Parameter-Robust Discretisations of Anisotropic Diffusion Problems

Mark Pearson

Linacre College

University of Oxford

A thesis submitted for the degree of

M.Sc. in Mathematical Modelling and Scientific Computing

Trinity Term 2022

Acknowledgements

Firstly, I would like to thank my supervisor, Prof Patrick Farrell. He provided me with guidance and direction for this project. Furthermore, he contributed his unpublished notes on the problem.

Secondly, thanks are given to Dr Kathryn Gillow for providing a well-organised course and constant support.

Finally, I thank Culham Centre for Fusion Energy for funding this project.

Abstract

In this thesis, we wish to solve highly anisotropic diffusion problems. We discuss existing methods, propose new methods and demonstrate how to implement these methods in Python using Firedrake. Then we test and compare the different methods on a range of examples. Additionally, we show this problem can benefit from a parallel implementation and we provide the code on GitHub at [1].

Contents

Chapter 1

Introduction and Current Methods

1.1 Introduction

In this paper, we aim to find robust methods to solve highly anisotropic diffusion problems on a bounded Lipschitz domain Ω. Thus we want to solve

$$\begin{cases} -\nabla \cdot (\mathbb{A}\nabla u) = f, & \text{in } \Omega, \\ \mathbf{n} \cdot \mathbb{A}\nabla u = 0, & \text{on } \Gamma_N, \\ u = 0, & \text{on } \Gamma_D, \end{cases} \tag{1.1}$$

where $\Gamma_N \cup \Gamma_D := \partial\Omega$ and

$$\mathbb{A} = \epsilon^{-1} A_{||} \mathbf{b} \otimes \mathbf{b} + (I - \mathbf{b} \otimes \mathbf{b}) \mathbb{A}_{\perp} (I - \mathbf{b} \otimes \mathbf{b}). \tag{1.2}$$

Here $\epsilon \ll 1$ is the anisotropic coefficient, $\mathbf{b}$ represents the electromagnetic vector field. From [2] we have restrictions on $\mathbf{b}$, they demand that

1. $\mathbf{b} \in \mathcal{C}^{\infty}(\Omega; \mathrm{R}^d)$;
2. $|\mathbf{b}| = 1$ for all $\mathbf{x} \in \Omega$;
3. $\mathbf{b} \cdot \mathbf{n} = 0$ on Γ_D the Dirichlet boundary;
4. $\mathbf{b} \cdot \mathbf{n} \neq 0$ on Γ_N the Neumann boundary;

where $\mathbf{n}$ is the outward normal. Additionally, we have $A_{||}$ as a positive scalar field and $\mathbb{A}$ is a symmetric positive definite matrix.

The main difficulty with solving this problem is that it is singular for $\varepsilon = 0$. In particular, the natural formulation becomes highly ill-conditioned for values of ε

required in fusion applications, where $10^{-8} < \varepsilon < 10^{-6}$. We seek discretisations that work in this range.

From multiple papers, the best notation for this problem is stated in [2] and [3]. Thus we will use a similar notation. For a scalar field $u : \Omega \to \mathbb{R}$ and vector field $v : \Omega \to \mathbb{R}^d$, we use the notation

$$v_{||} = (\mathbf{b} \otimes \mathbf{b})v = (v \cdot \mathbf{b})\mathbf{b}, \qquad v_\perp = (\mathbb{I} - \mathbf{b} \otimes \mathbf{b})v, \tag{1.3}$$

$$\nabla_{||} u = (\mathbf{b} \otimes \mathbf{b})\nabla u = (\nabla u \cdot \mathbf{b})\mathbf{b}, \qquad \nabla_\perp u = (\mathbb{I} - \mathbf{b} \otimes \mathbf{b})\nabla u, \tag{1.4}$$

$$\nabla_{||} \cdot v = \nabla \cdot v_{||}, \qquad \nabla_\perp \cdot v = \nabla \cdot v_\perp. \tag{1.5}$$

Now we define

$$\Delta_{||} u = \nabla_{||} \cdot (A_{||} \nabla_{||} u) = \nabla \cdot (A_{||}(\mathbf{b} \otimes \mathbf{b})\nabla u), \tag{1.6}$$

$$\Delta_\perp u = \nabla_\perp \cdot (\mathbb{A}_\perp \nabla_\perp u) = \nabla \cdot ((\mathbb{I} - \mathbf{b} \otimes \mathbf{b})\mathbb{A}_\perp(\mathbb{I} - \mathbf{b} \otimes \mathbf{b})\nabla u). \tag{1.7}$$

Thus from (1.6) and (1.7) it is clear to see (1.2) can be written as

$$\mathbb{A}\nabla u = \varepsilon^{-1}(A_{||}\nabla_{||} u)_{||} + (\mathbb{A}_\perp \nabla_\perp u)_\perp. \tag{1.8}$$

Thus putting (1.8) into the PDE (1.1) we get

$$\begin{cases} -\varepsilon^{-1}\Delta_{||} u - \Delta_\perp u = f, & \text{in } \Omega, \\ \mathbf{n} \cdot \mathbb{A}\nabla u = 0, & \text{on } \Gamma_N, \\ u = 0, & \text{on } \Gamma_D. \end{cases} \tag{1.9}$$

We now write the PDE (1.1) in primal weak form

$$\int_\Omega f\hat{u} d\mathbf{x} = -\int_\Omega \nabla \cdot (\mathbb{A}\nabla u)\hat{u}\mathbf{x}, \tag{1.10}$$

$$= -\int_{\partial\Omega} \hat{u}(\mathbb{A}\nabla u) \cdot \mathbf{n} d\mathbf{x} + \int_\Omega (\mathbb{A}\nabla u) \cdot \nabla\hat{u} d\mathbf{x}, \tag{1.11}$$

$$= -\int_{\Gamma_D} \hat{u}(\mathbb{A}\nabla u) \cdot \mathbf{n} + \varepsilon^{-1}\int_\Omega A_{||}\nabla_{||} u \cdot \nabla_{||}\hat{u} d\mathbf{x} + \int_\Omega (\mathbb{A}_\perp \nabla_\perp u) \cdot \nabla_\perp \hat{u} d\mathbf{x}, \tag{1.12}$$

where u is a trial function and $\hat{u}$ is a test function. To simplify notation we use

$$a_{||}(\alpha, \beta) = \int_\Omega A_{||}\nabla_{||}\alpha \cdot \nabla_{||}\beta d\mathbf{x}, \tag{1.13}$$

$$a_\perp(\alpha, \beta) = \int_\Omega (\mathbb{A}_\perp \nabla_\perp \alpha) \cdot \nabla_\perp \beta d\mathbf{x}. \tag{1.14}$$

For our numerical calculations, we use Firedrake [4] which written in code is

```
grad_para = lambda alpha: dot(outer(b,b), grad(alpha))
grad_perp = lambda alpha: dot(I-outer(b,b), grad(alpha))
a_perp = lambda alpha, beta: inner(A_perp*grad_perp(alpha),
                                   grad_perp(beta))
a_para = lambda alpha, beta: inner(A_para*grad_para(alpha),
                                   grad_para(beta))
```

Also, we have

$$\varepsilon^{-1}(A_{||}\nabla_{||}u)_{||}\cdot\mathbf{n}=\varepsilon^{-1}(A_{||}\nabla_{||}u)\cdot\mathbf{n}=\varepsilon^{-1}A_{||}(\nabla u\cdot\mathbf{b})\mathbf{b}\cdot\mathbf{n}=0, \tag{1.15}$$

on Γ_D as restriction 3 of the electromagnetic field $\mathbf{b}$ states $\mathbf{b}\cdot\mathbf{n}=0$. This means from (1.15) and using notation (1.13) and (1.14) we get (1.12) becomes

$$\int_\Omega f\hat{u}d\mathbf{x}=-\int_{\Gamma_D}\hat{u}(\mathbb{A}_\perp\nabla_\perp u)_\perp\cdot\mathbf{n}d\mathbf{x}+\varepsilon^{-1}a_{||}(u,\hat{u})+a_\perp(u,\hat{u}). \tag{1.16}$$

In the integral over Γ_D in (1.16) u can be replaced with 0 because in the boundary conditions for (1.1) $u,\hat{u}\in\mathcal{V}$. This is included in the formula to show how one would calculate the problem with non-zero boundary conditions. When there are non-zero Dirichlet boundary conditions we would use the weak form (1.11). Our methods require the use of the sets

$$\Gamma_{in}:=\{\mathbf{b}\in\Gamma:\mathbf{b}\cdot\mathbf{n}<0\}, \tag{1.17}$$

$$\mathcal{V}:=\{v\in\mathcal{H}^1(\Omega):v|_{\Gamma_D}=0\}, \tag{1.18}$$

$$\mathcal{L}:=\{\lambda\in\mathcal{H}^1(\Omega):\lambda|_{\Gamma_{in}\cup\Gamma_D}=0\}, \tag{1.19}$$

where $\mathcal{H}^k(\Omega)$ the Sobolev space of order k. Also, it should be noted that sets $\mathcal{V}$ and $\mathcal{L}$ are easy to discretise. However, when the magnetic field $\mathbf{b}$ contains closed field lines it is not clear how we should adapt $\mathcal{L}$. One approach found by our numerical simulations shows we can still get a reasonable solution when we include a restriction to zero on a point for every closed streamline. Thus $\Gamma_{CL}:=\{\text{one point on every closed streamline}\}$ and we can redefine $\mathcal{L}:=\{\lambda\in\mathcal{H}^1(\Omega):\lambda|_{\Gamma_{CL}\cup\Gamma_{in}\cup\Gamma_D}=0\}$.

1.2 Overview of Current Methods

We now state some methods for solving the PDE (1.1). For a few methods, we show the key parts of their Python implementation with Firedrake [4]. The full implementation of the methods can be found in Appendix C.

1.2.1 Singular Perturbation Method

The singular perturbation method also known as the primal formulation involves solving the weak form of equation (1.1). Therefore we have a trial function $u \in \mathcal{V}$ and a test function $\psi \in \mathcal{V}$. Thus we solve the weak form (1.20) for $(u) \in \mathcal{V}$.

$$(SP) \left\{ \varepsilon^{-1} a_{||}(u, \psi) + a_{\perp}(u, \psi) = \int_{\Omega} f \psi d\mathbf{x}, \forall \psi \in \mathcal{V}. \right. \tag{1.20}$$

Below we have the key points of our Firedrake [4] implementation

```
V = FunctionSpace(mesh, "CG", Order)
...
bcs = [DirichletBC(V, dOmegaDVal, dOmegaD)]
...
u = TrialFunction(V)
psi = TestFunction(V)
u_h = Function(V)
a = ((1/eps)*a_para(u,psi) + a_perp(u, psi))*dx
F = inner(f, psi)*dx
solve(a==F, u_h, bcs)
```

This method is the natural choice when $\varepsilon \approx 1$.

