VEMcomp: A Virtual Element MATLAB tool for semilinear parabolic PDEs in 2D and 3D

Massimo Frittelli, Anotida Madzvamuse, Ivonne Sgura

October 12, 2023

Abstract

We present a Virtual Element MATLAB solver for elliptic and parabolic, linear and semilinear 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 PDEs posed on different simple and complex geometries, involving bulk, surface, and bulk-surface PDE probems. The solver employs the Virtual Element Method of lowest polynomial order $k = 1$ on general polygonal and polyhedral meshes. VEMcomp has three purposes. First, for special geometries, VEMcomp generates polygonal and polyhedral meshes optimized for fast matrix assembly. Second, given a mesh for the considered geometry, possibly generated with an external program, VEMcomp computes all the matrices of the method. Third, for multiple classes of bulk, surface and bulk-surface PDEs, VEMcomp solves the considered PDE problem with the VEM, through a user-friendly interface. An extensive set of examples illustrates the usage of the library and the optimal convergence rate of the method.

1 Introduction

The Virtual Element Method (VEM) was first proposed in [8] for elliptic problems in two space dimensions as a generalization of the Finite Element Method, where the mesh elements can be general polygons instead of triangles. The usage of polygons with arbitrarily many 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 [19, 41], (ii) numerical solutions with high global regularity [11, 14], (iii) accurate approximation of boundaries [16, 18, 21], (iv) easy mesh pasting [13, 29], and (v) easy handling of complex domain shapes and cuts [17, 20].

Motivated by its multiple benefits, the VEM was quickly extended to numerous PDE problems and applications. A non-exhaustive list of models for which the VEMs has been employed comprises of (i) linear elliptic problems in two [8, 16] and three [10] space dimensions, (ii) semilinear elliptic problems in two and three space dimensions [43], (iii) linear heat equation in two [40] and three [44] space dimensions, (iv) semilinear parabolic equations [2] and reaction-diffusion systems [32], (v) elasticity [7, 30] and plasticity [4] problems, (vi) phase-field models [3, 5], (vii) fluid dynamics [1, 15], (viii) fracture models [17], (ix) surface [6, 29] and bulk-surface [24, 25, 26] PDEs, and recently (x) PDEs on evolving flat domains [42].

Over ten years of its existence, the VEM has established itself as a reliable technology with desirable properties. This has stimulated the development of the first open-source VEM libraries and codes. Here we will recall some of these libraries. The work in [38] 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 [31]. An Abaqus-MATLAB VEM code for coupled thermo-elasticity problems in 2D is presented in [22]. VEM libraries for elasticity problems in 2D are available in MATLAB/Octave [36] and C++ [37]. All the mentioned libraries are dedicated to specific cases or applications and are confined to the two space dimensional setting. To the best of the authors' knowledge, there is no open-source VEM library so far for PDE problems in three space dimensions and/or of surface or bulk-surface type. In this work, we contribute to the field of VEM open-source libraries. We propose a MATLAB library, which we call VEMcomp, for elliptic and parabolic PDE problems in 2D and 3D, including bulk, surface and bulk-surface PDE problems. VEMcomp has three purposes:

1. For special geometries, both in 2D and 3D, such as rectangles, ellipses, parallelepipeds and ellipsoids, the library generates polygonal and polyhedral meshes specifically optimized for fast matrix assembly, following [24, 25];

2. Given any polygonal or polyhedral mesh -not necessarily generated with VEMcomp itself-, the library generates all the matrices involved in the method (e.g. mass, stiffness). For polygonal mesh generation, it is worth recalling the MATLAB library `PolyMesher` [39];

3. For multiple classes of PDE problems, the library performs matrix assembly and provides a black-box interface that allows the user to set the problem parameters, and returns the VEM 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 [28, 27] and bulk-surface PDEs [24, 25].

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 polygonal and polyhedral meshes. Section 4 showcases VEMcomp's ability to compute local and global VEM matrices. In Section 5 we present VEMcomp's user-friendly solvers and solution plotters for the model problems outlined in Section 2. Section 6 lists several numerical examples that illustrate at once the usage of VEMcomp and the optimal convergence of the Virtual Element Method. 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 (i) PDE problems in three space dimensions and (ii) PDE problems on complex geometries, such as surface or bulk-surface PDEs. 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 elliptic and parabolic, linear and semilinear, bulk-only PDE systems of $n \in \mathbb{N}$ equations, and is given by

$$\begin{cases} \left[\dfrac{\partial u_i}{\partial t}\right] - d_i^\Omega \Delta u_i = f_i(u_1, \dots, u_n, \boldsymbol{x}, [t]), \qquad \boldsymbol{x} \in \Omega, \qquad i = 1, \dots, n; \\ u_i = 0 \quad \text{or} \quad \dfrac{\partial u_i}{\partial \boldsymbol{n}} = 0, \qquad \boldsymbol{x} \in \partial\Omega, \qquad i = 1, \dots, n, \end{cases} \tag{1}$$

where Δ denotes the Laplace operator in Ω, $d_1^\Omega, \dots, d_n^\Omega > 0$ are diffusion coefficients, $T > 0$ is the final time, $f_1, \dots, f_n$ are smooth enough linear or nonlinear functions. In (1), the expressions between square brackets appear in the parabolic (time-dependent) case, but not in the elliptic (stationary) case. The general model (1) comprises several notable bulk-only PDE problems that were solved with VEM in the literature, such as (i) linear [8, 10, 16] and semilinear elliptic problems [43], (ii) linear [40] and semilinear parabolic problems [2] including reaction-diffusion systems [32]. If $\Gamma = \partial\Omega$ is a sufficiently smooth manifold with empty boundary, a second class of PDEs that fall in VEMcomp's domain is the following class of surface PDEs or systems of $m \in \mathbb{N}$ surface PDEs:

$$\left[\frac{\partial u_j}{\partial t}\right] - d_j^\Gamma \Delta_\Gamma u_j = g_j(u_1, \dots, u_n, \boldsymbol{x}, [t]), \qquad \boldsymbol{x} \in \Gamma, \qquad j = 1, \dots, m, \tag{2}$$

where Δ_Γ represents the Laplace 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. In (2), no boundary conditions are needed since, as we mentioned, Γ has no boundary. In (2), the expressions between square brackets appear only in the time-dependent case. The general model (2) encompasses several

surface PDE (SPDE) models of interest, such as elliptic SPDEs [6, 29] and surface reaction-diffusion systems (SRDSs) [33, 34].
The third and most complex class of PDE problems addressed in this work is given by bulk-surface PDEs of the following form:

$$\begin{cases} \left[\dfrac{\partial u_i}{\partial t}\right] - d_i^\Omega \Delta u_i = f_i(u_1, \dots, u_n, \boldsymbol{x}, [t]), & \boldsymbol{x} \in \Omega, \qquad i = 1, \dots, n; \\ \left[\dfrac{\partial v_j}{\partial t}\right] - d_j^\Gamma \Delta_\Gamma v_j = g_j(u_1, \dots, u_n, v_1, \dots, v_m, \boldsymbol{x}, [t]), & \boldsymbol{x} \in \Gamma, \qquad j = 1, \dots, m; \\ \dfrac{\partial u_i}{\partial \boldsymbol{n}} = h_i(u_1, \dots, u_n, v_1, \dots, v_m, \boldsymbol{x}, [t]), & \boldsymbol{x} \in \Gamma, \qquad i = 1, \dots, n, \end{cases} \tag{3}$$

where Δ and Δ_Γ represents the Laplace operator in Ω and the Laplace-Beltrami operator on Γ, $d_1^\Omega, \dots, d_n^\Omega, d_1^\Gamma, \dots, d_m^\Gamma > 0$ are diffusion coefficients, $f_1, \dots, f_n, g_1, \dots, g_m, h_1, \dots, h_n$ are smooth enough linear or nonlinear functions, and $T > 0$ is the final time. In (3), the expressions between square brackets appear only in the time-dependent case. We recall that the VEM was extended to bulk-surface reaction-diffusion systems (BSRDSs) in two [24] and three [25] space dimensions, which fall within the general class (3).
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 (iii) matrix assembly and user-friendly solver for problems (1), (2) and (3).

