Project 2 - TMA4220 - 2023

Deadline : November 17, at 11:59 pm
Grade splitting: 80% project submission, 20% presentation
Weight on the final grade : 25%

To submit, send an email to `davide.murari@ntnu.no` attaching a folder with

- the Python/Matlab code,
- a PDF file with the explanations of the obtained results.

An alternative would be a Jupyter notebook with the theoretical derivations and code. Specify the group and the group members when you submit.

For this project you may choose one of the following three topics. You may choose between using a high-level finite element library in options 2-3 and a low-level implementation in option 1. If choosing options 2-3 you are recommended to use this notebook to familiarise yourself with Firedrake.

Presentations

During lecture time on 22 and 23 November each group will give a short presentation (up to 15 minutes) on your project. This presentation can include the following:

- What is the idea of the project?
- What did you discover from the project?
- Some nice figures from the project (solution plots, convergence plots, etc).
- What were the main difficulties?
- If you could take this project further what would you do?
- How was the work distributed between members of the group?

Option 1: Manual assembly of finite elements

We consider the same 2d problem as in the first project:

$$\begin{cases} -\Delta u(x,y) = f(x,y),\ (x,y) \in (0,1)^2, \\ u(x,y) = 0,\ x \in \{0,1\},\ \text{or },y \in \{0,1\}. \end{cases}$$

Throughout the project description, we refer to $[0,1]^2$ as Ω. We want to compare the solution we get using a triangular mesh over the unit square, with the tensor product one used in the first project. We will do so

both in the energy norm $|\cdot|_{H^1(\Omega)}$ and in the L_2-norm $\|\cdot\|_{L_2(\Omega)}$. Let us first start building the solution over a trianglular mesh.

For every task of the project, you are allowed to use any method from libraries like *Scipy* to approximate the necessary integrals, but are not permitted to use tools designed for finite element libraries. If you have any doubt on whether you are allowed to use a library please ask.

Task 1.1 Building the necessary finite element tools

Task 1.1.1 Get acquainted with the mesh generator

You can find a Python script that helps generating a mesh $\mathcal{T}_h$ over the unit square by clicking here. You will need to get various meshes and also access the boundary nodes of such meshes throughout the project. In this first exercise, you are tasked to plot 3 meshes with $n = 11, 21, 51$ nodes on the edges. Additionally, write a method called *getBoundaryNodes* that returns the list of indices corresponding to the boundary nodes when given as input the nodes of the mesh. You'll find guidelines about the mesh generating script in the script itself and in the readme file of the repository.

Task 1.1.2 Build the stiffness matrix for linear elements

Let us focus on the reference triangle of vertices $p_1 = (0,0)$, $p_2 = (1,0)$, $p_3 = (0,1)$. As already seen in the second exercise sheet, the linear basis functions on the reference triangle $T_r = \mathrm{conv}\{\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3\}$ write

$$\varphi_1(\xi_1,\xi_2) = 1 - \xi_1 - \xi_2,\ \varphi_2(\xi_1,\xi_2) = \xi_1,\ \varphi_3(\xi_1,\xi_2) = \xi_2.$$

We recall that the stiffness matrix defined on the reference triangle is a 3×3 matrix A_r of entries

$$(A_r)_{ij} = \int\int_{T_r} \langle \nabla\varphi_i(x,y), \nabla\varphi_j(x,y)\rangle \mathrm{d}x\mathrm{d}y.$$

Use the change of variables based on barycentric coordinates seen in the second exercise sheet, and write a method *computeLocalStiffnessDeg1* that takes as input the 3 vertices $\hat{p}_1, \hat{p}_2, \hat{p}_3$ of a generic triangle T, and returns the stiffness matrix A_T for the triangle $T = \mathrm{conv}\{\hat{\mathrm{p}}_1, \hat{\mathrm{p}}_2, \hat{\mathrm{p}}_3\}$, i.e. a 3×3 matrix with entries

$$(A_T)_{ij} = \int\int_{T} \hat{\varphi}_i(x,y)\hat{\varphi}_j(x,y)\mathrm{d}x\mathrm{d}y$$

where $\hat{\varphi}_1, \hat{\varphi}_2, \hat{\varphi}_3$ are the linear basis functions defined on T and hence such that $\hat{\varphi}_i(\hat{p}_j) = \delta_{ij}$ for $i,j = 1,2,3$.