1.2.2 Limit Method

We now introduce a limit method discussed in paper [3]. The limit method involves solving the equation (1.1) when $\varepsilon \to 0$. Therefore, $\Delta_{||} u = 0$ so since we have zero Dirichlet boundary conditions the solution u satisfies $\nabla_{||} u = 0$ thus it is constant along the streamlines of the vector field. We apply this restriction using a Lagrangian parameter. This leads to solving the weak form (1.21) for the trial functions $(u, q) \in \mathcal{V} \times \mathcal{L}$.

$$(LM) \begin{cases} a_{\perp}(u, \hat{u}) + a_{||}(q, \hat{u}) = \int_{\Omega} f \hat{u} d\mathbf{x}, & \forall \hat{u} \in \mathcal{V}, \\ a_{||}(u, \hat{q}) = 0, & \forall \hat{q} \in \mathcal{L}, \end{cases} \tag{1.21}$$

where $(\hat{u}, \hat{q}) \in \mathcal{V} \times \mathcal{L}$ are test functions. This can be written in Firedrake with the following code

```
V = FunctionSpace(mesh, "CG", Order)
L = FunctionSpace(mesh, "CG", Order)
Z = V*L
...
bc0 = DirichletBC(Z.sub(0), 0, dOmegaD)
bc1 = DirichletBC(Z.sub(1), 0, dOmegaD)
```

```
bc2 = DirichletBC(Z.sub(1), 0, dOmegaIN)
bcs = [bc0, bc1, bc2]
...
z = Function(Z)
u, q = split(z) #Trial Function
u_, q_ = split(TestFunction(Z))
F = (+ a_perp(u, u_)*dx + a_para(q, u_)*dx - inner(f, u_)*dx
     + a_para(u, q_)*dx )
solve(F==0,z,bcs)
(u_h, q_h) = z.split()
```

1.2.3 MMAP Method

We now introduce a Micro-Macro Asymptotic-Preserving method discussed in paper [5]. This method is similar to the (LM) method from section 1.2.2 but from the additional ε term it can deal with $\varepsilon \approx 1$. This leads to solving the weak form (1.22) for the trial functions $(u, q) \in \mathcal{V} \times \mathcal{L}$.

$$(MMAP) \begin{cases} a_\perp(u, \hat{u}) + a_{||}(q, \hat{u}) = \int_\Omega f\hat{u}d\mathbf{x}, & \forall \hat{u} \in \mathcal{V}, \\ a_{||}(u, \hat{q}) - \varepsilon a_{||}(q, \hat{q}) = 0, & \forall \hat{q} \in \mathcal{L}, \end{cases} \tag{1.22}$$

where $(\hat{u}, \hat{q}) \in \mathcal{V} \times \mathcal{L}$ are test functions. The code is the same as the (LM) method in section 1.2.2 but F is defined as

```
F = ( + a_perp(u, u_)*dx + a_para(q, u_)*dx - inner(f, u_)*dx
      + a_para(u, q_)*dx - eps*a_para(q, q_)*dx)
```

1.2.4 AP Method

We now discuss an Asymptotic-Preserving method stated in paper [3]. This method involves solving the weak form (1.23) for trial functions $(p, \lambda, q, \ell, \mu) \in \mathcal{V} \times \mathcal{L} \times \mathcal{V} \times \mathcal{V} \times \mathcal{L}$.

$$(AP) \begin{cases} a_\perp(p, \eta) + a_\perp(q, \eta) + a_{||}(\eta, \lambda) = \int_\Omega f\eta d\mathbf{x}, & \forall \eta \in \mathcal{V}, \\ a_{||}(p, \kappa) = 0, & \forall \kappa \in \mathcal{L}, \\ a_{||}(q, \xi) + \varepsilon a_\perp(p + q, \xi) = \int_\Omega (\varepsilon f - \ell)\xi d\mathbf{x}, & \forall \xi \in \mathcal{V}, \\ a_{||}(\chi, \mu) + \int_\Omega q\chi d\mathbf{x} = 0, & \forall \chi \in \mathcal{V}, \\ a_{||}(\ell, \tau) = 0, & \forall \tau \in \mathcal{L}, \end{cases} \tag{1.23}$$

where $(\eta, \kappa, \xi, \chi, \tau) \in \mathcal{V} \times \mathcal{L} \times \mathcal{V} \times \mathcal{V} \times \mathcal{L}$ are test functions. We have $u = p + q$.

1.2.5 Limit Stabilisation Method

The (LM) method stated in section 1.2.2 struggles when there is a magnetic field $\mathbf{b}$ with closed lines. Thus we introduce a stabilisation method which is discussed in [6]. We implement this modification to get the weak form

$$(LM\_STAB)\begin{cases} a_{\perp}(u,\hat{u}) + a_{||}(q,\hat{u}) = \int_{\Omega} f\hat{u}d\mathbf{x}, & \forall \hat{u} \in \mathcal{V}, \\ a_{||}(u,\hat{q}) = \sigma \int_{\Omega} q\hat{q}d\mathbf{x}, & \forall \hat{q} \in \mathcal{V}. \end{cases} \tag{1.24}$$

We solve for trial functions $(u,q) \in \mathcal{V}\times\mathcal{V}$ with test functions $(\hat{u},\hat{q}) \in \mathcal{V}\times\mathcal{V}$. Additionally, we do not use $\mathcal{L}$ therefore there are no complications when the electromagnetic field $\mathbf{b}$ has closed field lines or field lines that converge to a point like $\dot{x} = -x/|x|$ with $x = 0$. We can implement this in Firedrake [4] with the following code

```
V = FunctionSpace(mesh, "CG", Order)
Z = V*V
...
bc0 = DirichletBC(Z.sub(0), 0, dOmegaD)
bc1 = DirichletBC(Z.sub(1), 0, dOmegaD)
bcs = [bc0, bc1]
...
z = Function(Z)
u, q = split(z) #Trial Function
u_, q_ = split(TestFunction(Z))
F = (+ a_perp(u, u_)*dx+a_para(q, u_)*dx-inner(f, u_)*dx
     + a_para(u, q_)*dx-sigma*inner(q,q_)*dx)
solve(F==0,z,bcs)
(u_h, q_h) = z.split()
```

1.2.6 MMAP Stabilisation Method

We also apply the stabilisation method mentioned in [6] to the $(MMAP)$ method stated in section 1.2.3. Thus we get a weak form

$$(MMAP\_STAB)\begin{cases} a_{\perp}(u,\hat{u}) + a_{||}(q,\hat{u}) = \int_{\Omega} f\hat{u}d\mathbf{x}, & \forall \hat{u} \in \mathcal{V}, \\ a_{||}(u,\hat{q}) - \varepsilon a_{||}(q,\hat{q}) = \sigma \int_{\Omega} q\hat{q}d\mathbf{x}, & \forall \hat{q} \in \mathcal{V}, \end{cases} \tag{1.25}$$

where we solve for trial functions $(u,q) \in \mathcal{V}\times\mathcal{V}$ with test functions $(\hat{u},\hat{q}) \in \mathcal{V}\times\mathcal{V}$. The code is similar to the (LM_STAB) method in section 1.2.5 but F is defined as

```
F = (+ a_perp(u, u_)*dx+a_para(q, u_)*dx-inner(f, u_)*dx
     + a_para(u, q_)*dx-eps*a_para(q, q_)*dx-sigma*inner(q,q_)*dx)
```

1.2.7 Deluzet-Narski Method

This method turns the strongly anisotropic equation into two mildly anisotropic equations to be solved by iteration. This method is stated in [2]. This introduces $\varepsilon_0 \gg \varepsilon$ and gives the weak form (1.26) which solves for the sequence of trial functions $(u^{(n)}, q^{(n)}) \in \mathcal{V} \times \mathcal{V}$.

$$(DN)\begin{cases} a_{||}(u^{(n+1)}, \hat{u}) + \varepsilon_0 a_{\perp}(u^{(n+1)}, \hat{u}) = \varepsilon_0 \int_\Omega f\hat{u}d\mathbf{x} + (\varepsilon - \varepsilon_0)a_{||}(q^{(n)}, \hat{u}), & \forall \hat{u} \in \mathcal{V}, \\ a_{||}(q^{(n+1)}, \hat{q}) + \varepsilon_0 a_{\perp}(q^{(n+1)}, \hat{q}) = \int_\Omega f\hat{q}d\mathbf{x} - a_{\perp}(u^{(n+1)} - \varepsilon_0 q^{(n)}, \hat{q}), & \forall \hat{q} \in \mathcal{V}. \end{cases} \tag{1.26}$$

With test functions $(\hat{u}, \hat{q}) \in \mathcal{V} \times \mathcal{V}$. The Python implementation is in appendix C.

