{A7}$$

$$D = \frac{1}{2\sqrt{3}} \begin{bmatrix} 2\sqrt{3} & -1 & -1 & -1 \\ 2\sqrt{3} & -1 & -1 & 1 \\ 2\sqrt{3} & -1 & 1 & -1 \\ 2\sqrt{3} & -1 & 1 & 1 \\ 2\sqrt{3} & 1 & -1 & -1 \\ 2\sqrt{3} & 1 & -1 & 1 \\ 2\sqrt{3} & 1 & 1 & -1 \\ 2\sqrt{3} & 1 & 1 & 1 \end{bmatrix}; \qquad H = \frac{1}{36} \begin{bmatrix} 36 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{A8}$$

It then follows that

$$G = \frac{1}{3}\begin{bmatrix} 3\,0\,0\,1 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{bmatrix}, \qquad \widetilde{G} = \frac{1}{3}\begin{bmatrix} 0\,0\,0\,1 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{bmatrix}; \tag{A9}$$

$$\Pi_*^\nabla = \frac{1}{8}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -2\sqrt{3} & -2\sqrt{3} & -2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} \\ -2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} & -2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} \\ -2\sqrt{3} & 2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} \end{bmatrix}; \tag{A10}$$

$$\Pi^\nabla = \frac{1}{4}\begin{bmatrix} 2 & 1 & 1 & 0 & 1 & 0 & 0 & -1 \\ 1 & 2 & 0 & 1 & 0 & 1 & -1 & 0 \\ 1 & 0 & 2 & 1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 1 & 2 & -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 2 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 & 2 & 0 & 1 \\ 0 & -1 & 1 & 0 & 1 & 0 & 2 & 1 \\ -1 & 0 & 0 & 1 & 0 & 1 & 1 & 2 \end{bmatrix}. \tag{A11}$$

The final result in equation (12) then follows. To test the correctness of the numerically computed local matrices for the unit cube, we use the following script, whose output is commented at the end of Section 4.2.

```
%% Create the unit cube E and compute its local VEM matrices
P1 = [0 0 0; 0 1 0; 1 1 0; 1 0 0]; % bottom face
E1 = element2d(P1, false, false, [], sum(P1,1)/4);
P2 = [0 0 1; 0 1 1; 1 1 1; 1 0 1]; % top face
E2 = element2d(P2, false, false, [], sum(P2,1)/4);
P3 = [0 0 0; 0 1 0; 0 1 1; 0 0 1]; % back face
E3 = element2d(P3, false, false, [], sum(P3,1)/4);
P4 = [1 0 0; 1 1 0; 1 1 1; 1 0 1]; % front face
E4 = element2d(P4, false, false, [], sum(P4,1)/4);
P5 = [0 0 0; 1 0 0; 1 0 1; 0 0 1]; % left face
E5 = element2d(P5, false, false, [], sum(P5,1)/4);
P6 = [0 1 0; 1 1 0; 1 1 1; 0 1 1]; % right face
E6 = element2d(P6, false, false, [], sum(P6,1)/4);
P = unique([P1; P2; P3; P4; P5; P6], 'rows');
E = getLocalMatrices(element3d(P, [E1;E2;E3;E4;E5;E6], false, [], sum(P,1)/8));
%% Closed−form local VEM matrices
% Define K,M,C as in equation (12)−full script in the library
%% Evaluate maximum error of numerical local matrices
norm(K(:) − E.K(:),'inf')
norm(C(:) − E.C(:),'inf')
norm(M(:) − E.M(:),'inf')
```

Author Contributions All authors contributed equally to the manuscript and approved the submitted manuscript.

Funding Open access funding provided by Università del Salento within the CRUI-CARE Agreement. The work of MF was funded by Regione Puglia (Italy) through the research programme REFIN-Research for Innovation (protocol code 901D2CAA, project number UNISAL026). The work of IS is supported by PRIN2022 PNRR "BAT-MEN" (BATtery Modeling, Experiments & Numerics), Project code P20228C2PP_001, CUP F53D23010020001, funded by MIUR (Italian Ministry of University and Research) and European Union – NextGenerationEU. The work of MF and IS is supported by the research project "Metodi avanzati per la risoluzione di PDEs su griglie strutturate, e non" (INdAM-GNCS project CUP_E53C22001930001). MF and IS are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM). This work (AM) was supported by the Canada Research Chair (Tier 1) in Theoretical and Computational Biology (CRC-2022-00147), the Natural Sciences and Engineering Research Council of Canada (NSERC), Discovery Grants Program (RGPIN-2023-05231), the British Columbia Knowledge Development Fund (BCKDF), Canada Foundation for Innovation – John R. Evans Leaders Fund – Partnerships (CFI-JELF), the British Columbia Foundation for Non-Animal Research, and the UKRI Engineering and Physical Sciences Research Council (EPSRC: EP/J016780/1).