The meshing script you have available provides an array called *tri* which determines a local to global mapping. In other words, each row of *tri* determines the index of the vertices of a specific triangle in terms of the full mesh. Use this local-to-global mapping to assemble the full stiffness matrix based on the method *computeLocalStiffnessDeg1* you just wrote. More explicitly, write a method called *stiffnessMatrixDeg1* which takes as input the arrays *p*, *tri* provided by the mesh generator, and returns the global stiffness matrix A.

Task 1.1.3 Build the stiffness matrix for quadratic elements

We now want to do the same as in the previous task, but for piecewise quadratic elements. To do so, let us first introduce the expression of the quadratic basis functions on the reference triangle, we remark there are 6 basis function, and we add degrees of freedom at mid points of all the three edges to fully determine quadratic polynomials (cfr. quadratic Lagrangian elements). We write these six degrees of freedom as

$p_1 = (0,0)$, $p_2 = (1,0)$, $p_3 = (0,1)$, $p_4 = (1/2,0)$, $p_5 = (1/2,1/2)$, $p_6 = (0,1/2)$. The basis functions written in barycentric coordinates are:

$$\varphi_1(\xi_1,\xi_2) = (1-\xi_1-\xi_2)(1-2\xi_1-2\xi_2),\ \varphi_2(\xi_1,\xi_2) = \xi_1(2\xi_1-1),$$
$$\varphi_3(\xi_1,\xi_2) = \xi_2(2\xi_2-1)$$
$$\varphi_4(\xi_1,\xi_2) = 4\xi_1(1-\xi_1-\xi_2),\ \varphi_5(\xi_1,\xi_2) = 4\xi_1\xi_2,\ \varphi_6(\xi_1,\xi_2) = 4\xi_2(1-\xi_1-\xi_2).$$

Write a method called *computeLocalStiffnessDeg2* which computes the local stiffness matrix $A_T \in \mathbb{R}^{6\times 6}$ associated to a the triangle $T = \text{conv}\{\hat{p}_1, \hat{p}_2, \hat{p}_3\}$ given as inputs the vertices $\hat{p}_1, \hat{p}_2, \hat{p}_3$.

We now need to assemble the global stiffness matrix as in the case of linear basis functions. To do so we again need the local-to-global mapping provided by *tri*, however there is an additional complexity. More explicitly, this array only accounts for the vertices of the triangles. You hence need to think a way to consider also the midpoints of the edges. Present your choice and write a method called *stiffnessMatrixDeg2* that given the mesh variables *p*,*tri* obtained with the mesh generator, returns the global stiffness matrix A.

Task 1.1.4 Build the load vector for linear elements

Following a similar approach to the one used to assemble the stiffness matrix, write two methods

1. *localLoadVectorDeg1*,
2. *loadVectorDeg1*

where the first takes as input the vertices $\hat{p}_1, \hat{p}_2, \hat{p}_3$ of a generic triangle, and a function $f : T \to \mathbb{R}$, and returns the local load vector $F_T \in \mathbb{R}^3$ with entries

$$(F_T)_i = \int\int_T f(x,y)\hat{\varphi}_i(x,y)\mathrm{d}x\mathrm{d}y.$$

The method *loadVectorDeg1* instead should take as input the arrays *p*, *tri*, and the function $f : \Omega \to \mathbb{R}$, returning the global load vector F.

Task 1.1.5 Build the load vector for quadratic elements

Repeat the same procedure for quadratic elements. Call the methods

1. *localLoadVectorDeg2*,
2. *loadVectorDeg2*,

and be careful to adopt the same local-to-global mapping you've used to assemble the global stiffness matrix for quadratic elements.

Task 1.1.6 Apply the homogeneous Dirichlet boundary conditions

Write a method called *imposeDirichletBCs* that takes as inputs:

- the global stiffness matrix A,
- the global load vector F,
- the mesh arrays *p*, *tri*,
- the degree of the polynomials you are using, i.e. 1 or 2,

and returns A and F properly changed so that the homogeneous Dirichlet boundary conditions are imposed.

Hint: Change the method *getBoundaryNodes* written in the first task so it takes as input also the degree of the polynomials, and hence includes in the returned list not only the indices of the vertices at the boundary, but also of the midpoints in case the degree of the adopted polynomials is 2.