1.3 Proposed Methods

We introduce a new method based on unpublished notes from Prof Patrick Farrell. This involves introducing the auxiliary variable $q = \varepsilon^{-1}\mathbf{b} \cdot \nabla u$, after some algebraic manipulation we get the following equations which are equivalent

$$q = \varepsilon^{-1}\mathbf{b} \cdot \nabla u, \tag{1.27}$$

$$q = \varepsilon^{-1}\mathbf{b} \cdot \nabla_{||} u, \tag{1.28}$$

$$\mathbf{b}q = \varepsilon^{-1}\nabla_{||} u. \tag{1.29}$$

Also, we introduce new sets

$$\mathcal{Q}_{in} := \{q \in \mathcal{H}^1(\Omega) : q|_{\Gamma_{in}} = 0\}, \tag{1.30}$$

$$\mathcal{Q} := \{q \in \mathcal{H}^1(\Omega)\}. \tag{1.31}$$

1.3.1 PF Method

Therefore, by doing the substitution $q = \varepsilon^{-1}\mathbf{b} \cdot \nabla u$ into (1.1) we get the weak form

$$(PF)\begin{cases} \int_\Omega A_{||} q\mathbf{b} \cdot \nabla \hat{u} d\mathbf{x} + a_{\perp}(u, \hat{u}) = \int_\Omega f\hat{u}d\mathbf{x}, & \forall \hat{u} \in \mathcal{V}, \\ \int_\Omega \mathbf{b} \cdot \nabla \hat{u}\hat{q}d\mathbf{x} = \varepsilon \int_\Omega q\hat{q}d\mathbf{x}, & \forall \hat{q} \in \mathcal{Q}_{in}. \end{cases} \tag{1.32}$$

We solve for trial functions $(u, q) \in \mathcal{V} \times \mathcal{Q}_{in}$ with test functions $(\hat{u}, \hat{q}) \in \mathcal{V} \times \mathcal{Q}_{in}$. The key parts of our Firedrake [4] implementation are

```
V = FunctionSpace(mesh, "CG", Order)
Q = FunctionSpace(mesh, "CG", Order)
```

```
Z = V*Q
...
bc0 = DirichletBC(Z.sub(0), 0, dOmegaD)
bc2 = DirichletBC(Z.sub(1), 0, dOmegaIN)
bcs = [bc0, bc2]
...
z = Function(Z)
u, q = split(z)
v, w = split(TestFunction(Z))
F = (+inner(grad_perp(u), grad_perp(v))*dx+inner(q*b, grad(v))*dx
     - inner(f, v)*dx #1
     - eps * inner(q, w)*dx + inner(grad(u), w*b)*dx) #2
solve(F == 0, z, bcs)
(u_h, q_h) = z.split()
```

1.3.2 PF Stabilisation Method

We can use the stabilisation technique stated in [6] to get the weak form

$$(PF\_STAB)\begin{cases}\int_\Omega A_{\|} q\mathbf{b}\cdot\nabla\hat{u}d\mathbf{x}+a_\perp(u,\hat{u})=\int\Omega f\hat{u}d\mathbf{x}, & \forall\hat{u}\in\mathcal{V},\\ \int_\Omega \mathbf{b}\cdot\nabla\hat{u}\hat{q}d\mathbf{x}=(\varepsilon+\sigma)\int_\Omega q\hat{q}d\mathbf{x}, & \forall\hat{q}\in\mathcal{Q}.\end{cases} \tag{1.33}$$

We solve for trial functions $(u,q)\in\mathcal{V}\times\mathcal{Q}$ with test functions $(\hat{u},\hat{q})\in\mathcal{V}\times\mathcal{Q}$. Thus in our code, we change F and the boundary condition constraints to

```
bcs = [bc0]
...
F = (+inner(grad_perp(u), grad_perp(v))*dx+inner(q*b, grad(v))*dx
     - inner(f, v)*dx #1
     -eps*inner(q, w)*dx-sigma*inner(q, w)*dx
     + inner(grad(u), w*b)*dx) #2
```

Chapter 2

Numerical Demonstrations

In these examples, we calculate f from inserting an exact solution u through the PDE. Additionally, in these examples we take $A_{||} = 1$ and $\mathbb{A}_{\perp} = \mathbb{I}$, where $\mathbb{I}$ is the identity matrix. Also, for the stabilisation methods we take $\sigma = 0.1$. But first, we briefly discuss the basic idea of how the finite element solver Firedrake [4] which is based on the package FEniCSx [7] calculates the solution. Also, we visualise data in ParaView [8], Ipyvolume [9] and Matplotlib [10].

Moreover, in section 3.1.4 we discuss how our solution can be improved using heterogeneous computing techniques.

2.1 Mapping to Reference Element

For each finite element, we would like to create a linear transformation to a reference element. Thus we only need to calculate the basis functions for one element. We will define a linear transformation $\mathcal{M}$ which takes the reference element RE to a finite element FE. Where $\mathcal{M}^{-1}$ denotes the inverse.

$$\mathcal{M} : RE \to FE, \tag{2.1}$$

$$\mathcal{M}^{-1} : FE \to RE. \tag{2.2}$$

We denote $\phi_i^{(RE)}$ as the basis functions on the reference element and $\phi_i^{(FE)}$ as the basis functions on a finite element. We have the following relationship

$$\phi_i^{(FE)}(x) = \phi_i^{(RE)}(\xi), \tag{2.3}$$

$$\phi_i^{(FE)}(x) = \phi_i^{(RE)}(\mathcal{M}^{-1}(x)), \tag{2.4}$$

$$\phi_i^{(FE)}(\mathcal{M}(\xi)) = \phi_i^{(RE)}(\xi). \tag{2.5}$$

where $\mathbf{x} \in FE$ and $\xi \in RE$. Also, the i means the same basis function location in other words $\mathcal{M}(\ell_i^{(RE)}) = \ell_i^{(FE)}$. Thus we will use the equation (2.6) to calculate the integral in the RE region.

$$\int_{\mathcal{M}(RE)=FE} g(\mathbf{x})d\mathbf{x} = \int_{RE} g(\mathcal{M}(\xi)) \cdot |\det(D\mathcal{M}(\xi))| d\mathbf{x}, \tag{2.6}$$

where g is an arbitrary function.

2.1.1 Interval (1D)

For an interval finite element we have $0 \leq \xi \leq 1$ and $L \leq x \leq U$. Thus we get the transformation

$$\mathcal{M}(\xi) = L + \xi(U - L), \tag{2.7}$$

$$\mathcal{M}^{-1}(x) = \frac{x - L}{U - L}. \tag{2.8}$$

Using (2.6) we now find an integral that is simpler to calculate for a function f.

$$\int_{FE} f(x)\phi_i^{(FE)}(x)dx = \int_{RE} f(\mathcal{M}(\xi))\phi_i^{(FE)}(\mathcal{M}(\xi)) \cdot |\det(D\mathcal{M}(\xi))| d\xi, \tag{2.9}$$

$$= (U - L) \int_{RE} f(\mathcal{M}(\xi))\phi_i^{(RE)}(\xi)d\xi. \tag{2.10}$$

Furthermore, we can use

$$\partial_x \phi_i^{(FE)}(x) = \partial_\xi \phi_i^{(FE)}(\mathcal{M}(\xi))\xi_x = \partial_\xi \phi_i^{(PE)}(\xi)/(U - L), \tag{2.11}$$

when derivatives of test functions are involved.

2.1.2 Triangle (2D)

For a triangle finite element we have $0 \leq \xi_0, \xi_1 \leq 1$ and the constraint $\xi_0 + \xi_1 \leq 1$. Where ξ_0 represents the $(1, 0)$ direction and ξ_1 represents the $(0, 1)$ direction on the reference element RE. Additionally, we have positions on the finite element FE

$$\mathcal{M}(0, 0) = \ell_0^{(FE)} = (x_0, y_0), \tag{2.12}$$

$$\mathcal{M}(1, 0) = \ell_1^{(FE)} = (x_1, y_1), \tag{2.13}$$

$$\mathcal{M}(0, 1) = \ell_2^{(FE)} = (x_2, y_2), \tag{2.14}$$

where the linear transformation $\mathcal{M}(\xi_0, \xi_1)$ is defined as

$$\mathcal{M}(\xi_0, \xi_1) = \ell_0^{(FE)} + \xi_0(\ell_1^{(FE)} - \ell_0^{(FE)}) + \xi_1(\ell_2^{(FE)} - \ell_0^{(FE)}), \tag{2.15}$$

$$x = x_0 + \xi_0(x_1 - x_0) + \xi_1(x_2 - x_0), \tag{2.16}$$

$$y = y_0 + \xi_0(y_1 - y_0) + \xi_1(y_2 - y_0). \tag{2.17}$$

And the inverse linear transformation $\mathcal{M}^{-1}(x, y)$ defined as

$$\begin{bmatrix} \xi_0 \\ \xi_1 \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} (y_2 - y_0)(x - x_0) + (x_0 - x_2)(y - y_0) \\ (y_0 - y_1)(x - x_0) + (x_1 - x_0)(y - y_0) \end{bmatrix}. \tag{2.18}$$

Where $2A = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$ and $|A|$ is the area of the finite element triangle. Now we acquire the Jacobian of the transformation $\mathcal{M}(\xi_0, \xi_1)$.

$$D\mathcal{M}(\xi_0, \xi_1) = \begin{bmatrix} \frac{\partial x}{\partial \xi_0} & \frac{\partial x}{\partial \xi_1} \\ \frac{\partial y}{\partial \xi_0} & \frac{\partial y}{\partial \xi_1} \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}. \tag{2.19}$$

Next, we calculate the absolute value of the determinant.

$$|\det(D\mathcal{M}(\xi))| = |(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)| = 2|A|. \tag{2.20}$$

We substitute into (2.6) to get

$$\int_{FE} f(x, y)\phi_i^{(FE)}(x, y)dxdy = 2|A| \int_{RE} f(\mathcal{M}(\xi_0, \xi_1))\phi_i^{(RE)}(\xi_0, \xi_1)d\xi_0 d\xi_1. \tag{2.21}$$

Furthermore, we can use

$$\nabla\phi_i^{(FE)}(x, y) = \begin{bmatrix} \partial_x \\ \partial_y \end{bmatrix} \phi_i^{(RE)}(\xi_0, \xi_1) = \begin{bmatrix} \frac{\partial \xi_0}{\partial x} & \frac{\partial \xi_1}{\partial x} \\ \frac{\partial \xi_0}{\partial y} & \frac{\partial \xi_1}{\partial y} \end{bmatrix} \begin{bmatrix} \partial_{\xi_0} \\ \partial_{\xi_1} \end{bmatrix} \phi_i^{(RE)}(\xi_0, \xi_1), \tag{2.22}$$

when derivatives of test functions are involved. From the inverse transformation $\mathcal{M}^{-1}(x, y)$ (2.18) we have

$$\begin{bmatrix} \frac{\partial \xi_0}{\partial x} & \frac{\partial \xi_1}{\partial x} \\ \frac{\partial \xi_0}{\partial y} & \frac{\partial \xi_1}{\partial y} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_2 - y_0 & y_0 - y_1 \\ x_0 - x_2 & x_1 - x_0 \end{bmatrix}. \tag{2.23}$$

2.2 Derivation of Order 2 Lagrange Finite Element

In our numerical simulations, we use a continuous Lagrange order 2 finite element. We now provide a method for the derivation of this finite element on an interval and a triangle. Additionally, following a similar procedure, we can derive the basis functions for other domains. The resource [11] states basis functions for many finite elements.

This is not implemented in code we instead use the Firedrake implementation of CG order 2.

For each element, we have basis functions $\phi_i(\mathbf{x})$ which have a linear combination from the set $\{1, x, x^2, y, y^2, z, z^2, xy, yz, xz\}$ and locations on a domain ℓ_i. To find the coefficients of the linear combination we must solve

$$\phi_i(\ell_j) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases} \tag{2.24}$$

In section 2.1 we discuss how to map a finite element to its reference finite element.