3 Mesh generation and representation

Polygonal and polyhedral mesh generation is a niche topic, and very little easy to use software is available. For domains in two space dimensions, we mention the software PolyMesher [39]. To the best of the authors' knowledge, there is no open-source software for polyhedral mesh generation in three space dimensions. For special three dimensional domains, 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 countless open-source generators exist.
We start by presenting basic classes that allow to represent single elements in two and three space dimensions.

3.1 The class `element2d_dummy`

The class `element2d_dummy` represents a polygonal element in 2D. It contains minimal information that uniquely identify the element. To create an element with `NVert` vertexes, use the following constructor

```
obj = element2d_dummy(P);
```

where `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. When an `element2d_dummy` is created, the following properties are set by the above constructor

```
properties (SetAccess = private)
        P(:,3) double % Coordinates of vertexes
        NVert(1,1) double % Number of vertexes
end
```

When needed, an `element2d_dummy` can be provided with optional information contained in the following optional properties:

```
properties (SetAccess = private)
        P_ind(:,1) double = []
```

```
        is_boundary(1,1) logical
        is_square(1,1) logical
end
```

defined as follows:

- If the `element2d_dummy` is part of a mesh, and the array `PP` of size `NMesh` $\times$ 3 contains the coordinates of all the nodes, then the property `P_ind` contains the indexes of the nodes `P` of the `element2d_dummy`, i.e. `PP(P_ind,:)  = P`;
- The logical `is_boundary` determines if the `element2d_dummy` 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 VEM matrices;
- The logical `is_square` determines if the `element2d_dummy` is a square (for which the local VEM matrices are known in closed form).

3.2 The class `element3d_dummy`

The class `element3d_dummy` represent a polyhedral element in 3D. It contains minimal information that uniquely identify the element. To create an element with `NVert` vertexes and `NFaces` faces, use the following constructor

```
        obj = element3d_dummy(P,Faces)
```

where `P` is a `NVert` $\times$ 3 array containing the coordinates of the vertexes in any order and `Faces` is a `NFaces` $\times$ 1 array of `element2d_dummy`. When an `element3d_dummy` is created, the following properties are set by the above constructor

```
properties (SetAccess = private)
        P(:,3) double % Coordinates of vertexes
        Faces(:,1) element2d_dummy % Faces
end
```

When needed, an `element3d_dummy` can be provided, through the above constructor, with the following optional properties:

```
properties (SetAccess = private)
        Pind(:,1) double = []
        iscube(1,1) logical
end
```

defined as follows:

- If the `element3d_dummy` is part of a mesh, and the array `PP` of size `NMesh` $\times$ 3 contains the coordinates of all the nodes, then the property `P_ind` contains the indexes of the nodes `P` of the `element3d_dummy`, i.e. `PP(P_ind,:)  = P`;
- The logical `is_cube` determines if the `element3d_dummy` is a cube (for which the local VEM matrices are known in closed form).

3.3 Generating meshes for special geometries

We can now state that any polygonal mesh in 2D can be represented as a collection of `element2d_dummy`, while any polthedral mesh in 3D can be represented as a collection of `element3d_dummy`. Even if the user is free to write custom code to generate meshes as collections of `element2d_dummy` or `element3d_dummy`, VEMcomp comes with two functions for the generation of meshes in this form, for domains that are defined as level sets of Lipschitz functions. We will start with the 2D case. Let $Q \subset \mathbb{R}^2$ be a closed 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

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^2 \mid f(\boldsymbol{x}) \leq 0\}, \quad \text{and} \quad \Gamma = \{\boldsymbol{x} \in \mathbb{R}^2 \mid f(\boldsymbol{x}) = 0\}. \tag{4}$$

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 [35]. However, our purpose is to produce a polygonal bulk-surface mesh of Ω. 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. $\Gamma_h = \partial\Omega_h$. In other words, we adopt a bulk-surface variant of the marching square algorithm that produces a conforming bulk-surface mesh. This approach improves and generalizes the strategy proposed in our previous work in [24]. VEMcomp implements the proposed algorithm in the function `generate_mesh_2d`, whose syntax is as follows

```
[P, h, BulkElements, SurfaceElements] =
      generate_mesh_2d(fun, box, Nx, tol);
```

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

- `fun` is the level function used in (4), 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 $\mathbb{R}^2$;
- `box` is a 2×2 array that defines the bounding box $Q =$ `box(1,:)` $\times$ `box(2,:)`;
- `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;
- `tol` > 0 is the minimum distance between distinct nodes. Any two nodes closer than `tol` will be merged into a unique node, in order to avoid excessively distorted and ill-conditioned elements;
- `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, and thanks to conformity, 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_dummy` format;
- `SurfaceElements` is a $M \times 2$ array, where $M \in \mathbb{N}$ is the number of surface elements, that defines the surface elements in the following way, which is standard in finite element codes: for each $i = 1, \ldots, M$, the nodes of the i-th surface element (a segment) are `P(SurfaceElements(i,:),:)`.

In a similar way, we address mesh generation in 3D. Let $Q \subset \mathbb{R}^2$ be a 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

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^3 \mid f(\boldsymbol{x}) \leq 0\}, \quad \text{and} \quad \Gamma = \{\boldsymbol{x} \in \mathbb{R}^3 \mid f(\boldsymbol{x}) = 0\}. \tag{5}$$

Similarly to the 2D case, we generate the surface mesh Γ_h with the well-known *marching cube* algorithm [35]. For the bulk mesh Ω_h, we simply use the cubes and the cut cubes produced as a by-product of the marching cubes algorithm. Once again, we have the conformity property that $\Gamma_h = \partial\Omega_h$. This bulk-surface variant of the marching cube algorithm was proposed in our work [26] and is now implemented in VEMcomp in the function `generate_mesh_3d`, whose syntax is as follows

```
[P, h, BulkElements, SurfaceElements]
      = generate_mesh_3d(fun, box, Nx, tol);
```

In the above, inputs and outputs are analogous to the 2D case: `fun` (this time as in (5)), `box` (this time 3×2), `Nx`, `tol`, `P` (this time $N \times 3$), `h`, `BulkElements` (this time a list of `element3d_dummy`), `SurfaceElements` (this time a list of `element2d_dummy`).

4 Computation of local and global matrices

VEMcomp has two classes specifically designed for the computation of the local VEM matrices of lowest polynomial order $k = 1$ (stiffness, mass, and consistency matrices) in two and three space dimensions: `element2d` and `element3d`, respectively. Moreover, VEMcomp provides two functions for the assembly of global matrices in 2D and 3D: `generate_matrices_2d` and `generate_matrices_3d`, respectively. All these aspects will be covered in this Section. Before starting, we remark that VEMcomp's matrix computation facilities rely on the following mesh regularity assumptions:

(A1) in 2D, every (polygonal) element is star-shaped w.r.t. at least one point.

(A2) in 3D, every (polyhedral) element is star-shaped w.r.t. at least one point and so are all of its faces.

Assumptions (A1)-(A2) are not restrictive, as they are standard throughout the VEM literature, see for instance [12] for 2D and [10] for 3D. Moreover, we point out that all meshes generated with the `generate_mesh_2d` and `generate_mesh_3d` functions presented in the previous Section are guaranteed to fulfil (A1)-(A2), and are thus suitable as inputs to VEMcomp's matrix computation tools.