Task 1.1.7 Implement the calculation of the H^1-seminorm and L_2 norm

We recall that called $e(x,y) = u(x,y) - u_h(x,y)$ the error function, the energy norm for our problem corresponds to the H^1-seminorm and writes

$$|e|_{H^1(\Omega)} = \left(\int \int_\Omega \langle \nabla e(x,y), \nabla e(x,y) \rangle \mathrm{d}x\mathrm{d}y \right)^{1/2}, \tag{1}$$

while the L_2-norm writes

$$\|e\|_{L_2(\Omega)} = \left(\int \int_\Omega |e(x,y)|^2 \mathrm{d}x\mathrm{d}y \right)^{1/2}. \tag{2}$$

Write the methods *computeEnergyNorm* and *computeL2Norm* that take the following inputs:

- the array $\hat{u}_h$ representing the values of the approximate solution at the mesh nodes (i.e. the solution of the linear system $A\hat{u}_h = F$)
- the mesh arrays p, tri,
- the degree of the polynomials used to compute the approximate solution $\hat{u}_h$,
- the analytical solution u,

and returns the approximation of the energy norm (1) and of the L_2-norm (2) respectively.

Task 1.2 Testing the finite element solver and analysis

Task 1.2.1 Test problem over triangular mesh

Now, we have everything ready to solve a test problem. Consider the reference problem

$$\begin{cases} -\Delta u(x,y) = f(x,y), \ (x,y) \in \Omega \\ u(x,y) = 0, \ (x,y) \in \partial\Omega. \end{cases} \tag{3}$$

with $f(x,y)$ chosen such that

$$u_{a,b,\beta}(x,y) := x(1-x)y(1-y)e^{-\beta((x-a)^2+(y-b)^2)}$$

is the analytical solution of (3), where $a, b \in (0,1)$ and $\beta > 0$. Compute the right hand side and then proceed with the following experimentations. More explicitly, our goal is to see how the finite element solver behaves as we change the values of a, b, β, i.e. the shape of the analytical solution we are aiming to approximate.

We now want to compute the finite element approximation of the solution for meshes $\mathcal{T}_h$ having $n+1$ equispaced nodes in both directions, i.e. x and y, on the domain $[0,1]^2$, i.e. $(x_i, y_j) = (hi, hj)$ where $h = 1/n$.

For each of the ns in the set $\{2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8\}$ assemble the stiffness matrix A, and the load vector F taking piecewise linear basis functions, and also piecewise quadratic basis functions. Apply the homogeneous Dirichlet boundary conditions, and solve the linear systems $A\hat{u}_h = F$ with your favourite solver for linear systems. Be careful that the system can get quite big as n increases, so an informed choice on the solver needs to be made here.

Let $u_{a,b,\beta}(x,y)$ be the analytical solution to the problem. Compute the L_2 and the energy norm of the error $e_{a,b,\beta}(x,y) = u_{a,b,\beta}(x,y) - u_h(x,y)$, where

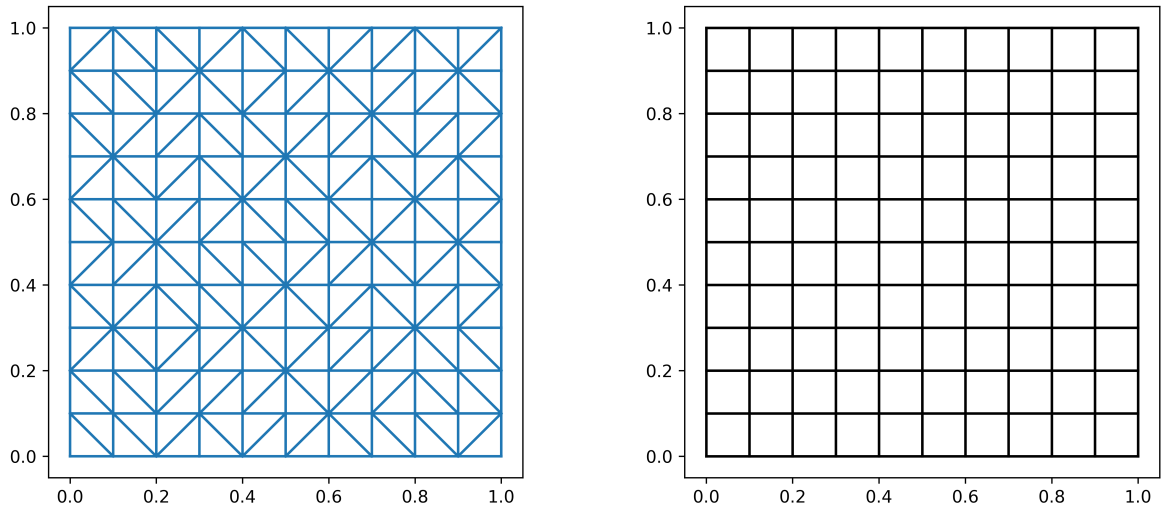
$$u_h(x,y) = \sum_{i=1}^{N} \hat{u}_i \hat{\varphi}_i(x,y)$$