2.2.1 Interval (1D)

Therefore, we have $i \in \{0, 1, 2\}$ and basis functions

$$\phi_i(x) = \xi_{i0} + \xi_{i1}x + \xi_{i2}x^2, \tag{2.25}$$

with $\ell_0 = 0, \ell_1 = 1$ and $\ell_2 = 0.5$. Thus by using (2.24) we get equations to solve

$$\phi_i(\ell_0) = \phi_i(0) = \xi_{i0} = \delta_{i0}, \tag{2.26}$$

$$\phi_i(\ell_1) = \phi_i(1) = \xi_{i0} + \xi_{i1} + \xi_{i2} = \delta_{i1}, \tag{2.27}$$

$$\phi_i(\ell_2) = \phi_i(0.5) = \xi_{i0} + \frac{\xi_{i1}}{2} + \frac{\xi_{i2}}{4} = \delta_{i2}, \tag{2.28}$$

where δ represents Kronecker delta. This can be put into matrix form

$$\begin{bmatrix} \xi_{i0} & \xi_{i1} & \xi_{i2} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1/2 \\ 0 & 1 & 1/4 \end{bmatrix} = \begin{bmatrix} \delta_{i0} & \delta_{i1} & \delta_{i2} \end{bmatrix}. \tag{2.29}$$

It is trivial to implement this into a matrix for all basis functions in this element.

$$\begin{bmatrix} \xi_{00} & \xi_{01} & \xi_{02} \\ \xi_{10} & \xi_{11} & \xi_{12} \\ \xi_{20} & \xi_{21} & \xi_{22} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1/2 \\ 0 & 1 & 1/4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.30}$$

Thus we get

$$\begin{bmatrix} \xi_{00} & \xi_{01} & \xi_{02} \\ \xi_{10} & \xi_{11} & \xi_{12} \\ \xi_{20} & \xi_{21} & \xi_{22} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1/2 \\ 0 & 1 & 1/4 \end{bmatrix}^{-1}, \tag{2.31}$$

$$\begin{bmatrix} \xi_{00} & \xi_{01} & \xi_{02} \\ \xi_{10} & \xi_{11} & \xi_{12} \\ \xi_{20} & \xi_{21} & \xi_{22} \end{bmatrix} = \begin{bmatrix} 1 & -3 & 2 \\ 0 & -1 & 2 \\ 0 & 4 & -4 \end{bmatrix}. \tag{2.32}$$

This leads to the basis functions

$$\phi_0(x) = 2x^2 - 3x + 1, \tag{2.33}$$

$$\phi_1(x) = x(2x - 1), \tag{2.34}$$

$$\phi_2(x) = 4x(1 - x). \tag{2.35}$$

With their visual representation in Figure 2.1.

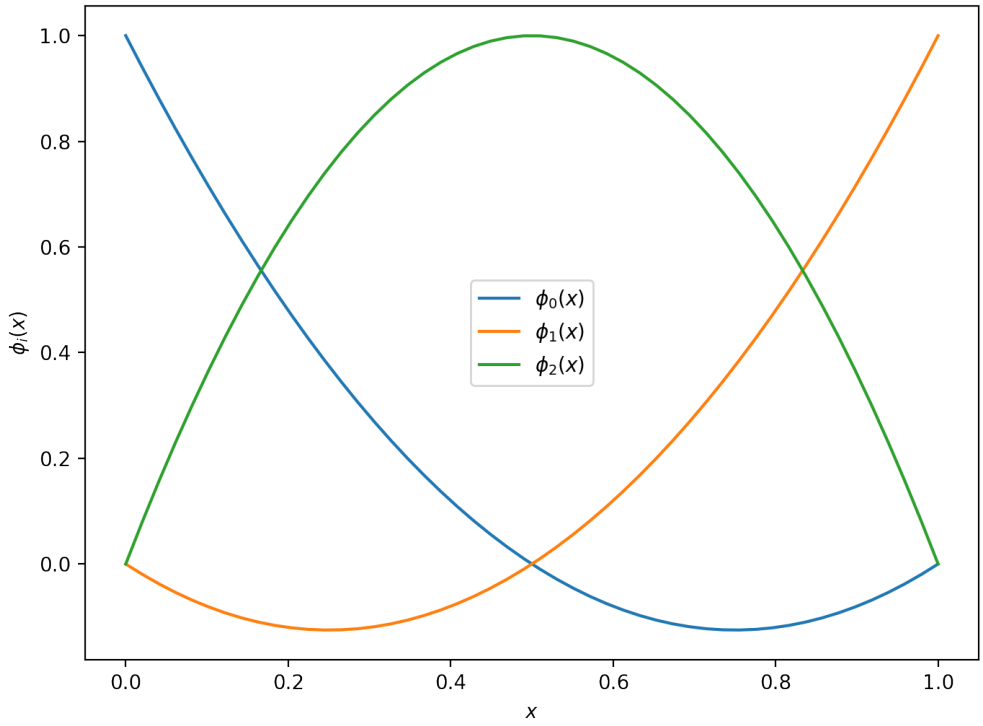


Figure 2.1: Lagrage order 2 basis functions on interval $[0, 1]$

2.2.2 Triangle (2D)

For a Lagrange element of order 2 on a triangle we have $i \in \{0, 1, 2, 3, 4, 5\}$ and the basis functions are of the form

$$\phi_i(x, y) = \xi_{i0} + \xi_{i1}x + \xi_{i2}y + \xi_{i3}xy + \xi_{i4}x^2 + \xi_{i5}y^2, \tag{2.36}$$

where the domain is $x + y \leq 1, x \geq 0$ and $y \geq 0$. For ℓ_i we get

$$\ell_0 = (0,0), \qquad \ell_3 = (1/2, 1/2), \tag{2.37}$$

$$\ell_1 = (1,0), \qquad \ell_4 = (0, 1/2), \tag{2.38}$$

$$\ell_2 = (0,1), \qquad \ell_5 = (1/2, 0). \tag{2.39}$$

This leads to the set of equations

$$\phi_i(\ell_0) = \xi_{i0}, \tag{2.40}$$

$$\phi_i(\ell_1) = \xi_{i0} + \xi_{i1} + \xi_{i4}, \tag{2.41}$$

$$\phi_i(\ell_2) = \xi_{i0} + \xi_{i2} + \xi_{i5}, \tag{2.42}$$

$$\phi_i(\ell_3) = \xi_{i0} + \frac{\xi_{i1} + \xi_{i2}}{2} + \frac{\xi_{i3} + \xi_{i4} + \xi_{i5}}{4}, \tag{2.43}$$

$$\phi_i(\ell_4) = \xi_{i0} + \frac{\xi_{i2}}{2} + \frac{\xi_{i5}}{4}, \tag{2.44}$$

$$\phi_i(\ell_5) = \xi_{i0} + \frac{\xi_{i1}}{2} + \frac{\xi_{i4}}{4}. \tag{2.45}$$

Therefore, using the equations created by (2.24) we get the matrix of coefficients

$$\begin{bmatrix} \xi_{00} & \xi_{01} & \xi_{02} & \xi_{03} & \xi_{04} & \xi_{05} \\ \xi_{10} & \xi_{11} & \xi_{12} & \xi_{13} & \xi_{14} & \xi_{25} \\ \xi_{20} & \xi_{21} & \xi_{22} & \xi_{23} & \xi_{24} & \xi_{25} \\ \xi_{30} & \xi_{31} & \xi_{32} & \xi_{33} & \xi_{34} & \xi_{35} \\ \xi_{40} & \xi_{41} & \xi_{42} & \xi_{43} & \xi_{44} & \xi_{45} \\ \xi_{50} & \xi_{51} & \xi_{52} & \xi_{53} & \xi_{54} & \xi_{55} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1/4 & 0 & 0 \\ 0 & 1 & 0 & 1/4 & 0 & 1/4 \\ 0 & 0 & 1 & 1/4 & 1/4 & 0 \end{bmatrix}^{-1}, \tag{2.46}$$

$$\begin{bmatrix} \xi_{00} & \xi_{01} & \xi_{02} & \xi_{03} & \xi_{04} & \xi_{05} \\ \xi_{10} & \xi_{11} & \xi_{12} & \xi_{13} & \xi_{14} & \xi_{25} \\ \xi_{20} & \xi_{21} & \xi_{22} & \xi_{23} & \xi_{24} & \xi_{25} \\ \xi_{30} & \xi_{31} & \xi_{32} & \xi_{33} & \xi_{34} & \xi_{35} \\ \xi_{40} & \xi_{41} & \xi_{42} & \xi_{43} & \xi_{44} & \xi_{45} \\ \xi_{50} & \xi_{51} & \xi_{52} & \xi_{53} & \xi_{54} & \xi_{55} \end{bmatrix} = \begin{bmatrix} 1 & -3 & -3 & 4 & 2 & 2 \\ 0 & -1 & 0 & 0 & 2 & 0 \\ 0 & 0 & -1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & -4 & 0 & -4 \\ 0 & 4 & 0 & -4 & -4 & 0 \end{bmatrix}. \tag{2.47}$$

Thus leading to basis functions

$$\phi_0(x,y) = 2(x+y)^2 - 3(x+y) + 1, \tag{2.48}$$

$$\phi_1(x,y) = x(2x-1), \tag{2.49}$$

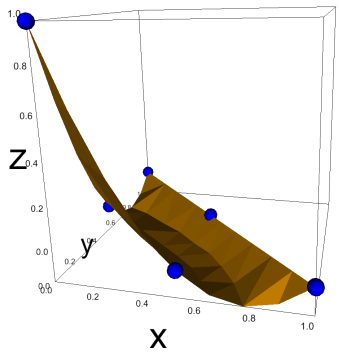
$$\phi_2(x,y) = y(2y-1), \tag{2.50}$$

$$\phi_3(x,y) = 4xy, \tag{2.51}$$

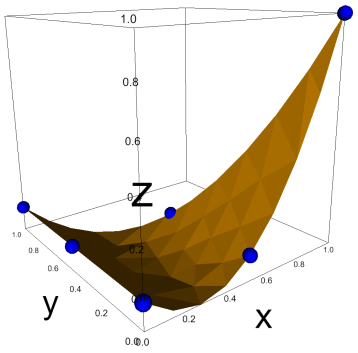
$$\phi_4(x,y) = 4y(1-x-y), \tag{2.52}$$

$$\phi_5(x,y) = 4x(1-x-y). \tag{2.53}$$

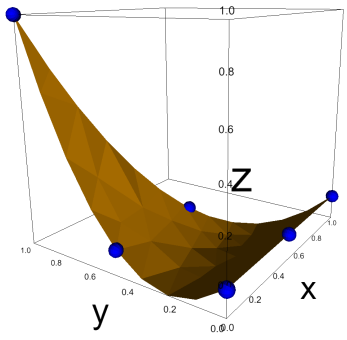
With the visual representation in Figure 2.3.