Code Availability The VEMcomp package is available in MF's GitHub repository under the GPLv2 license. The Matlab scripts used in Section 6 and in the Appendix A are also incorporated in the article.

Declarations

Conflicts of Interest The authors have no conflict of interest to declare.

Ethics Approval Not applicable.

References

1. Beirão Da Veiga, L., Brezzi, F., Cangiani, A., Manzini, G., Marini, L.D., Russo, A.: Basic principles of virtual element methods. Math. Models Methods Appl. Sci. **23**(01), 199–214 (2013). https://doi.org/10.1051/m2an/2013138
2. Cangiani, A., Georgoulis, E.H., Pryer, T., Sutton, O.J.: A posteriori error estimates for the virtual element method. Numer. Math. **137**, 857–893 (2017). https://doi.org/10.1007/s00211-017-0891-9
3. Huyssteen, D., Rivarola, F.L., Etse, G., Steinmann, P.: On mesh refinement procedures for the virtual element method for two-dimensional elastic problems. Comput. Methods Appl. Mech. Eng. **393**, 114849 (2022). https://doi.org/10.1016/j.cma.2022.114849
4. Beirão Da Veiga, L., Dassi, F., Russo, A.: A $\mathcal{C}^1$ virtual element method on polyhedral meshes. Comput. Math. App. **79**(7), 1936–1955 (2020). https://doi.org/10.1016/j.camwa.2019.06.019
5. Beirão Da Veiga, L., Manzini, G.: A virtual element method with arbitrary regularity. IMA J. Numer. Anal. **34**(2), 759–781 (2014). https://doi.org/10.1093/imanum/drt018
6. Antonietti, P.F., Beirão Da Veiga, L., Scacchi, S., Verani, M.: A $\mathcal{C}^1$ virtual element method for the Cahn–Hilliard equation with polygonal meshes. SIAM J. Numer. Anal. **54**(1), 34–56 (2016). https://doi.org/10.1137/15M1008117