for the proper N and basis functions. Plot the errors in a log-log scale and verify if the obtained results correspond to the expected convergence rates seen in class. **Repeat this experiment** for various combinations of a, b, β, especially including cases where β is large. Include the results in your report and comment on them.

Task 1.2.2 Comparison of the results of triangular mesh with quadrilateral one for linear elements

In this task you have to compare the results obtained for the triangular mesh with those you got in the previous project for the tensor product elements on a quadrilateral mesh. Since in the first project we only focused on 2d linear elements, we do this comparison only for linear elements. In Figure 1 you can see two meshes compared in the case where they both have 10 elements for each edge. Repeat the experiments done in subsection Task 1.2.1 for the linear tensor product elements studied in the first project.

Comment on the differences you see between the two families of finite elements both in terms of performance, accuracy, and in terms of difficulty of implementation. For this comparison, use again the reference problem (3) and repeat the tests for various combinations of a, b, β.



(a) Triangular mesh

(b) Quadrilateral mesh

Figure 1: Comparison of the two meshes with $n = 10$ elements per edge.

Option 2: Adaptive finite elements

In this task you will work with a posteriori indicators and use them to adapt meshes with an aim to make the error uniform. In this task you may use any (open source) software you like, but are recommended to

use Firedrake.

Task 2.1 Theory

Consider the 2D Helmholtz like equation

$$-\Delta u + cu = f \text{ in } \Omega$$
$$u = 0 \text{ on } \partial\Omega.$$

Find a finite element discretisation of this problem. Show a residual based a posteriori upper bound.

Task 2.2 Sharp and reliable

Fix c to be your favourite (non-zero) constant. Fix $f(x)$ such that you know the exact solution (you may also choose $f(x)$ to be a forcing term to enforce an exact solution). Use the Firedrake utility mesh `UnitSquareMesh(nx,ny)` (or similar) to construct a mesh with N elements on each edge of a unit square domain. Write a function which takes in N, a polynomial degree k which solves the finite element approximation with continuous Lagrange elements. The function should return the (global) a posteriori indicator and the error in the energy norm.

Vary N and investigate the convergence of the a posteriori indicator. Is it sharp and reliable? Does this depend on k?

Task 2.3 Mesh generation

Utilising `mesher.py`, which is available here, write a function which takes an unrefined mesh (with roughly N points along an edge) and refines about a collection of points. This function should take in a list of points to refine. Bonus points if you can generalise this to an iterative list (to perform multiple refinements). This function should write a mesh to file which can then read into Firedrake using `firedrake.Mesh(``mesh_name.msh'')`

Task 2.4 Localisation of indicator

Localise your indicator by projecting/interpolating it into the space of discontinuous constant functions over the mesh, which is written in Firedrake as

```
DG0 = firedrake.FunctionSpace(mesh, ``Discontinuous Lagrange'',0).
```

Build a function which finds the location of the elements which are greater than some tolerance γ.

Hint: Here's a hack to find the locations of points:

- Project the indicator into `DG0`, obtaining a Firedrake function `eta`.
- Convert this function to a vector through `eta.dat.data`.
- Interpolate the function x and the function y into `DG0` as follows.

```
x,y = firedrake.SpatialCoordinate(mesh)
f_x = firedrake.Function(DG0).interpolate(x)
f_y = firedrake.Function(DG0).interpolate(y)
```

- Convert `f_x` and `f_y` to vectors to find the points in x and y associated to a location in a `DG0` vector.

Task 2.5 Adaptive refinement over the unit square

Choose $f(x,y) = \sin(16\pi x)\sin(2\pi y)$. Perform (at least one step of) mesh refinement through

- Solving over a uniform mesh.
- Computing the local indicator and finding the locations where it is greater than some tolerance γ.