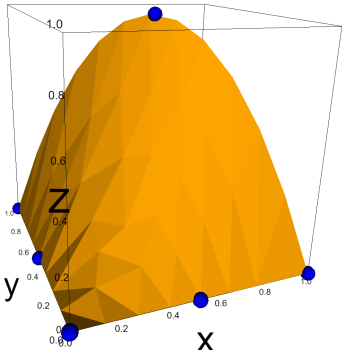
(a) $\phi_0(x, y) = 2(x + y)^2 - 3(x + y) + 1$,



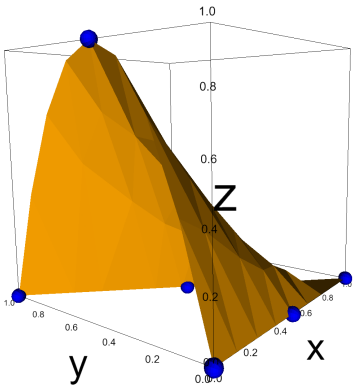
(b) $\phi_1(x, y) = x(2x - 1)$,



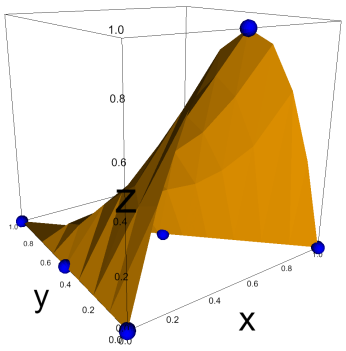
(c) $\phi_2(x, y) = y(2y - 1)$,



(d) $\phi_3(x, y) = 4xy$,



(e) $\phi_4(x, y) = 4y(1 - x - y)$,



(f) $\phi_5(x, y) = 4x(1 - x - y)$,

Figure 2.3: Basis Functions for Order 2 Lagrange Finite Element on a Triangle.

2.3 Derivation of the Linear System

We wish to find a matrix $\mathbb{B}^{(G)}$ and vector $\mathbf{F}^{(G)}$ so that the solution $\mathbf{u}$ to $\mathbb{B}^{(G)}\mathbf{u} = \mathbf{F}^{(G)}$ contains the coefficients for u in the discretised space. We can use a local formulation approach on the domain of a single finite element (Ω_{FE}). Thus we calculate the local $\mathbb{B}^{(L)}$ and $\mathbf{F}^{(L)}$. Then using a local to global map we add the results to the global system. Here is an algorithm to add the local system to the global system.

```
for j in range(d):
    F_Global[Map[j]] += F_Local[j]
    for i in range(d):
        B_Global_Sparse[Map[j], Map[i]] += B_Local[j, i]
```

This approach is well known for one test function. But when multiple test functions are involved it may not be clear how this translates to a linear system. Thus we show the derivation of the linear system for the $(MMAP)$ method.

In these derivations of linear systems, we will use a finite element with two basis functions. In this case $d = 2$. But it is trivial how to introduce more basis functions. Additionally, Firedrake [4] does this automatically thus it does not need to be implemented in code.

2.3.1 Linear System for (SP)

We will discretise u from (1.20) on the finite element domain with basis functions ϕ_0 and ϕ_1. Thus we substitute $u = u_0\phi_0 + u_1\phi_1$ into (1.20) to get equations

$$\varepsilon^{-1}u_0 a_{||}(\psi_0,\psi_i)+\varepsilon^{-1}u_1 a_{||}(\psi_1,\psi_i)+u_0 a_{\perp}(\psi_0,\psi_i)+u_1 a_{\perp}(\psi_1,\psi_i) = \int_{\Omega_{FE}} f\psi_i d\mathbf{x}. \quad (2.54)$$

For $i \in \{0,1\}$. It should be noted that the integrals are over the finite element domain Ω_{FE}. Thus from (2.54) we get linear system

$$\begin{bmatrix} \varepsilon^{-1}a_{||}(\psi_0,\psi_0) + a_{\perp}(\psi_0,\psi_0) & \varepsilon^{-1}a_{||}(\psi_1,\psi_0) + a_{\perp}(\psi_1,\psi_0) \\ \varepsilon^{-1}a_{||}(\psi_0,\psi_1) + a_{\perp}(\psi_0,\psi_1) & \varepsilon^{-1}a_{||}(\psi_1,\psi_1) + a_{\perp}(\psi_1,\psi_1) \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} = \begin{bmatrix} \int_{\Omega_{FE}} f\psi_0 d\mathbf{x} \\ \int_{\Omega_{FE}} f\psi_1 d\mathbf{x} \end{bmatrix}. \quad (2.55)$$

Then we add this local system to the global system. For $\varepsilon \ll 0$ we can see this matrix is ill-conditioned thus the (SP) method only works for $\varepsilon \approx 1$.

2.3.2 Linear System for $(MMAP)$

Now we will discretise u and q from (1.22). For the local formulation we get test functions $\hat{u}_0, \hat{u}_1, \hat{q}_0$ and $\hat{q}_1$. Now we substitute $u = u_0\hat{u}_0 + u_1\hat{u}_1$ and $q = q_0\hat{q}_0 + q_1\hat{q}_1$

into (1.22). Thus we get equations

$$\begin{cases} u_0 a_\perp(\hat{u}_0, \hat{u}_i) + u_1 a_\perp(\hat{u}_1, \hat{u}_i) + q_0 a_{||}(\hat{q}_0, \hat{u}_i) + q_1 a_{||}(\hat{q}_1, \hat{u}_i) & = \int_{\Omega_{FE}} f \hat{u}_i d\mathbf{x}, \\ u_0 a_{||}(\hat{u}_0, \hat{q}_i) + u_1 a_{||}(\hat{u}_1, \hat{q}_i) - \varepsilon q_0 a_{||}(\hat{q}_0, \hat{q}_i) - \varepsilon q_1 a_{||}(\hat{q}_1, \hat{q}_i) & = 0. \end{cases} \tag{2.56}$$

For $i \in \{0, 1\}$. Thus from (2.56), we get the linear system

$$\begin{bmatrix} a_\perp(\hat{u}_0, \hat{u}_0) & a_\perp(\hat{u}_1, \hat{u}_0) & a_{||}(\hat{q}_0, \hat{u}_0) & a_{||}(\hat{q}_1, \hat{u}_0) \\ a_\perp(\hat{u}_0, \hat{u}_1) & a_\perp(\hat{u}_1, \hat{u}_1) & a_{||}(\hat{q}_0, \hat{u}_1) & a_{||}(\hat{q}_1, \hat{u}_1) \\ a_{||}(\hat{u}_0, \hat{q}_0) & a_{||}(\hat{u}_1, \hat{q}_0) & -\varepsilon a_{||}(\hat{q}_0, \hat{q}_0) & -\varepsilon a_{||}(\hat{q}_1, \hat{q}_0) \\ a_{||}(\hat{u}_0, \hat{q}_1) & a_{||}(\hat{u}_1, \hat{q}_1) & -\varepsilon a_{||}(\hat{q}_0, \hat{q}_1) & -\varepsilon a_{||}(\hat{q}_1, \hat{q}_1) \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ q_0 \\ q_1 \end{bmatrix} = \begin{bmatrix} \int_{\Omega_{FE}} f \hat{u}_0 d\mathbf{x} \\ \int_{\Omega_{FE}} f \hat{u}_1 d\mathbf{x} \\ 0 \\ 0 \end{bmatrix}. \tag{2.57}$$

Then we add this local system to the global system. We can see this matrix is well-conditioned thus the $(MMAP)$ method works for $0 < \varepsilon < 1$.

2.4 Example 1

We will solve the PDE (1.1) with $\Omega = (0, 1)^2, \Gamma_D = \{y = 0 \text{ or } y = 1\}$ and $\Gamma_N = \{x = 0 \text{ or } x = 1\}$. We chose a magnetic field

$$\mathbf{b} = \frac{\mathbf{B}}{|\mathbf{B}|}, \mathbf{B} = \begin{bmatrix} \alpha(2y - 1)\cos(m\pi x) + \pi \\ \alpha m\pi(y^2 - y)\sin(m\pi x) \end{bmatrix}. \tag{2.58}$$

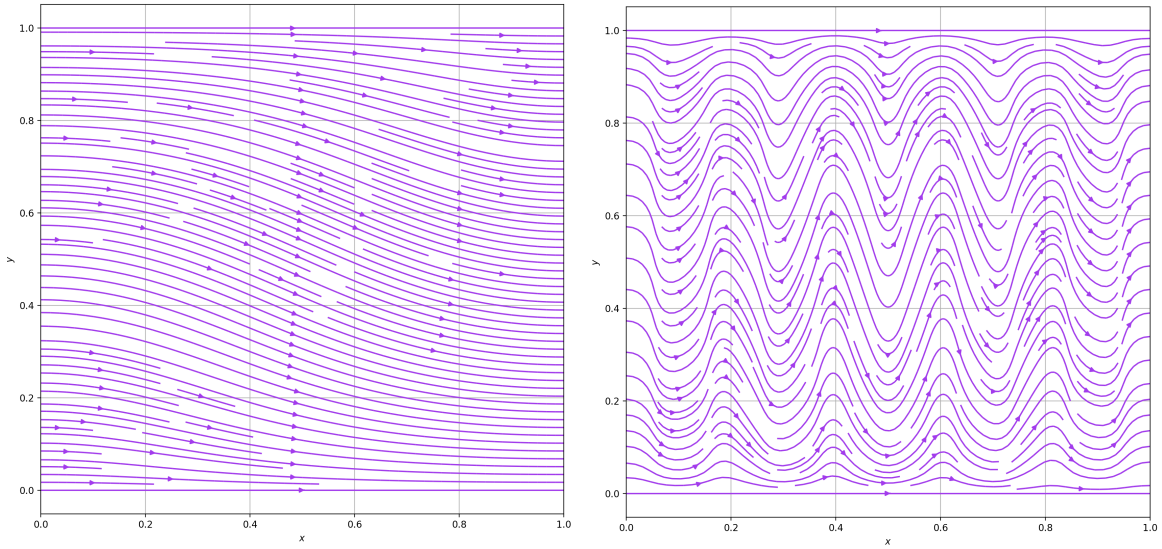
Thus we have $\Gamma_{in} := \{x = 0\}$ and we have the streamlines defined by

$$\pi y + \alpha(y^2 - y)\cos(m\pi x) = C, \tag{2.59}$$

with $0 \leq C \leq \pi$. This is calculated in Appendix B. When $\varepsilon \ll 1$ the solution must be constant along streamlines. Hence, we calculate the streamlines to find exact solutions to test. This leads to an exact solution

$$u = \sin(\pi y + \alpha(y^2 - y)\cos(m\pi x)) + \varepsilon \cos(2\pi x)\sin(\pi y). \tag{2.60}$$

This test problem came from [2]. However, the same or similar problem is very popular and appears in papers [12], [3], [5] and [6]. We will cover three sub-examples with $(\alpha = 0), (\alpha = 2, m = 1)$ and $(\alpha = 2, m = 10)$. Thus we get electromagnetic fields shown in Figure 2.4. The field for $\alpha = 0$ is not shown as it is a simple vector field with $\mathbf{b} = [1, 0]$.