4.1 The class `element2d`

The class `element2d` represents a polygonal element in 2D with `NVert` vertexes that is star-shaped w.r.t. at least one point. To create an instance of the class, use the following constructor

```
obj = element2d(P, P0);
```

where

- `P` is a `NVert` $\times$ 3 matrix whose rows are the coordinates of the ordered vertexes.
- `P0`, of size 1×3, is a point w.r.t. which the element is star-shaped.

Upon initialisation, the object stores `P` and `P0` and automatically computes several other `properties` of the element:

```
properties(SetAccess = private)
        P(:,3) double
        P0(1,3) double
        NVert(1,1) double
        Area(1,1) double
        OrientedArea(1,3) double
        Centroid (1,3) double
        Diameter(1,1) double
        K(:,:) double
        M(:,:) double
end
```

that can be queried from the object. In the above:

- `NVert` is the number of vertexes
- `Area` is the surface area of the element
- `OrientedArea` is a vector orthogonal to the element whose modulus is the element area
- `Centroid` is the centroid of the element
- `Diameter` is the diameter of the element
- `K` is the local stiffness matrix
- `M` is the local mass matrix

The usage of `element2d` will be demonstrated in Section 4.3.

4.2 The class `element3d`

The class `element3d` represents a polygonal element in 3D with `NVert` vertexes that is star-shaped w.r.t. at least one point and whose faces fulfill the same property. To create an instance of the class, use the following constructor

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

where

- `Faces` is a `NFaces` $\times$ 1 array of `element2d` representing the faces
- `P` is a `NVert` $\times$ 3 matrix whose rows are the coordinates of the vertexes
- `P0` is a point w.r.t. which the element is star-shaped.

We remark that, even if the vertexes `P` are already contained in the `Faces`, the `property P` is still needed to specify vertex ordering. Upon initialisation, the object stores `Faces`, `P`, and `P0` and automatically computes several other public `properties` of the element:

```
properties(SetAccess = private)
        Faces(:,1) element2d
        P(:,3) double
        P0(1,3) double
        NVert(1,1) double
        NFaces(1,1) double
        Volume(1,1) double
        Centroid(1,3) double
        Diameter(1,1) double
        K(:,:) double
        M(:,:) double
end
```

that can be queried from the object. In the above:

- `NVert` is the number of vertexes and `NFaces` is the number of faces
- `Volume`, `Centroid` and `Diameter` are self-explanatory
- `K` is the local stiffness matrix and `M` is the local mass matrix.

The usage of `element3d` will be demonstrated later on.

4.3 A worked example in 2D: the unit square

Here we will show the usage of `element2d` to compute the local matrices of the unit square, thereby presenting the closed-form counterpart. Consider the unit square contained in the xy-plane:

$$F = \{(x, y, z) \in \mathbb{R}^3 | (x, y) \in [0, 1]^2,\ z = 0\}, \tag{6}$$

which can be thought of as the polygon enclosed by the vertexes $(0, 0, 0)$, $(0, 1, 0)$, $(1, 1, 0)$, and $(1, 0, 0)$. Notice that node ordering affects the resulting matrices. We start by computing the closed form of the VEM local mass and stiffness matrices of F for the lowest order case $k = 1$.
As shown in [9], the computation of the local mass and stiffness matrices relies on three fundamental matrices $B \in \mathbb{R}^{3\times\texttt{NVert}}$, $D \in \mathbb{R}^{\texttt{NVert}\times 3}$, $H \in \mathbb{R}^{3\times 3}$, which are uniquely determined by the vertexes and whose lengthy definitions we do not report here. With the matrices B, D, H in hand, the following matrices can be obtained:

- $G := BD \in \mathbb{R}^{3\times 3}$;
- $\widetilde{G} := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} G \in \mathbb{R}^{3\times 3}$;

- $\Pi^\nabla_* := G^{-1}B \in \mathbb{R}^{3\times \texttt{NVert}}$;
- $\Pi^\nabla := D\Pi^\nabla_* \in \mathbb{R}^{\texttt{NVert}\times\texttt{NVert}}$.

Finally, the local stiffness and mass matrices are given by

$$K = (\Pi^\nabla_*)^T \widetilde{G} \Pi^\nabla_* + (I - \Pi^\nabla)^T (I - \Pi^\nabla); \tag{7}$$
$$M = (\Pi^\nabla_*)^T H \Pi^\nabla_* + \mathrm{Area(F)} (I - \Pi^\nabla)^T (I - \Pi^\nabla). \tag{8}$$

For the unit square F, as shown in [9], it holds that

$$B = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -\sqrt{2} & \sqrt{2} & \sqrt{2} & -\sqrt{2} \\ -\sqrt{2} & -\sqrt{2} & \sqrt{2} & \sqrt{2} \end{bmatrix}; \tag{9}$$
$$D = \frac{1}{4} \begin{bmatrix} 4 & -\sqrt{2} & -\sqrt{2} \\ 4 & \sqrt{2} & -\sqrt{2} \\ 4 & \sqrt{2} & \sqrt{2} \\ 4 & -\sqrt{2} & \sqrt{2} \end{bmatrix}; \tag{10}$$
$$H = \frac{1}{24} \begin{bmatrix} 24 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{11}$$

It follows that

$$G = \frac{1}{2} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \widetilde{G} = \frac{1}{2} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \tag{12}$$
$$\Pi^\nabla_* = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2\sqrt{2} & 2\sqrt{2} & 2\sqrt{2} & -2\sqrt{2} \\ -2\sqrt{2} & -2\sqrt{2} & 2\sqrt{2} & 2\sqrt{2} \end{bmatrix}; \tag{13}$$
$$\Pi^\nabla = \frac{1}{4} \begin{bmatrix} 3 & 1 & -1 & 1 \\ 1 & 3 & 1 & -1 \\ -1 & 1 & 3 & 1 \\ 1 & -1 & 1 & 3 \end{bmatrix}. \tag{14}$$

We finally obtain the local stiffness and mass matrices:

$$K = \frac{1}{4} \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}; \tag{15}$$
$$M = \frac{1}{48} \begin{bmatrix} 17 & -9 & 13 & -9 \\ -9 & 17 & -9 & 13 \\ 13 & -9 & 17 & -9 \\ -9 & 13 & -9 & 17 \end{bmatrix}. \tag{16}$$

We now show the usage of `element2d` to compute the matrices K and M numerically. To this end, we need to create an object of class `element2d`. To call the constructor, we define the array `P`

```
P = [0 0 0; 0 1 0; 1 1 0; 1 0 0];
```

containing the vertexes of `F` and the array `P0` as

```
P0 = [.5 .5 0];
```

because F is star-shaped w.r.t. `P0`. Notice that, since F is convex, `P0` can be chosen as any point in F, even a vertex. We are ready to create the object:

```
F = element2d(P, P0)
```

Because there is no semicolon in the above command, the following output appears in the command window:

```
E1 =

  element2d with properties:

                P: [4x3 double]
               P0: [0.5000 0.5000 0]
            NVert: 4
             Area: 1
     OrientedArea: [0 0 -1]
         Centroid: [0.5000 0.5000 0]
         Diameter: 1.4142
                K: [4x4 double]
                M: [4x4 double]
```

In this specific case, the `Centroid` coincides with `P0`. By querying the stiffness and mass matrices of `F` (with `format rat` for better readability), we can see the outputs

```
>> E1.K

ans =

       3/4          -1/4          -1/4          -1/4
      -1/4           3/4          -1/4          -1/4
      -1/4          -1/4           3/4          -1/4
      -1/4          -1/4          -1/4           3/4

>> E1.M

ans =

      17/48         -3/16         13/48         -3/16
      -3/16         17/48         -3/16         13/48
      13/48         -3/16         17/48         -3/16
      -3/16         13/48         -3/16         17/48
```

which agree with (15)-(16).