- Remeshing your problem (with gmsh).
- Re-solving over the new mesh.

Do the same with

$$f(x,y) = \begin{cases} 10 \text{ for } x,y \in [0.45, 0.55] \\ 1 \text{ for } x,y \in [0.4, 0.45) \cup (0.55, 0.6] \\ 0 \text{ else.} \end{cases}$$

Task 2.6 L-shaped domain

Here we generalise our set up to mixed boundary conditions. Consider the Helmholtz equation

$$\begin{aligned} -\Delta u + cu &= 0 \text{ in } \Omega \\ u &= 0 \text{ on } \Gamma_D \\ \frac{\partial u}{\partial n} &= g \text{ on } \Gamma_N, \end{aligned}$$

where g is defined such that the boundary conditions are consistent with

$$u(x,y)\big|_{\Gamma_N} = \sin\left(2\pi\sqrt{(x-1)^2 + (y-1)^2}\right).$$

Let our domain Ω be L-shaped, as described in Figure 2. We note that at $(1,1)$ the Neumann condition is

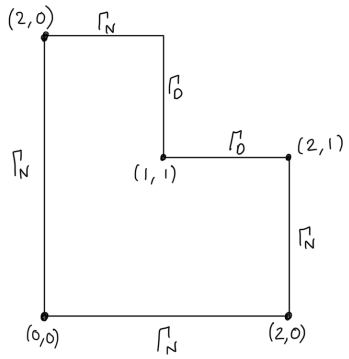


Figure 2: L-shaped domain

consistent with the Dirichlet boundary conditions.

Repeat the adaptive refinement procedure given in Task 2.5 for $k = 1$ for this domain. What is the ideal mesh? Why is the case?

Option 3: Biharmonic problem

Disclaimer: This project involves a small amount of content which has not been covered in the lectures (namely preconditioning linear systems). No theoretical results are needed on preconditioning.

In this task you will work with the biharmonic problem (a fourth order elliptic PDE), investigate conforming and nonconforming approximations, and study the condition number of the underlying linear system.

Task 3.1 Theory

Consider the fourth order scalar PDE

$$\nabla^4 u := \nabla^2 \nabla^2 u = \nabla \cdot \nabla (\nabla \cdot \nabla u) = f$$

where $u \in H^4(\Omega)$.

1. Show that the weak formulation is described by

$$\left\langle \nabla^2 u, \nabla^2 v \right\rangle + \int_{\partial\Omega} \left(\nabla \left(\nabla^2 u \right) \cdot \boldsymbol{n} \right) v \, \mathrm{d}S - \int_{\partial\Omega} \left(\nabla^2 u \right) (\nabla v \cdot \boldsymbol{n}) \, \mathrm{d}S = \langle f, v \rangle \qquad \forall v \in H^2(\Omega),$$

   where $\boldsymbol{n}$ is the outward unit normal vector to the boundary. How much regularity is needed on u?

2. Assuming purely Neumann boundary conditions, i.e.,

$$\nabla^2 u = f \text{ on } \partial\Omega$$
$$\nabla \left(\nabla^2 u \right) \cdot \boldsymbol{n} = g \text{ on } \partial\Omega,$$

   is this problem well posed? Hint: Find a non-trivial nullspace.

3. For the remainder of this project consider the stabilised weak form

$$\left\langle \nabla^2 u, \nabla^2 v \right\rangle + \langle \nabla u, \nabla v \rangle + \langle u, v \rangle + \int_{\partial\Omega} \left(\nabla \left(\nabla^2 u \right) \cdot \boldsymbol{n} - \nabla u \cdot \boldsymbol{n} \right) v \, \mathrm{d}S - \int_{\partial\Omega} \left(\nabla^2 u \right) (\nabla v \cdot \boldsymbol{n}) \, \mathrm{d}S = \langle f, v \rangle \qquad \forall v \in H^2(\Omega). \tag{4}$$

   What is the PDE we are now solving? What are the Dirichlet and Neumann boundary conditions of this modified problem. Prove this form, with zero Neumann boundary conditions, is stable if $f(x) \in L_2(\Omega)$. In the sequel you may fix pure Neumann boundary conditions for this modified problem.