(a) $\alpha = 2, m = 1$.

(b) $\alpha = 2, m = 10$.

Figure 2.4: Electromagetic vector field $\mathbf{b}$ from (2.58).

In Figure 2.5 we have the exact solution u (2.60) to Example 1. We have the source term f which is too complicated to write analytically so the visualisation is given in Figure 2.6. We consider the equation with $\varepsilon = 0.1$ and $\varepsilon = 10^{-10}$. We notice the $\varepsilon = 0.1$ and $\varepsilon = 10^{-10}$ variants are visually similar for the source term f and exact term u, which suggests we should consider a method discussed in section 3.1.2. The main difference is the solution u is not constant on streamlines of the vector field $\mathbf{b}$ this makes logical sense because increasing the anisotropic strength (make ε smaller) leads to less diffusion perpendicular to the magnetic field and undisturbed diffusion parallel to the magnetic field.

In Figures 2.7 and 2.8 we numerically simulate the PDE created by Example 1 with a 20×20 grid, where each finite element is a square with width 0.05 discretised with an order 2 Lagrange element. Also, each red dot represents a numerical calculation with varying ε. Where the degrees of freedom for $(MMAP), (PF), (LM)$ are 3362, (SP) is 1681 and (AP) is 8405.

In Figure 2.7 we compare methods $(MMAP), (PF)$ and (AP). The error plot for $(MMAP)$ is located under the error plot for (AP) this strongly suggests these methods will calculate a similar value for the error. Thus for further numerical calculations, we will only consider the $(MMAP)$ because for similar error it has fewer degrees of freedom than the (AP) method. This decreases the amount of computational power used. Additionally, when the vector field is not anisotropic ($\alpha = 0$) all

methods produce small errors for $0 < \varepsilon \leq 1$. Also, Figure 2.7c visually suggests when the vector field is mildly anisotropic the $(MMAP)$ method performs better than the (PF) method. However, from Figure 2.7e we infer when the vector field is highly anisotropic (PF) performs better than $(MMAP)$.

In the numerical calculations shown in Figure 2.8, it follows our logic that the (LM) method produces a small error for $\varepsilon \ll 1$ and a much larger error for $\varepsilon \approx 1$. Also, it shows the (SP) method failing when $\varepsilon \ll 1$. This strongly suggests that we have implemented our code correctly. For further numerical calculations we use $(MMAP)$ instead of (LM) and (SP) as in Figure 2.8 $(MMAP)$ has the smallest error for $0 < \varepsilon \leq 1$.

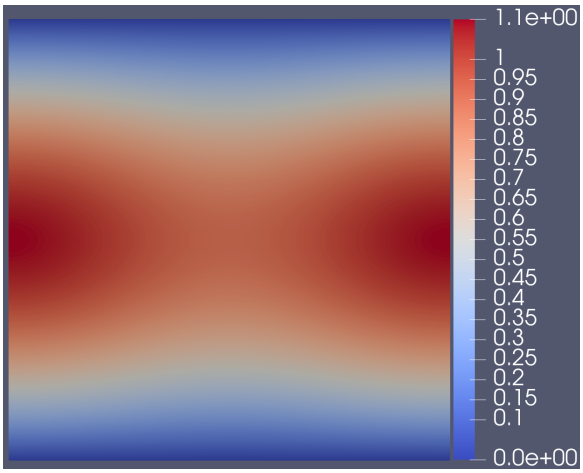
However, the (SP) method has fewer degrees of freedom than the $(MMAP)$ method and from Figure 2.8 there exists an ε_0 where when $\varepsilon \geq \varepsilon_0$ the (SP) method has a similar error to the $(MMAP)$ method. Thus we can propose a new method that involves using $(MMAP)$ when $\varepsilon < \varepsilon_0$ and using the (SP) method when $\varepsilon \leq \varepsilon_0$. One problem with this method is that when the PDE becomes more anisotropic ε_0 increases and we are interested in methods for solving highly anisotropic diffusion problems with $\varepsilon \ll 1$. Thus this method is not further discussed.

L2 Error		$\alpha = 0$		$\alpha = 2, m = 1$		$\alpha = 2, m = 10$	
Size	dof	(PF)	$(MMAP)$	(PF)	$(MMAP)$	(PF)	$(MMAP)$
10×10	882	1.26×10^{-4}	1.26×10^{-4}	1.64×10^{-1}	2.25×10^{-4}	4.29×10^{-2}	3.16×10^{-1}
20×20	3362	1.58×10^{-5}	1.58×10^{-5}	4.01×10^{-2}	2.80×10^{-5}	2.94×10^{-2}	1.25×10^{-1}
40×40	13122	1.97×10^{-6}	1.97×10^{-6}	1.42×10^{-3}	3.44×10^{-6}	4.23×10^{-3}	1.29×10^{-2}
80×80	51842	2.46×10^{-7}	2.46×10^{-7}	2.38×10^{-5}	4.25×10^{-7}	9.71×10^{-4}	9.70×10^{-4}
160×160	206082	3.09×10^{-8}	3.09×10^{-8}	1.47×10^{-6}	5.25×10^{-8}	2.19×10^{-4}	6.36×10^{-5}

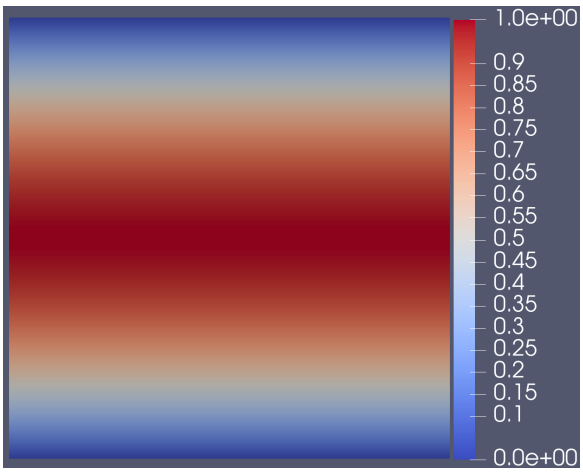
H1 Error		$\alpha = 0$		$\alpha = 2, m = 1$		$\alpha = 2, m = 10$	
Size	dof	(PF)	$(MMAP)$	(PF)	$(MMAP)$	(PF)	$(MMAP)$
10×10	882	8.16×10^{-3}	8.16×10^{-3}	1.00×10^{0}	1.42×10^{-2}	2.57×10^{0}	4.22×10^{0}
20×20	3362	2.04×10^{-3}	2.04×10^{-3}	3.99×10^{-1}	3.57×10^{-3}	1.21×10^{0}	1.99×10^{0}
40×40	13122	5.11×10^{-4}	5.11×10^{-4}	4.37×10^{-2}	8.89×10^{-4}	4.38×10^{-1}	3.20×10^{-1}
80×80	51842	1.28×10^{-4}	1.28×10^{-4}	7.23×10^{-3}	2.21×10^{-4}	2.01×10^{-1}	5.00×10^{-2}
160×160	206082	3.19×10^{-5}	3.19×10^{-5}	1.77×10^{-3}	5.49×10^{-5}	9.79×10^{-2}	1.10×10^{-2}

Table 2.1: Error for varying quadrilateral grid size for variants of Example 1 using (PF) and $(MMAP)$ with order 2 Lagrange finite element and $\varepsilon = 10^{-10}$.

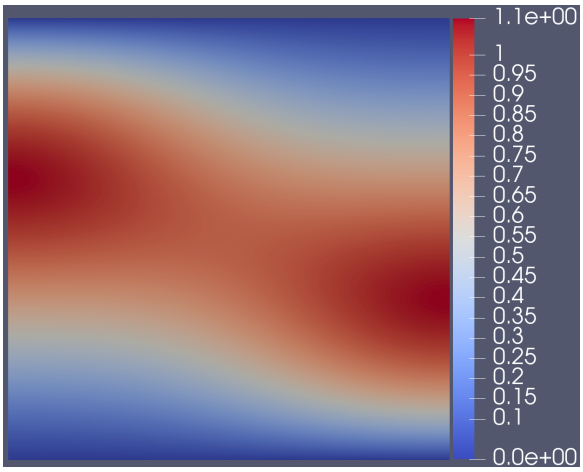
As can be seen in Table 2.1 when the number of finite elements in the grid has increased the error is reduced.



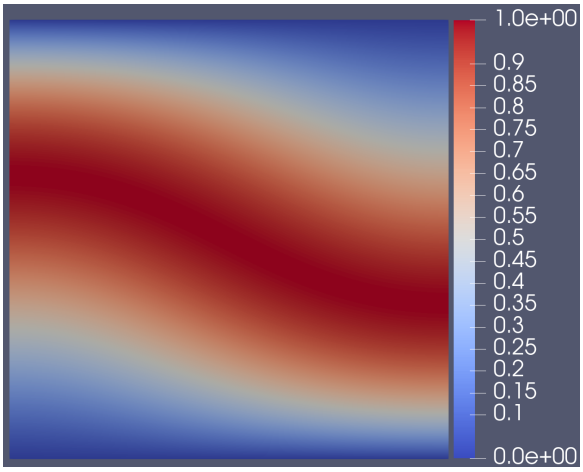
(a) $\alpha = 0$ and $\varepsilon = 0.1$.