7. Beirão Da Veiga, L., Russo, A., Vacca, G.: The virtual element method with curved edges. ESAIM: Math. Modell. Numer. Anal. **53**(2), 375–404 (2019). https://doi.org/10.1051/m2an/2018052
8. Bertoluzza, S., Pennacchio, M., Prada, D.: High order VEM on curved domains. Rendiconti Lincei. **30**(2), 391–412 (2019). https://doi.org/10.4171/rlm/853
9. Dassi, F., Fumagalli, A., Losapio, D., Scialò, S., Scotti, A., Vacca, G.: The mixed virtual element method on curved edges in two dimensions. Comput. Methods Appl. Mech. Eng. **386**, 114098 (2021). https://doi.org/10.1016/j.cma.2021.114098
10. Dassi, F., Fumagalli, A., Mazzieri, I., Scotti, A., Vacca, G.: A virtual element method for the wave equation on curved edges in two dimensions. J. Sci. Comput. **90**, 1–25 (2022). https://doi.org/10.1007/s10915-021-01683-w
11. Dassi, F., Fumagalli, A., Scotti, A., Vacca, G.: Bend 3d mixed virtual element method for Darcy problems. Comput. Math. App. **119**, 1–12 (2022). https://doi.org/10.1016/j.camwa.2022.05.023
12. Beirão Da Veiga, L., Lovadina, C., Vacca, G.: Virtual elements for the Navier–Stokes problem on polygonal meshes. SIAM J. Numer. Anal. **56**(3), 1210–1242 (2018). https://doi.org/10.1137/17m1132811
13. Frittelli, M., Sgura, I.: Virtual element method for the Laplace-Beltrami equation on surfaces. ESAIM: Math. Model. Numer. Anal. **52**(3), 965–993 (2018). https://doi.org/10.1051/m2an/2017040
14. Benedetto, M.F., Berrone, S., Pieraccini, S., Scialò, S.: The virtual element method for discrete fracture network simulations. Comput. Methods Appl. Mech. Eng. **280**, 135–156 (2014). https://doi.org/10.1016/j.cma.2014.07.016
15. Beirão Da Veiga, L., Dassi, F., Russo, A.: High-order virtual element method on polyhedral meshes. Comput. Math. App. **74**(5), 1110–1122 (2017). https://doi.org/10.1016/j.camwa.2017.03.021
16. Xiao, L., Zhou, M., Zhao, J.: The nonconforming virtual element method for semilinear elliptic problems. Appl. Math. Comput. **433**, 127402 (2022). https://doi.org/10.1016/j.amc.2022.127402
17. Vacca, G., Beirão Da Veiga, L.: Virtual element methods for parabolic problems on polygonal meshes. Numer. Methods Partial Diff. Equ. **31**(6), 2110–2134 (2015). https://doi.org/10.1002/num.21982
18. Zhao, J., Zhang, B., Zhu, X.: The nonconforming virtual element method for parabolic problems. Appl. Numer. Math. **143**, 97–111 (2019). https://doi.org/10.1016/j.apnum.2019.04.002
19. Adak, D., Natarajan, E., Kumar, S.: Convergence analysis of virtual element methods for semilinear parabolic problems on polygonal meshes. Numer. Methods Partial Diff Equ. **35**(1), 222–245 (2019). https://doi.org/10.1002/num.22298
20. Huang, J., Lin, S.: A posteriori error analysis of a non-consistent virtual element method for reaction diffusion equations. Appl. Math. Lett. **122**, 107531 (2021). https://doi.org/10.1016/j.aml.2021.107531
21. Beirão Da Veiga, L., Brezzi, F., Marini, L.D.: Virtual elements for linear elasticity problems. SIAM J. Numer Anal. **51**(2), 794–812 (2013). https://doi.org/10.1137/120874746
22. Gain, A.L., Talischi, C., Paulino, G.H.: On the virtual element method for three-dimensional linear elasticity problems on arbitrary polyhedral meshes. Comput. Methods Appl. Mech. Eng. **282**, 132–160 (2014). https://doi.org/10.1016/j.cma.2014.05.005
23. Aldakheel, F., Hudobivnik, B., Wriggers, P.: Virtual elements for finite thermo-plasticity problems. Comput. Mech. **64**, 1347–1360 (2019). https://doi.org/10.1007/s00466-019-01714-2
24. Aldakheel, F., Hudobivnik, B., Hussein, A., Wriggers, P.: Phase-field modeling of brittle fracture using an efficient virtual element scheme. Comput. Methods Appl. Mech. Eng. **341**, 443–466 (2018). https://doi.org/10.1016/j.cma.2018.07.008
25. Liu, X., He, Z., Chen, Z.: A fully discrete virtual element scheme for the Cahn-Hilliard equation in mixed form. Comput. Phys. Commun. **246**, 106870 (2020). https://doi.org/10.1016/j.cpc.2019.106870
26. Adak, D., Mora, D., Natarajan, S., Silgado, A.: A virtual element discretization for the time dependent Navier–Stokes equations in stream-function formulation. ESAIM: Math. Model. Numer. Anal. **55**(5), 2535–2566 (2021). https://doi.org/10.1051/m2an/2021058
27. Beirão Da Veiga, L., Mora, D., Vacca, G.: The Stokes complex for virtual elements with application to Navier–Stokes flows. J. Sci. Comput. **81**, 990–1018 (2019). https://doi.org/10.1007/s10915-019-01049-3
28. Bachini, E., Manzini, G., Putti, M.: Arbitrary-order intrinsic virtual element method for elliptic equations on surfaces. Calcolo. **58**(3), 30 (2021). https://doi.org/10.1007/s10092-021-00418-5
29. Frittelli, M., Madzvamuse, A., Sgura, I.: Bulk-surface virtual element method for systems of PDEs in two-space dimensions. Numer. Math. **147**(2), 305–348 (2021). https://doi.org/10.1007/s00211-020-01167-3