4.4 A worked example in 3D: the unit cube

Here we will show the usage of `element3d` to compute the local matrices of the unit cube $E = [0,1]^3$, thereby presenting the closed-form counterpart. Because vertex ordering is reflected in the resulting matrices, we order the vertexes as follows:

$$(0,0,0)\ (0,0,1)\ (0,1,0)\ (0,1,1)\ (1,0,0)\ (1,0,1)\ (1,1,0)\ (1,1,1). \tag{17}$$

We start by computing the closed form of the VEM local mass and stiffness matrices of E for the lowest order case $k = 1$. As shown in [9], the computation of the mass and stiffness matrices relies on three fundamental matrices, similarly to the 2D case:

- $B \in \mathbb{R}^{4\times \texttt{NVert}}$;
- $D \in \mathbb{R}^{\texttt{NVert}\times 4}$;
- $H \in \mathbb{R}^{4\times 4}$,

whose lengthy definitions we do not report here. With the above matrices in hand, the following matrices can be obtained:

- $G := BD \in \mathbb{R}^{4\times 4}$;
- $\widetilde{G} := \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} G \in \mathbb{R}^{4\times 4}$;
- $\Pi_*^\nabla := G^{-1}B \in \mathbb{R}^{4\times \texttt{NVert}}$;
- $\Pi^\nabla := D\Pi_*^\nabla \in \mathbb{R}^{\texttt{NVert}\times\texttt{NVert}}$.

Finally, the local stiffness and mass matrices are given by

$$K = (\Pi_*^\nabla)^T \widetilde{G} \Pi_*^\nabla + \mathrm{Diam}(E)(I - \Pi^\nabla)^T (I - \Pi^\nabla); \tag{18}$$

$$M = (\Pi_*^\nabla)^T H \Pi_*^\nabla + \mathrm{Volume(F)}(I - \Pi^\nabla)^T (I - \Pi^\nabla). \tag{19}$$

For the unit cube E, it is possible to show analytically that

$$B = \frac{1}{8\sqrt{3}} \begin{bmatrix} \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} & \sqrt{3} \\ -2 & -2 & -2 & -2 & 2 & 2 & 2 & 2 \\ -2 & -2 & 2 & 2 & -2 & -2 & 2 & 2 \\ -2 & 2 & -2 & 2 & -2 & 2 & -2 & 2 \end{bmatrix}; \tag{20}$$

$$D = \frac{1}{2\sqrt{3}} \begin{bmatrix} 2\sqrt{3} & -1 & -1 & -1 \\ 2\sqrt{3} & -1 & -1 & 1 \\ 2\sqrt{3} & -1 & 1 & -1 \\ 2\sqrt{3} & -1 & 1 & 1 \\ 2\sqrt{3} & 1 & -1 & -1 \\ 2\sqrt{3} & 1 & -1 & 1 \\ 2\sqrt{3} & 1 & 1 & -1 \\ 2\sqrt{3} & 1 & 1 & 1 \end{bmatrix}; \tag{21}$$

$$H = \frac{1}{36} \begin{bmatrix} 36 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{22}$$

It follows that

$$G = \frac{1}{3} \begin{bmatrix} 3 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \widetilde{G} = \frac{1}{3} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \tag{23}$$

$$\Pi_*^\nabla = \frac{1}{8} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -2\sqrt{3} & -2\sqrt{3} & -2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} \\ -2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} & -2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & 2\sqrt{3} \\ -2\sqrt{3} & 2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} & -2\sqrt{3} & 2\sqrt{3} \end{bmatrix}; \tag{24}$$

$$\Pi^\nabla = \frac{1}{4} \begin{bmatrix} 2 & 1 & 1 & 0 & 1 & 0 & 0 & -1 \\ 1 & 2 & 0 & 1 & 0 & 1 & -1 & 0 \\ 1 & 0 & 2 & 1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 1 & 2 & -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 2 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 & 2 & 0 & 1 \\ 0 & -1 & 1 & 0 & 1 & 0 & 2 & 1 \\ -1 & 0 & 0 & 1 & 0 & 1 & 1 & 2 \end{bmatrix}. \tag{25}$$

We finally obtain the analytical local stiffness and mass matrices:

$$K = \frac{1}{16}\begin{bmatrix} 3 & 1 & 1 & -1 & 1 & -1 & -1 & -3 \\ 1 & 3 & -1 & 1 & -1 & 1 & -3 & -1 \\ 1 & -1 & 3 & 1 & -1 & -3 & 1 & -1 \\ -1 & 1 & 1 & 3 & -3 & -1 & -1 & 1 \\ 1 & -1 & -1 & -3 & 3 & 1 & 1 & -1 \\ -1 & 1 & -3 & -1 & 1 & 3 & -1 & 1 \\ -1 & -3 & 1 & -1 & 1 & -1 & 3 & 1 \\ -3 & -1 & -1 & 1 & -1 & 1 & 1 & 3 \end{bmatrix} + \frac{\sqrt{3}}{4}\begin{bmatrix} 2 & -1 & -1 & 0 & -1 & 0 & 0 & 1 \\ -1 & 2 & 0 & -1 & 0 & -1 & 1 & 0 \\ -1 & 0 & 2 & -1 & 0 & 1 & -1 & 0 \\ 0 & -1 & -1 & 2 & 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 2 & -1 & -1 & 0 \\ 0 & -1 & 1 & 0 & -1 & 2 & 0 & -1 \\ 0 & 1 & -1 & 0 & -1 & 0 & 2 & -1 \\ 1 & 0 & 0 & -1 & 0 & -1 & -1 & 2 \end{bmatrix}; \tag{26}$$

$$M = \frac{1}{96}\begin{bmatrix} 51 & -22 & -22 & 1 & -22 & 1 & 1 & 24 \\ -22 & 51 & 1 & -22 & 1 & -22 & 24 & 1 \\ -22 & 1 & 51 & -22 & 1 & 24 & -22 & 1 \\ 1 & -22 & -22 & 51 & 24 & 1 & 1 & -22 \\ -22 & 1 & 1 & 24 & 51 & -22 & -22 & 1 \\ 1 & -22 & 24 & 1 & -22 & 51 & 1 & -22 \\ 1 & 24 & -22 & 1 & -22 & 1 & 51 & -22 \\ 24 & 1 & 1 & -22 & 1 & -22 & -22 & 51 \end{bmatrix}. \tag{27}$$

We now show the usage of `element3d` to compute the matrices K and M numerically. To this end, we need to create an object of class `element3d`. To call the constructor, we first need to create six instances of `element2d` representing the faces of E:

```
P1 = [0 0 0; 0 1 0; 1 1 0; 1 0 0]; % bottom face
E1 = element2d(P1, sum(P1,1)/4);

P2 = [0 0 1; 0 1 1; 1 1 1; 1 0 1]; % top face
E2 = element2d(P2, sum(P2,1)/4);

P3 = [0 0 0; 0 1 0; 0 1 1; 0 0 1]; % back face
E3 = element2d(P3, sum(P3,1)/4);

P4 = [1 0 0; 1 1 0; 1 1 1; 1 0 1]; % front face
E4 = element2d(P4, sum(P4,1)/4);

P5 = [0 0 0; 1 0 0; 1 0 1; 0 0 1]; % left face
E5 = element2d(P5, sum(P5,1)/4);

P6 = [0 1 0; 1 1 0; 1 1 1; 0 1 1]; % right face
E6 = element2d(P6, sum(P6,1)/4);
```