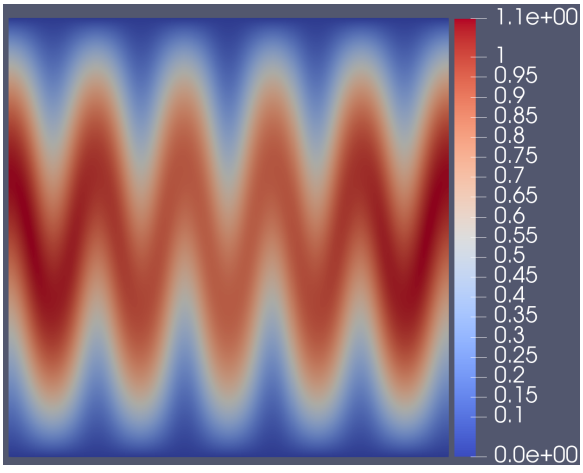
(b) $\alpha = 0$ and $\varepsilon = 10^{-10}$.



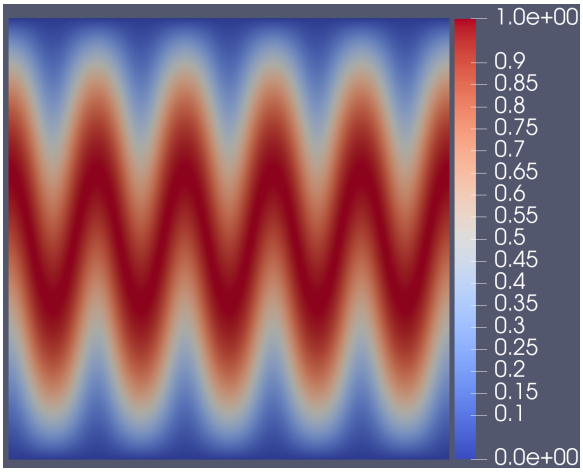
(c) $\alpha = 2, m = 1$ and $\varepsilon = 0.1$.



(d) $\alpha = 2, m = 1$ and $\varepsilon = 10^{-10}$.

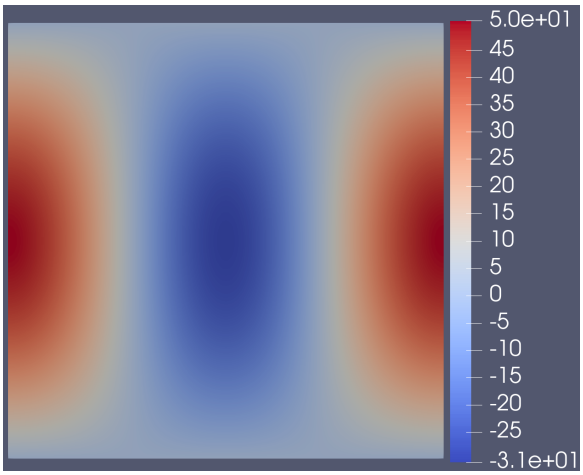


(e) $\alpha = 2, m = 10$ and $\varepsilon = 0.1$.

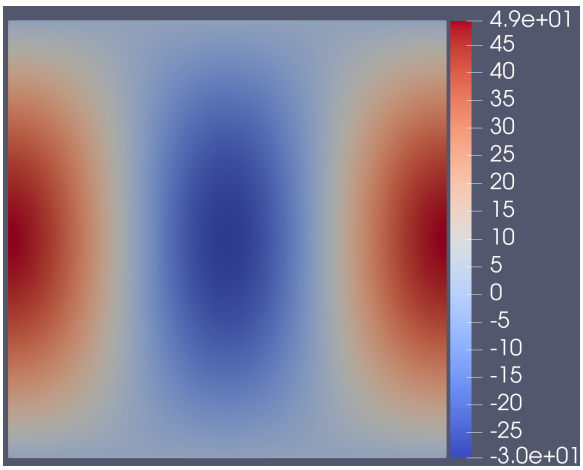


(f) $\alpha = 2, m = 10$ and $\varepsilon = 10^{-10}$.

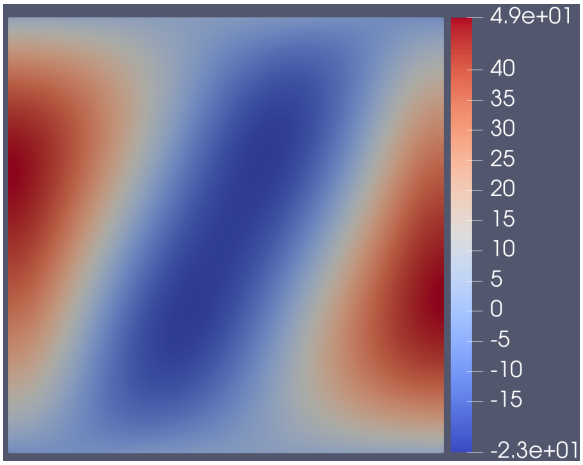
Figure 2.5: Exact solution u (2.60) for Example 1.



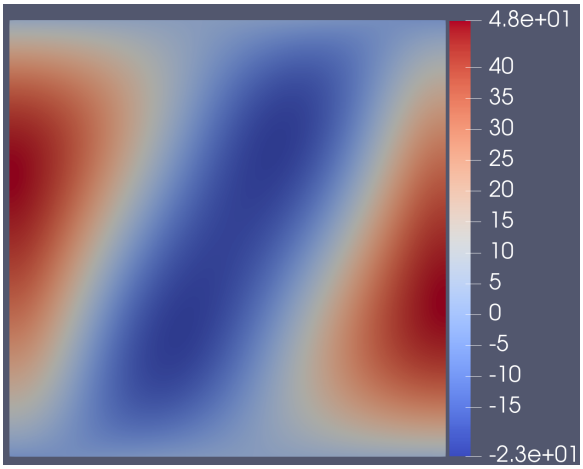
(a) $\alpha = 0$ and $\varepsilon = 0.1$.



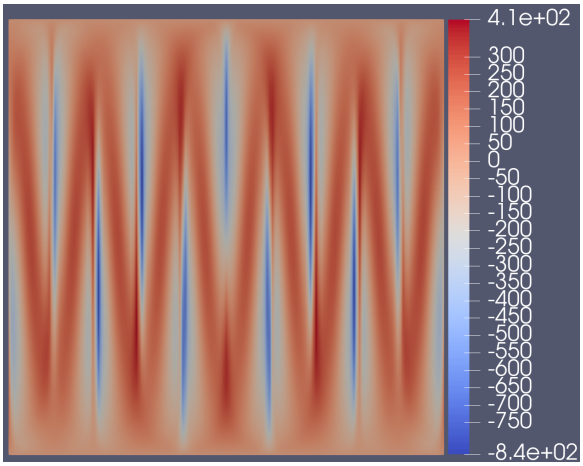
(b) $\alpha = 0$ and $\varepsilon = 10^{-10}$.



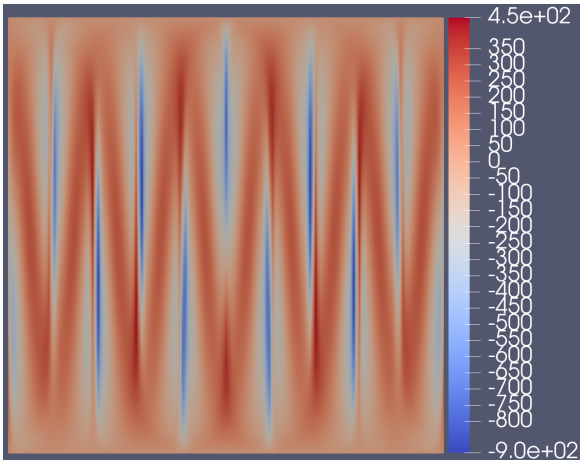
(c) $\alpha = 2, m = 1$ and $\varepsilon = 0.1$.



(d) $\alpha = 2, m = 1$ and $\varepsilon = 10^{-10}$.

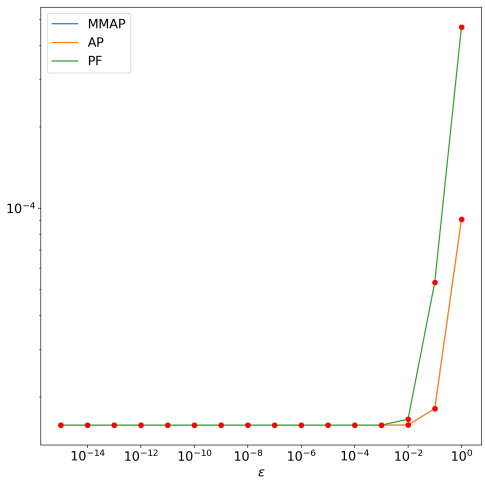


(e) $\alpha = 2, m = 10$ and $\varepsilon = 0.1$.

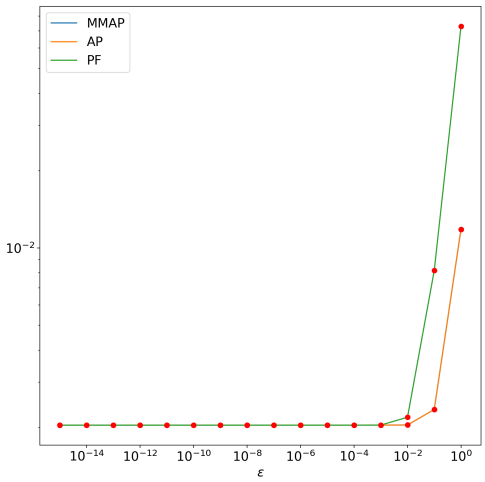


(f) $\alpha = 2, m = 10$ and $\varepsilon = 10^{-10}$.

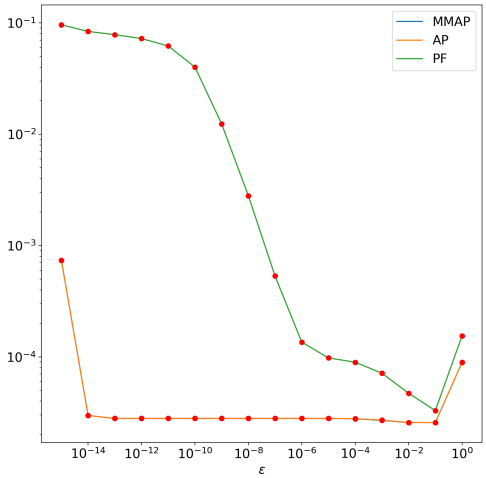
Figure 2.6: Source term f for Example 1.