Task 3.2 H^2-conforming approximation with Argyris elements

Discretise (4) with finite elements and implement the method over a unit square domain with non-zero boundary conditions. Choose boundary conditions and $f(x)$ such that you know the exact solution ($f(x)$ may be used as a forcing term). Use the Firedrake utility mesh `UnitSquareMesh(nx,ny)` (or similar) to construct a mesh with N elements on each edge of a unit square domain. Write a function which takes in N, computes the finite element approximation using 5-th order Argyris elements (as introduced in the lectures) over a triangular mesh. The function should output the error in the L_2, H^1 and H^2 norms. Vary N and investigate the convergence of this method.

Task 3.3 Non-conforming approximation with continuous Lagrange elements

Here we consider H^1 methods and modify the bilinear form $a(u,v) := \left\langle \nabla^2 u, \nabla^2 v \right\rangle$ in the spirit of a discontinuous Galerkin approximation. That is to say, in the discrete setting we replace the bilinear form with $a_h : H^1 \times H^1 \to \mathbb{R}$ such that

$$a_h(u_h, v_h) := \sum_{T \in \mathcal{T}_h} \left\langle \nabla^2 u, \nabla^2 v \right\rangle_T + \sum_{e \in \mathcal{E}_h} \frac{\sigma}{|e|} \int_e [\![\nabla u \cdot \boldsymbol{n}]\!] [\![\nabla v \cdot \boldsymbol{n}]\!] \, \mathrm{d}S,$$

where $[\![\cdot]\!]$ represents the jump of the solution over the edge e. We note that the derivative of a H^1 function is discontinuous. This choice is informed by [2]. The jump term is known as the penalty term as it penalises the discontinity of the derivative and brings the nonconforming solution closer to being in H^2. Note that σ is a user specified parameter, and must be sufficiently large for existence/uniqueness. For ease of implementation, the following Firedrake functions are recommended

- Element diameter: `h = CellSize(mesh)` (given a Firedrake mesh)
- Normal vector: `n = FacetNormal(mesh)` (given a Firedrake mesh)
- Penalty term: `sigma/h * jump(grad(u),n) * jump(grad(v),n) * dS` (given a Firedrake function `u` and test function `v`).

We further define the broken H^2 norm such that for $w \in H^1(\Omega) \cap H^2(T)$ for all $T \in \mathcal{T}_h$ as

$$\|w\|_h := a_h(w,w) + \|w\|_{H^1(\Omega)}.$$

This is sometimes referred to as a mesh dependent norm due to the mesh dependency of the nonconforming bilinear form a_h. When applied to $w \in H^2(\Omega)$ it is equivalent to the H^2 norm.

What is the minimal degree of Lagrange elements required for this discretisation? Build a function similar to that built in Task 3.2 using continuous Lagrange elements. Investigate the dependency of the solution on σ and how this effects smoothness in practice (for example, through plotting the derivative of the solution). Investigate the convergence of this approximation, and compare it against the conforming approximation with Argyris elements.

Warning: If you choose σ to be too small the method may become unstable. The lowest stable value of this parameter depends on the polynomial degree k.

Task 3.4 Preconditioning

Investigate the condition number of the underlying matrix of the nonconforming finite element discretisation. Defining your form as F, this matrix can be accessed as a sparse matrix through

```
A = firedrake.assemble(firedrake.lhs(F),mat_type='aij').M.handle.getValuesCSR()
A = scipy.sparse.csr_matrix((A[2],A[1],A[0]))
```

How does the condition number scale with N and k? Assemble a finite element discretisation of the Poisson equation and use the resulting matrix to precondition your problem. How does this change things? You *do not* need to solve the preconditioned system.

References

[1] Mark Ainsworth and J Tinsley Oden. A posteriori error estimation in finite element analysis. *Computer methods in applied mechanics and engineering*, 142(1-2):1–88, 1997.

[2] Susanne C. Brenner and Li-Yeng Sung. C^0 interior penalty methods for fourth order elliptic boundary value problems on polygonal domains. *J. Sci. Comput.*, 22/23:83–118, 2005.

[3] Patrick E Farrell, Abdalaziz Hamdan, and Scott P MacLachlan. A new mixed finite-element method for H^2 elliptic problems. *arXiv preprint arXiv:2105.07289*, 2021.

[4] David A Ham, Paul HJ Kelly, Lawrence Mitchell, C Cotter, Robert C Kirby, Koki Sagiyama, Nacime Bouziani, Sophia Vorderwuelbecke, T Gregory, Jack Betteridge, et al. Firedrake user manual. 2023.