For each of the faces, we have chosen `P0` as the midpoint of its vertexes for convenience, but of course other choices are possible since every face is convex and thus star-shaped w.r.t. every point of the face itself. We are ready to create the `element3d`:

```
P = unique([P1; P2; P3; P4; P5; P6],'rows');
E = element3d([E1;E2;E3;E4;E5;E6], P, sum(P,1)/8);
```

We have used the command `unique` to extract a set of all vertexes with no repetitions. MATLAB will sort the vertexes in `P` in "increasing order", that is as in (17). Again, the `P0` is chosen as the midpoint of all vertexes for convenience. Let us have a look at the 3D element `E`:

```
>> E

E =

  element3d with properties:

        Faces: [6x1 element2d]
            P: [8x3 double]
           P0: [0.5000 0.5000 0.5000]
        NVert: 8
       NFaces: 6
       Volume: 1.0000
     Centroid: [0.5000 0.5000 0.5000]
     Diameter: 1.7321
            K: [8x8 double]
            M: [8x8 double]
```

By querying the numerically computed stiffness and mass matrices of `E`, we can see the outputs

```
>> E.K

ans =

    1.0535   -0.3705   -0.3705   -0.0625   -0.3705   -0.0625   -0.0625    0.2455
   -0.3705    1.0535   -0.0625   -0.3705   -0.0625   -0.3705    0.2455   -0.0625
   -0.3705   -0.0625    1.0535   -0.3705   -0.0625    0.2455   -0.3705   -0.0625
   -0.0625   -0.3705   -0.3705    1.0535    0.2455   -0.0625   -0.0625   -0.3705
   -0.3705   -0.0625   -0.0625    0.2455    1.0535   -0.3705   -0.3705   -0.0625
   -0.0625   -0.3705    0.2455   -0.0625   -0.3705    1.0535   -0.0625   -0.3705
   -0.0625    0.2455   -0.3705   -0.0625   -0.3705   -0.0625    1.0535   -0.3705
    0.2455   -0.0625   -0.0625   -0.3705   -0.0625   -0.3705   -0.3705    1.0535

>> E.M

ans =

    17/32   -11/48   -11/48     1/96   -11/48     1/96     1/96     1/4
   -11/48    17/32     1/96   -11/48     1/96   -11/48     1/4      1/96
   -11/48     1/96    17/32   -11/48     1/96     1/4    -11/48     1/96
     1/96   -11/48   -11/48    17/32     1/4      1/96     1/96   -11/48
   -11/48     1/96     1/96     1/4     17/32   -11/48   -11/48     1/96
     1/96   -11/48     1/4      1/96   -11/48    17/32     1/96   -11/48
     1/96     1/4    -11/48     1/96   -11/48     1/96    17/32   -11/48
     1/4      1/96     1/96   -11/48     1/96   -11/48   -11/48    17/32
```

which agree with (26)-(27) up to machine precision.

4.5 Global matrix assembly

Once a mesh has been generated in the format returned by the functions `generate_mesh_2d` or `generate_mesh_3d` in Section 3, VEMcomp provides two functions for global matrix assembly in 2D and 3D. The 2D case is covered by the `assemble_matrices_2d`, whose syntax is as follows:

```
[K,M,KS,MS,R] = assemble_matrices_2d(P,ElementsBulk,ElementsSurface);
```

In the above, the inputs are as returned by the function `generate_mesh_2d`, while the outputs are defined as follows

- `K`, `M` are the stiffness and mass matrices in the bulk, stored as sparse matrices;
- `KS`, `MS` are the stiffness and mass matrices on the surface, stored as sparse matrices;
- `R` is the reduction matrix.

For full definitions of the above matrices, see [24]. Analogously, for matrix assembly in 3D, VEMcomp provides the function `assemble_matrices_3d`, whose syntax is

```
[K,M,KS,MS,R] = assemble_matrices_3d(P,ElementsBulk,ElementsSurface);
```

In the above, the inputs are as returned by the function `generate_mesh_3d` in Section 3.3, while the outputs are defined as in the 2D case. For full definitions of these matrices in the 3D case, see [26]. The function `assemble_matrices_3d` can take an optional input `xcut` and an optional output `ElementsPlot` for plotting purposes, defined as follows. The plot is obtained by cutting the bulk at a specified abscissa `xcut` (optional input). The function `assemble_matrices_3d` then returns the optional output `ElementsPlot`, a polyhedral surface of the cut bulk, as a list of `element2d_dummy`.

5 User-friendly solver and plotter

Given a bulk-surface mesh in 3D and the respective global matrices, VEMcomp solves the elliptic BSPDE (3) with the script `ELLIPTIC_BS_3D.m`. For ease of presentation, the script is confined to the case $m = n = 1$ of one bulk species and one surface species. The user can easily modify the script to address the case of arbitrary m and n. The user can set the input to the script using the first four lines, as in the following example:

```
load('mesh_and_matrices.mat');
f1 = @(P) P(:,1).^2 + P(:,2).^2 + P(:,3).^2 - 6;
g1 = @(P) P(:,1).^2 + P(:,2).^2 + P(:,3).^2 + 1;
h1 = @(P) P(:,1).^2 + 8;
```

In the above listing, the first line loads a `.mat` file that contains a 3D bulk-surface mesh and its respective matrices, previously constructed with the `generate_mesh_3d` and `assemble_matrices_3d` functions. The `f1`, `g1`, and `h1` functions are as in the right-hand sides in (3). If the `.mat` file contains the optional output `ElementsPlot` produced by the `assemble_matrices_3d` function, then the `ELLIPTIC_BS_3D.m` script will produce a plot of the numerical solution.

6 Numerical examples

We now present an extensive list of numerical experiments carried out in VEMcomp. The experiments showcase the usage of VEMcomp and the optimal convergence of the Virtual Element Method in its various forms.

6.1 Bulk-only problems

This first set of examples showcases the application of VEMcomp to bulk-only PDE problems of increasing complexity. In Example 6.1.1 we consider a Poisson problem in 2D on a circle with Dirichlet boundary conditions. In Example 6.1.2 we show the solution of a Poisson problem in 3D on the sphere .

6.1.1 Poisson problem on two-dimensional domain

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

$$\begin{cases} -\Delta u = 2\pi^2 \sin \pi x \sin \pi y, & (x,y) \in \Omega; \\ u(x,y) = \sin \pi x \sin \pi y, & (x,y) \in \partial\Omega, \end{cases} \tag{28}$$

whose exact solution is given by

$$u(x, y) = \sin \pi x \sin \pi y, \qquad (x, y) \in \Omega. \tag{29}$$

We consider a sequence of four meshes $i = 1, \dots, 4$. The i-th mesh is obtained with the marching squares algorithm as explained in Section 3.3. On each mesh we solve the discrete problem, we compute the error in $L^2(\Omega)$ norm and the respective convergence rate. As shown in Table [TODO] the convergence in $L^2(\Omega)$ norm is optimal, i.e. quadratic. The numerical solution obtained on the finest mesh is plotted in Fig. [TODO].