30. Frittelli, M., Madzvamuse, A., Sgura, I.: The bulk-surface virtual element method for reaction-diffusion PDEs: analysis and applications. Commun. Comput. Phys. **33**(3), 733–763 (2023). https://doi.org/10.4208/cicp.OA-2022-0204
31. Frittelli, M., Madzvamuse, A., Sgura, I.: Virtual element method for elliptic bulk-surface PDEs in three space dimensions. Numer. Methods Partial Diff. Equ. **39**(6), 4221–4247 (2023). https://doi.org/10.1002/num.23040
32. Frittelli, M., Sgura, I., Bozzini, B.: Turing patterns in a 3D morpho-chemical bulk-surface reaction-diffusion system for battery modeling. Math. Eng. **6**(2) (2024). https://doi.org/10.3934/mine.2024015
33. Wells, H., Hubbard, M.E., Cangiani, A.: A velocity-based moving mesh virtual element method. Comput. Math. App. **155**, 110–125 (2024). https://doi.org/10.1016/j.camwa.2023.12.005
34. Sutton, O.J.: The virtual element method in 50 lines of MATLAB. Numer. Algo. **75**(4), 1141–1159 (2016). https://doi.org/10.1007/s11075-016-0235-3
35. Herrera, C., Corrales-Barquero, R., Arroyo-Esquivel, J., Calvo, J.G.: A numerical implementation for the high-order 2D virtual element method in MATLAB. Numer. Algo. **92**(3), 1707–1721 (2022). https://doi.org/10.1007/s11075-022-01361-4
36. Dhanush, V., Natarajan, S.: Implementation of the virtual element method for coupled thermo-elasticity in Abaqus. Numer. Algo. **80**(3), 1037–1058 (2018). https://doi.org/10.1007/s11075-018-0516-0
37. Ortiz-Bernardin, A.: VEMLAB version 2.4.1. https://camlab.cl/software/vemlab/. Accessed 03 Jul 2023
38. Ortiz-Bernardin, A., Alvarez, C., Hitschfeld-Kahler, N., Russo, A., Silva-Valenzuela, R., Olate-Sanzana, E.: Veamy: an extensible object-oriented C++ library for the virtual element method. Numer. Algo. **82**(4), 1189–1220 (2019). https://doi.org/10.1007/s11075-018-00651-0
39. Yu, Y.: mVEM: a MATLAB software package for the virtual element methods. (2022). https://doi.org/10.48550/arXiv.2204.01339 arXiv preprint arXiv:2204.01339
40. Savaré, S., Chanon, O.: VEM3D. https://github.com/deatinor/VEM3D. Accessed 19 Apr 2024
41. Persson, P.-O., Strang, G.: A simple mesh generator in MATLAB. SIAM Rev. **46**(2), 329–345 (2004). https://doi.org/10.1137/S0036144503429121
42. Frittelli, M., Madzvamuse, A., Sgura, I., Venkataraman, C.: Preserving invariance properties of reaction-diffusion systems on stationary surfaces. IMA J. Numer. Anal. **39**(1), 235–270 (2017). https://doi.org/10.1093/imanum/drx058
43. Frittelli, M., Madzvamuse, A., Sgura, I., Venkataraman, C.: Lumped finite elements for reaction-cross-diffusion systems on stationary surfaces. Comput. Math. App. **74**(12), 3008–3023 (2017). https://doi.org/10.1016/j.camwa.2017.07.044
44. Lacitignola, D., Bozzini, B., Frittelli, M., Sgura, I.: Turing pattern formation on the sphere for a morphochemical reaction-diffusion model for electrodeposition. Commun. Nonlinear Sci. Numer. Simul. **48**, 484–508 (2017). https://doi.org/10.1016/j.cnsns.2017.01.008
45. Lacitignola, D., Frittelli, M., Cusimano, V., De Gaetano, A.: Pattern formation on a growing oblate spheroid. an application to adult sea urchin development. J. Comput. Dyn. **9**(2), 185–206 (2022). https://doi.org/10.3934/jcd.2021027
46. Talischi, C., Paulino, G.H., Pereira, A., Menezes, I.F.M.: PolyMesher: a general-purpose mesh generator for polygonal elements written in Matlab. Struct. Multidiscip. Optim. **45**(3), 309–328 (2012). https://doi.org/10.1007/s00158-011-0706-z
47. Alvarez, C.: Delynoi: An object-oriented C++ library for the generation of polygonal meshes. https://camlab.cl/software/delynoi/. Accessed 18 Apr 2024
48. Salinas-Fernández, S., Hitschfeld-Kahler, N.: Polylla: Polygonal/polyhedral meshing algorithm based on terminal-edge regions and terminal-face regions. (2023). arXiv preprint arXiv:2310.03665
49. Ji, W.: 3D voronoi via fem mesh, the voronoi3d class, visualization. https://it.mathworks.com/matlabcentral/fileexchange/94955-3d-voronoi-via-fem-mesh-the-voronoi3d-class-visualization/. Accessed 03 May 2024
50. Beirão Da Veiga, L., Lovadina, C., Russo, A.: Stability analysis for the virtual element method. Math. Models Methods Appl. Sci. **27**(13), 2557–2594 (2017). https://doi.org/10.1142/s021820251750052x
51. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. ACM Siggraph Comput. Graph. **21**(4), 163–169 (1987). https://doi.org/10.1145/37402.37422
52. FEniCS Project: Collection of Documented Demos 10. Generate Mesh. (2024). https://fenicsproject.org/olddocs/dolfin/1.4.0/python/demo/documented/mesh-generation/python/documentation.html. Accessed 22 Nov 2023

53. deal.II: Reference Documentation for deal.II - Grids and Triangulations. https://www.dealii.org/developer/doxygen/deal.II/group__grid.html. Accessed 22 Nov 2023
54. Beirão Da Veiga, L., Brezzi, F., Marini, L.D., Russo, A.: The hitchhiker's guide to the virtual element method. Math. Models Methods Appl. Sci. **24**(08), 1541–1573 (2014). https://doi.org/10.1142/S021820251440003X
55. MathWorks: Partial Differential Equation Toolbox User's Guide. (2024). https://it.mathworks.com/help/pdf_doc/pde/pde.pdf
56. Madzvamuse, A., Ndakwo, H., Barreira, R.: Stability analysis of reaction-diffusion models on evolving domains: The effects of cross-diffusion. Discret. Contin. Dyn. Syst. **36**(4), 2133–2170 (2015). https://doi.org/10.3934/dcds.2016.36.2133
57. Lacitignola, D., Bozzini, B., Sgura, I.: Spatio-temporal organization in a morphochemical electrodeposition model: Hopf and Turing instabilities and their interplay. Eur. J. Appl. Math. **26**(2), 143–173 (2015). https://doi.org/10.1017/s0956792514000370
58. Ahrens, J., Geveci, B., Law, C.: ParaView: An end-user tool for large data visualization. In: Hansen, C.D., Johnson, C.R. (eds.) Visualization Handbook, pp. 717–731. Butterworth-Heinemann, Burlington (2005). https://doi.org/10.1016/b978-012387582-2/50038-1
59. Tuncer, N., Madzvamuse, A., Meir, A.: Projected finite elements for reaction-diffusion systems on stationary closed surfaces. Appl. Numer. Math. **96**, 45–71 (2015). https://doi.org/10.1016/j.apnum.2014.12.012
60. Elliott, C.M., Ranner, T.: Finite element analysis for a coupled bulk-surface partial differential equation. IMA J. Numer. Anal. **33**(2), 377–402 (2013). https://doi.org/10.1093/imanum/drs022
61. Madzvamuse, A., Chung, A.H.W.: The bulk-surface finite element method for reaction–diffusion systems on stationary volumes. Finite Elem. Anal. Des. **108**, 9–21 (2016). https://doi.org/10.1016/j.finel.2015.09.002
62. Mascotto, L.: The role of stabilization in the virtual element method: a survey. Comput. Math. App. **151**, 244–251 (2023). https://doi.org/10.1016/j.camwa.2023.09.045

Authors and Affiliations

Massimo Frittelli[1] · Anotida Madzvamuse[2,3,4,5] · Ivonne Sgura[1]

✉ Massimo Frittelli
massimo.frittelli@unisalento.it

Anotida Madzvamuse
am823@math.ubc.ca

Ivonne Sgura
ivonne.sgura@unisalento.it

1 Department of Mathematics and Physics "E. De Giorgi", Via per Arnesano, University of Salento, 73100 Lecce, Italy

2 Mathematics Department, The University of British Columbia, Vancouver, British Columbia, Canada

3 Department of Mathematics and Applied Mathematics, University of Pretoria, Pretoria 0132, South Africa

4 Department of Mathematics and Applied Mathematics, University of Johannesburg, PO Box 524, Auckland Park 2006, South Africa

5 Department of Mathematics and Computational Science, Faculty of Science, University of Zimbabwe, Mount Pleasant, Harare, Zimbabwe