6.1.2 Poisson equation on the sphere

We now consider a 3D domain, the unit sphere Ω in 3D. The test problem, in spherical coordinates, is as follows

$$\begin{cases} - \quad \Delta u + u = 4(3 - 5(x^2 + y^2 + z^2)^2) + (1 - (x^2 + y^2 + z^2)^2)^2 \qquad \text{in } \Omega; \\ \nabla u \cdot \boldsymbol{n} = 0 \qquad \text{on } \partial\Omega, \end{cases} \tag{30}$$

whose exact solution in spherical coordinates is given by

$$u(x, y, z) = (1 - (x^2 + y^2 + z^2))^2. \tag{31}$$

We consider a sequence of four cubic meshes $i = 1, \dots, 4$. The i-th mesh is obtained by subdividing each dimension into $5i$ intervals, thereby producing a cubic bounding mesh. The cubic elements that are not fully contained in the sphere are then cut. This procedure is described in detail in [26]. Such a mesh is shown in Fig. 1. On each mesh we solve the discrete problem, we compute the error in $L^2(\Omega)$ norm and the respective convergence rate. As shown in Table 1, the convergence in $L^2(\Omega)$ norm is optimal, i.e. quadratic. The numerical solution obtained on the finest mesh is plotted in Fig. 2.

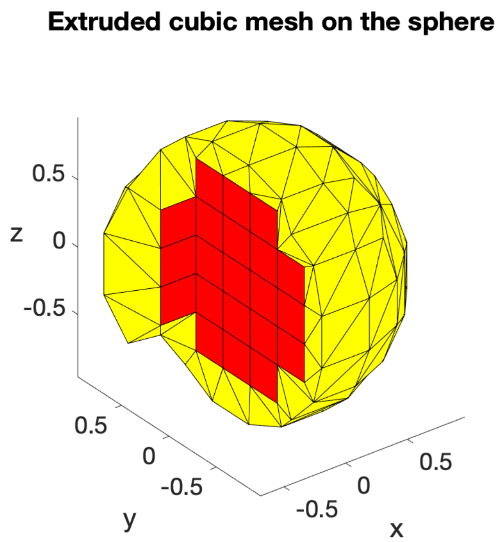

Figure 1: Example of an extruded cubic mesh on the unit sphere. The outermost elements, highlighted in yellow are irregular (but still star-shaped or even convex) 8-vertex polyhedra. The interior is filled with cubes (red).

Table 1: Poisson equation (30) on the unit sphere Ω in 3D. The VEM implemented in VEMcomp shows optimal quadratic convergence in $L^2(\Omega)$ norm.

i	N	h	$L^2(\Omega)$ error	$L^2(\Omega)$ rate
1	111	0.6928	1.3767	-
2	799	0.3464	4.4137e-01	1.6412
3	5749	0.1732	1.2532e-01	1.8164
4	40381	0.0866	3.3139e-02	1.9190

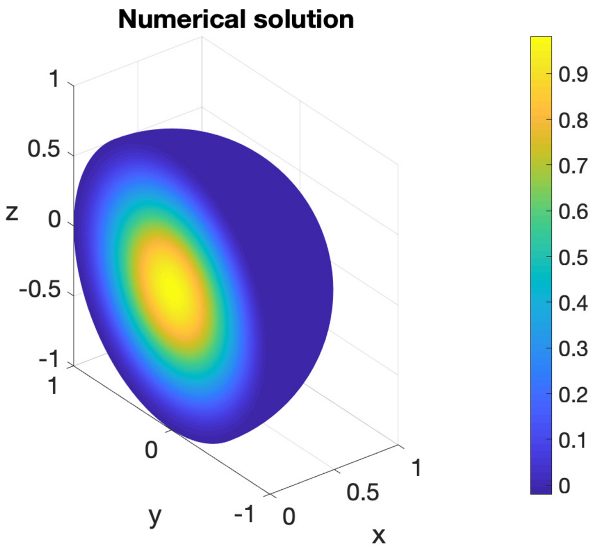

Figure 2: Poisson equation (30) on the unit sphere Ω in 3D: numerical solution obtained on the finest mesh for $i = 4$ with $N = 40381$ nodes.

6.2 Numerical examples for bulk-surface problems

We now apply VEMcomp to different types of BSPDEs. In Example 6.2.1 we show a baseline problem given by a bulk-surface linear elliptic problem on the sphere. Example 6.2.2 introduces the time variable, by addressing the bulk-surface linear heat equation on the sphere. In Example 6.2.3, we use VEMcomp to solve the top-end PDE problem considered in this work: a bulk-surface reaction-diffusion system (BSRDS) on the sphere.

6.2.1 Elliptic bulk-surface problem on the sphere

We numerically solve the following elliptic bulk-surface problem, found in [23], on the unit sphere Ω in 3D:

$$\begin{cases} -\quad \Delta u + u = xyz \qquad \text{in } \Omega \\ -\quad \Delta_\Gamma v + v + \nabla u \cdot \boldsymbol{n} = 29xyz \qquad \text{on } \partial\Omega \\ \quad\ \nabla u \cdot \boldsymbol{n} = -u + 2v \qquad \text{on } \partial\Omega \end{cases} \tag{32}$$

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 consider the same sequence of four meshes Ω_i, $i = 1, 2, 3, 4$ of Experiment 6.1.2. The surface meshes are induced by the corresponding bulk mesh, i.e. $\Gamma_i = \partial\Omega_i$, $i = 1, \ldots, 4$. On each mesh we solve the discrete problem, we compute the error in $L^2(\Omega) \times L^2(\Gamma)$ norm and the respective convergence rate. As shown in Table 2, the convergence in $L^2(\Omega) \times L^2(\Gamma)$ norm is optimal, i.e. quadratic. The numerical solution obtained on the finest mesh is plotted in Fig. 3.

Table 2: Elliptic bulk-surface problem (32) on the unit sphere Ω in 3D. The VEM implemented in VEMcomp shows optimal quadratic convergence in $L^2(\Omega) \times L^2(\Gamma)$ norm. Times required for the solution of the linear system are shown.

i	N	h	$L^2(\Omega) \times L^2(\Gamma)$ error	$L^2(\Omega) \times L^2(\Gamma)$ rate	Time (s)
1	111	0.6928	2.1985e-01	-	0.002159
2	799	0.3464	3.8387e-02	2.5178	0.015645
3	5749	0.1732	8.5674e-03	2.1637	0.197641
4	40381	0.0866	1.9884e-03	2.1072	5.994934

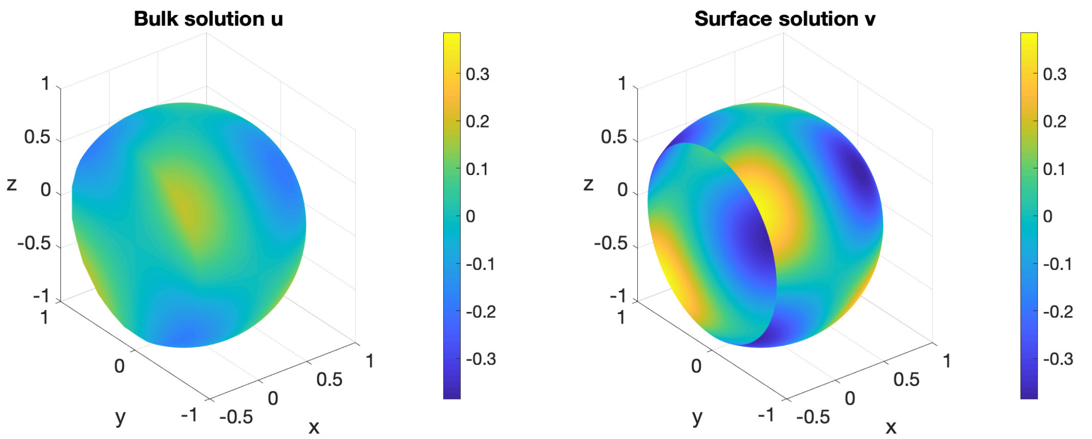

Figure 3: Elliptic bulk-surface problem (32) on the unit sphere Ω in 3D: numerical solution obtained on the finest mesh for $i = 4$ with $N = 40381$ nodes. Left: bulk component u. Right: surface component v.

6.2.2 Parabolic bulk-surface problem on the sphere

We numerically solve the following parabolic bulk-surface problem, found in [24], on the unit sphere Ω in 3D:

$$\begin{cases} \dot{u} - \Delta u = xyze^t & \text{in } \Omega \times [0, T]; \\ \dot{v} - \Delta_\Gamma v + \nabla u \cdot \boldsymbol{n} = 16xyze^t & \text{on } \partial\Omega \times [0, T]; \\ \nabla u \cdot \boldsymbol{n} = 3xyze^t & \text{on } \partial\Omega \times [0, T], \end{cases} \tag{33}$$

for final time $T = 1$, whose exact solution is given by

$$\begin{aligned} u(x, y, z, t) &= xyze^t, \qquad (x, y, z, t) \in \Omega \times [0, T]; \\ v(x, y, z, t) &= 2xyze^t, \qquad (x, y, z, t) \in \partial\Omega \times [0, T]. \end{aligned}$$

We consider the same sequence of four meshes Ω_i, $i = 1, 2, 3, 4$ of Experiment 6.2.1. Correspondingly, we choose timesteps $\tau_i = 2^{1-i}$, $i = 1, 2, 3, 4$. On each mesh we solve the discrete problem, we compute the error in $L^2(\Omega) \times L^2(\Gamma)$ norm at the final time $T = 1$ and the respective convergence rate. As shown in Table 3, the convergence in $L^2(\Omega) \times L^2(\Gamma)$ norm is optimal, i.e. quadratic in space and linear in time. The numerical solution at the final time obtained on the finest mesh is plotted in Fig. 4.

Table 3: Parabolic bulk-surface problem (33) on the unit sphere Ω in 3D. The VEM implemented in VEMcomp shows optimal quadratic convergence in $L^2(\Omega) \times L^2(\Gamma)$ norm. Times required for the time integration are shown.

i	N	h	τ	$L^2(\Omega) \times L^2(\Gamma)$ error	$L^2(\Omega) \times L^2(\Gamma)$ rate	Time (s)
1	111	0.6928	1	1.2074	-	0.002417
2	799	0.3464	2.5e-1	4.3481e-01	1.4734	0.038881
3	5749	0.1732	6.25e-2	1.2110e-01	1.8442	2.134601
4	40381	0.0866	1.5625e-2	3.0881e-02	1.9714	287.345570

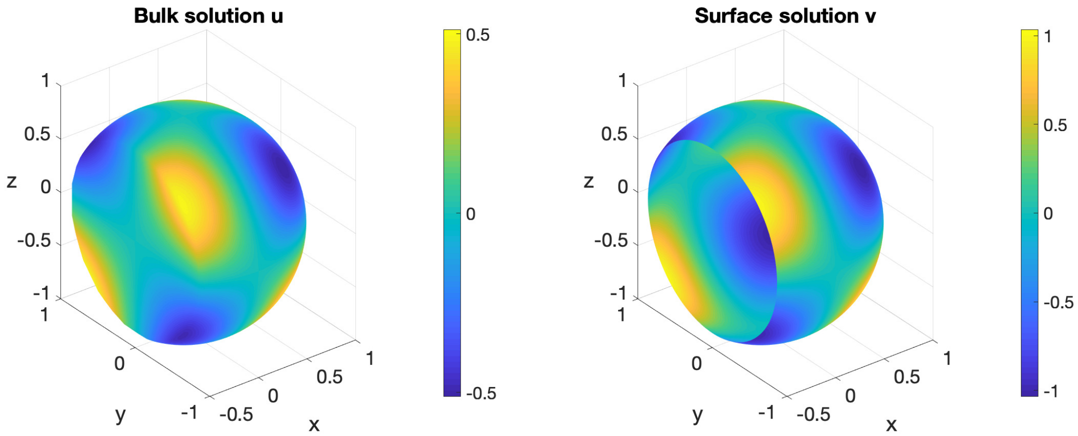

Figure 4: Parabolic bulk-surface problem (33) on the unit sphere Ω in 3D: numerical solution obtained on the finest mesh for $i = 4$ with $N = 40381$ nodes and timestep $\tau = 1.5625e - 2$. Left: bulk component u. Right: surface component v.

6.2.3 Bulk-surface reaction-diffusion system on the sphere

In this final example we solve the following BSRDS considered in [25]. [TODO].

7 Conclusions

The present VEMcomp package is intended as a proof-of-concept, with the main goal of being user-friendly and self-explicative. For this reason, VEMcomp has room for improvement in terms of computational performances. To this end, two main challenges are (i) devising cheaper mesh generation strategies and (ii) devising cheaper quadrature formulas in 2D and 3D.

References

[1] D Adak, D Mora, S Natarajan, and A Silgado. A virtual element discretization for the time dependent Navier–Stokes equations in stream-function formulation. *ESAIM: Mathematical Modelling and Numerical Analysis*, 55(5):2535–2566, 2021. doi:10.1051/m2an/2021058.

[2] D Adak, E Natarajan, and S Kumar. Convergence analysis of virtual element methods for semilinear parabolic problems on polygonal meshes. *Numerical Methods for Partial Differential Equations*, 35(1):222–245, 2019. doi:10.1002/num.22298.

[3] F Aldakheel, B Hudobivnik, A Hussein, and P Wriggers. Phase-field modeling of brittle fracture using an efficient virtual element scheme. *Computer Methods in Applied Mechanics and Engineering*, 341:443–466, 2018. doi:10.1016/j.cma.2018.07.008.

[4] F Aldakheel, B Hudobivnik, and P Wriggers. Virtual elements for finite thermo-plasticity problems. *Computational Mechanics*, 64:1347–1360, 2019. doi:10.1007/s00466-019-01714-2.

[5] P F Antonietti, L Beirão Da Veiga, S Scacchi, and M Verani. A $\mathcal{C}^1$ virtual element method for the Cahn–Hilliard equation with polygonal meshes. *SIAM Journal on Numerical Analysis*, 54(1):34–56, 2016. doi:10.1137/15M1008117.

[6] E Bachini, G Manzini, and M Putti. Arbitrary-order intrinsic virtual element method for elliptic equations on surfaces. *Calcolo*, 58(3):30, 2021. doi:10.1007/s10092-021-00418-5.

[7] L Beirão Da Veiga, F Brezzi, and L D Marini. Virtual elements for linear elasticity problems. *SIAM Journal on Numerical Analysis*, 51(2):794–812, 2013. doi:10.1137/120874746.

[8] L Beirão da Veiga, F Brezzi, A Cangiani, G Manzini, L D Marini, and A Russo. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23(01):199–214, 2013. doi:10.1051/m2an/2013138.

[9] L Beirão da Veiga, F Brezzi, L D Marini, and A Russo. The hitchhiker's guide to the virtual element method. *Mathematical models and methods in applied sciences*, 24(08):1541–1573, 2014. doi:10.1142/S021820251440003X.

[10] L Beirão Da Veiga, F Dassi, and A Russo. High-order virtual element method on polyhedral meshes. *Computers & Mathematics with Applications*, 74(5):1110–1122, 2017. doi:10.1016/j.camwa.2017.03.021.

[11] L Beirão da Veiga, F Dassi, and A Russo. A $\mathcal{C}^1$ virtual element method on polyhedral meshes. *Computers & Mathematics with Applications*, 79(7):1936–1955, 2020. doi:10.1016/j.camwa.2019.06.019.

[12] L Beirão da Veiga, C Lovadina, and A Russo. Stability analysis for the virtual element method. *Mathematical Models and Methods in Applied Sciences*, 27(13):2557–2594, 2017. doi:10.1142/s021820251750052x.

[13] L Beirão Da Veiga, C Lovadina, and G Vacca. Virtual elements for the Navier–Stokes problem on polygonal meshes. *SIAM Journal on Numerical Analysis*, 56(3):1210–1242, 2018. doi:10.1137/17m1132811.

[14] L Beirão da Veiga and G Manzini. A virtual element method with arbitrary regularity. *IMA Journal of Numerical Analysis*, 34(2):759–781, 2014. doi:10.1093/imanum/drt018.

[15] L Beirão da Veiga, D Mora, and G Vacca. The Stokes complex for virtual elements with application to Navier–Stokes flows. *Journal of Scientific Computing*, 81:990–1018, 2019. doi:10.1007/s10915-019-01049-3.

[16] L Beirão Da Veiga, A Russo, and G Vacca. The virtual element method with curved edges. *ESAIM: Mathematical Modelling and Numerical Analysis*, 53(2):375–404, 2019. doi:10.1051/m2an/2018052.

[17] M F Benedetto, S Berrone, S Pieraccini, and S Scialò. The virtual element method for discrete fracture network simulations. *Computer Methods in Applied Mechanics and Engineering*, 280:135–156, 2014. doi:10.1016/j.cma.2014.07.016.

[18] S Bertoluzza, M Pennacchio, and D Prada. High order VEM on curved domains. *Rendiconti Lincei*, 30(2):391–412, 2019. doi:10.4171/rlm/853.

[19] A Cangiani, E H Georgoulis, T Pryer, and O J Sutton. A posteriori error estimates for the virtual element method. *Numerische mathematik*, 137:857–893, 2017. doi:10.1007/s00211-017-0891-9.

[20] F Dassi, A Fumagalli, I Mazzieri, A Scotti, and G Vacca. A virtual element method for the wave equation on curved edges in two dimensions. *Journal of Scientific Computing*, 90:1–25, 2022. doi:10.1007/s10915-021-01683-w.

[21] F Dassi, A Fumagalli, A Scotti, and G Vacca. Bend 3d mixed virtual element method for darcy problems. *Computers & Mathematics with Applications*, 119:1–12, 2022. doi:10.1016/j.camwa.2022.05.023.

[22] V. Dhanush and S. Natarajan. Implementation of the virtual element method for coupled thermo-elasticity in abaqus. *Numerical Algorithms*, 80(3):1037–1058, 2018. doi:10.1007/s11075-018-0516-0.

[23] C M Elliott and T Ranner. Finite element analysis for a coupled bulk–surface partial differential equation. *IMA Journal of Numerical Analysis*, 33(2):377–402, 2013. doi:10.1093/imanum/drs022.

[24] M Frittelli, A Madzvamuse, and I Sgura. Bulk-surface virtual element method for systems of PDEs in two-space dimensions. *Numerische Mathematik*, 147(2):305–348, 2021. doi:10.1007/s00211-020-01167-3.

[25] M Frittelli, A Madzvamuse, and I Sgura. The bulk-surface virtual element method for reaction-diffusion PDEs: analysis and applications. *Communications in Computational Physics*, Accepted for publication, 2023. doi:10.4208/cicp.OA-2022-0204.

[26] M Frittelli, A Madzvamuse, and I Sgura. Virtual element method for elliptic bulk-surface PDEs in three space dimensions. *Numerical Methods for Partial Differential Equations*, 39(6):4221–4247, 2023. doi:10.1002/num.23040.

[27] M Frittelli, A Madzvamuse, I Sgura, and C Venkataraman. Lumped finite elements for reaction–cross–diffusion systems on stationary surfaces. *Computers & Mathematics with Applications*, 74(12):3008–3023, 2017. doi:10.1016/j.camwa.2017.07.044.

[28] M Frittelli, A Madzvamuse, I Sgura, and C Venkataraman. Preserving invariance properties of reaction–diffusion systems on stationary surfaces. *IMA Journal of Numerical Analysis*, 39(1):235–270, 2017. doi:10.1093/imanum/drx058.

[29] M Frittelli and I Sgura. Virtual element method for the Laplace-Beltrami equation on surfaces. *ESAIM: Mathematical Modelling and Numerical Analysis*, 52(3):965–993, 2018. doi:10.1051/m2an/2017040.

[30] A L Gain, C Talischi, and G H Paulino. On the virtual element method for three-dimensional linear elasticity problems on arbitrary polyhedral meshes. *Computer Methods in Applied Mechanics and Engineering*, 282:132–160, 2014. doi:10.1016/j.cma.2014.05.005.

[31] C Herrera, R Corrales-Barquero, J Arroyo-Esquivel, and J G Calvo. A numerical implementation for the high-order 2D virtual element method in MATLAB. *Numerical Algorithms*, 92(3):1707–1721, 2022. doi:10.1007/s11075-022-01361-4.

[32] J Huang and S Lin. A posteriori error analysis of a non-consistent virtual element method for reaction diffusion equations. *Applied Mathematics Letters*, 122:107531, 2021. doi:10.1016/j.aml.2021.107531.

[33] D Lacitignola, B Bozzini, M Frittelli, and I Sgura. Turing pattern formation on the sphere for a morphochemical reaction-diffusion model for electrodeposition. *Communications in Nonlinear Science and Numerical Simulation*, 48:484–508, 2017. doi:10.1016/j.cnsns.2017.01.008.

[34] D Lacitignola, M Frittelli, V Cusimano, and A De Gaetano. Pattern formation on a growing oblate spheroid. an application to adult sea urchin development. *Journal of Computational Dynamics*, 9(2):185–206, 2022. doi:10.3934/jcd.2021027.

[35] W E Lorensen and H E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. doi:10.1145/37402.37422.

[36] A Ortiz-Bernardin. Vemlab version 2.4.1. Accessed: 2023-03-07. URL: https://camlab.cl/software/vemlab/.

[37] A Ortiz-Bernardin, C Alvarez, N Hitschfeld-Kahler, A Russo, R Silva-Valenzuela, and E Olate-Sanzana. Veamy: an extensible object-oriented C++ library for the virtual element method. *Numerical Algorithms*, 82(4):1189–1220, 2019. doi:10.1007/s11075-018-00651-0.

[38] O J Sutton. The virtual element method in 50 lines of MATLAB. *Numerical Algorithms*, 75(4):1141–1159, 2016. doi:10.1007/s11075-016-0235-3.

[39] C Talischi, G H Paulino, A Pereira, and I F M Menezes. PolyMesher: a general-purpose mesh generator for polygonal elements written in matlab. *Structural and Multidisciplinary Optimization*, 45(3):309–328, 2012. doi:10.1007/s00158-011-0706-z.

[40] G Vacca and L Beirão da Veiga. Virtual element methods for parabolic problems on polygonal meshes. *Numerical Methods for Partial Differential Equations*, 31(6):2110–2134, 2015. doi:10.1002/num.21982.

[41] D van Huyssteen, F L Rivarola, G Etse, and P Steinmann. On mesh refinement procedures for the virtual element method for two-dimensional elastic problems. *Computer Methods in Applied Mechanics and Engineering*, 393:114849, 2022. doi:10.1016/j.cma.2022.114849.

[42] H Wells, M E Hubbard, and A Cangiani. A velocity-based moving mesh virtual element method, 2022. arXiv:2211.13521.

[43] L Xiao, M Zhou, and J Zhao. The nonconforming virtual element method for semilinear elliptic problems. *Applied Mathematics and Computation*, 433:127402, 2022. doi:10.1016/j.amc.2022.127402.

[44] J Zhao, B Zhang, and X Zhu. The nonconforming virtual element method for parabolic problems. *Applied Numerical Mathematics*, 143:97–111, 2019. doi:10.1016/j.apnum.2019.04.002.