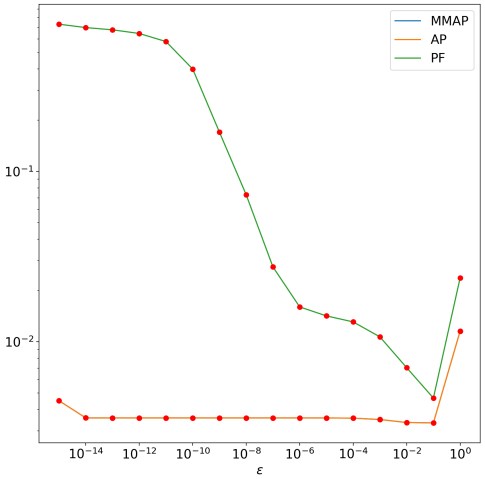
(a) L2 Error $\alpha = 0$.



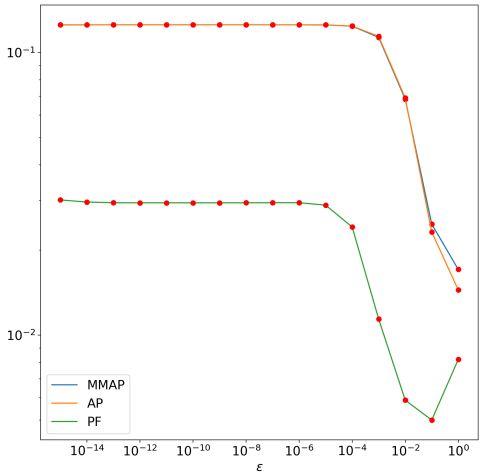
(b) H1 Error $\alpha = 0$.



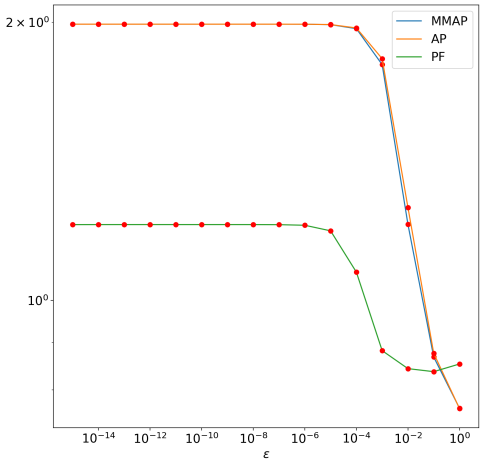
(c) L2 Error $\alpha = 2, m = 1$.



(d) H1 Error, $\alpha = 2, m = 1$.

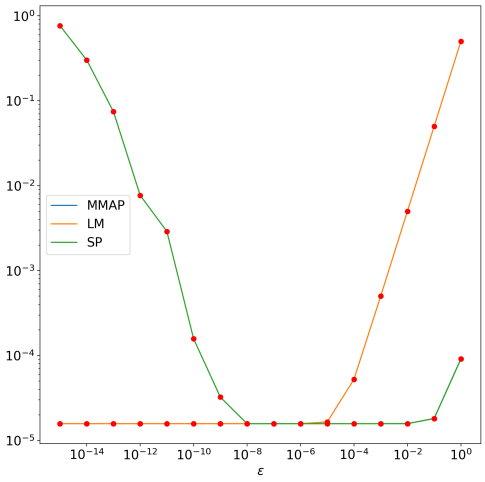


(e) L2 Error, $\alpha = 2, m = 10$.

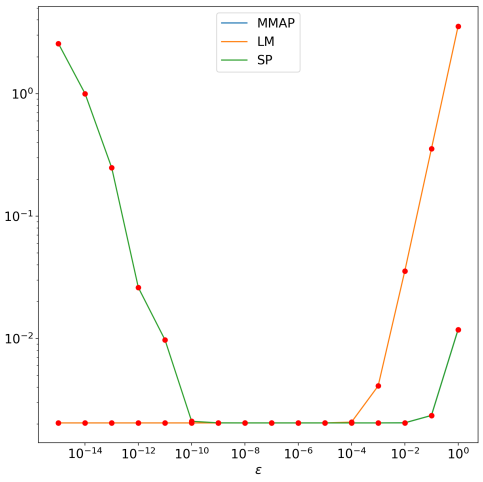


(f) H1 Error, $\alpha = 2, m = 10$.

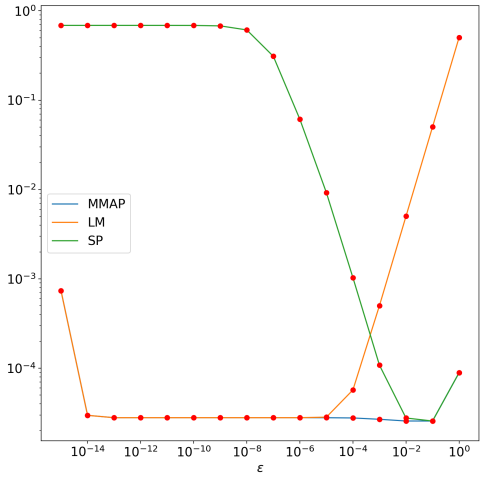
Figure 2.7: Numerical demonstration for solver $(MMAP)$, (AP) and (PF) on a 20×20 quadrilateral grid using CG order 2 for Example 1, with $x = \varepsilon$ and $y =$ Error.



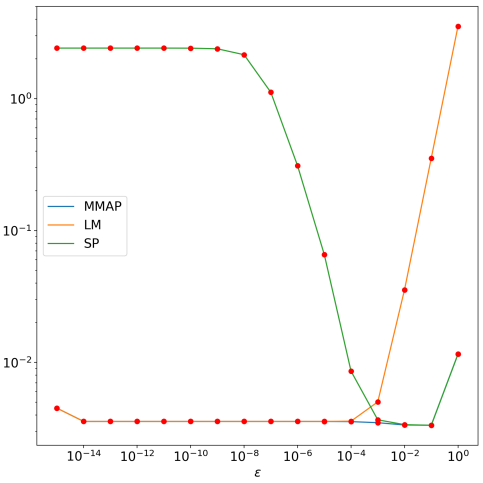
(a) L2 Error $\alpha = 0$.



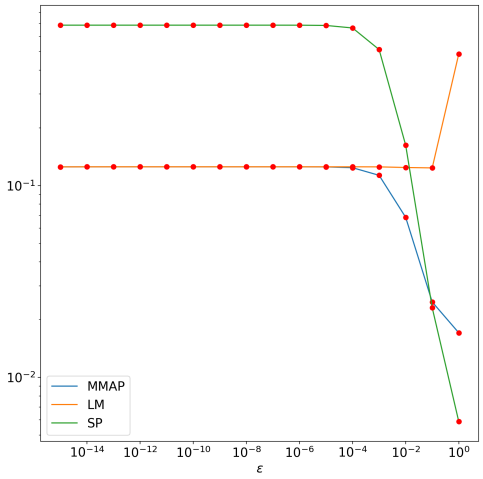
(b) H1 Error $\alpha = 0$.



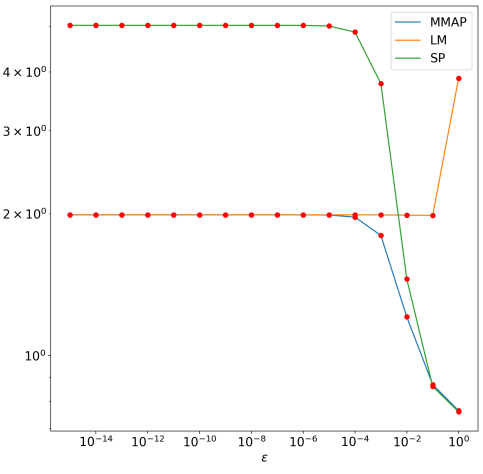
(c) L2 Error $\alpha = 2, m = 1$.



(d) H1 Error, $\alpha = 2, m = 1$.



(e) L2 Error, $\alpha = 2, m = 10$.



(f) H1 Error, $\alpha = 2, m = 10$.

Figure 2.8: Numerical demonstration for solver $(MMAP), (LM)$ and (SP) on a 20×20 quadrilateral grid using CG order 2 for Example 1, with $x = \varepsilon$ and $y =$ Error.

	$\Gamma_{in} := \{x = 0\}$		$\Gamma_{in} := \{x = 1\}$	
L2 Error	PF	$MMAP$	PF	$MMAP$
$\alpha = 0$	$2.4624E-07$	$2.4624E-07$	$2.4624E-07$	$2.4624E-07$
$\alpha = 2, m = 1$	$2.3834E-05$	$4.2451E-07$	$2.3834E-05$	$4.2451E-07$
$\alpha = 2, m = 10$	$9.7179E-04$	$9.7017E-04$	$9.7179E-04$	$9.7017E-04$

	$\Gamma_{in} := \{x = 0\}$		$\Gamma_{in} := \{x = 1\}$	
H1 Error	PF	$MMAP$	PF	$MMAP$
$\alpha = 0$	$1.2767E-04$	$1.2767E-04$	$1.2767E-04$	$1.2767E-04$
$\alpha = 2, m = 1$	$7.2261E-03$	$2.2108E-04$	$7.2261E-03$	$2.2108E-04$
$\alpha = 2, m = 10$	$2.0097E-01$	$4.9986E-02$	$2.0097E-01$	$4.9986E-02$

Table 2.2: Error for different definitions of Γ_{in} on an 80×80 quadrilateral grid with order 2 Lagrange finite element, 51842 degrees of freedom and $\varepsilon = 10^{-10}$.

Thus numerical experimentation shown in Table 2.2 strongly suggests that the set $\mathcal{L}$ can be set to zero anywhere on the streamline. In other words, if $\Gamma_{SL} :=$ { one point on every streamline } then we can have $\mathcal{L} := \{\lambda \in \mathcal{H}^1(\Omega) : \lambda|_{\Gamma_{SL} \cup \Gamma_D} = 0\}$.

2.5 Example 2, Annulus

We now propose an example of an annulus with closed field lines. We will solve the PDE (1.1) with $\Omega := \{x, y \in \mathbb{R}, 1 < x^2 + y^2 < 4\}$, $\Gamma_D := \{x, y \in \mathbb{R}, (x^2 + y^2 = 1 \text{ or } x^2 + y^2 = 4)\}$ and $\Gamma_N := \emptyset$. We chose the magnetic field

$$\mathbf{b} = \frac{\mathbf{B}}{|\mathbf{B}|} = \begin{bmatrix} -y \\ x \end{bmatrix} / \sqrt{x^2 + y^2}, \mathbf{B} = \begin{bmatrix} -y \\ x \end{bmatrix}, \tag{2.61}$$

which is visualised in Figure 2.9 and the streamlines follow the equation

$$x^2 + y^2 = R^2, \text{ with } 1 \leq R \leq 2. \tag{2.62}$$

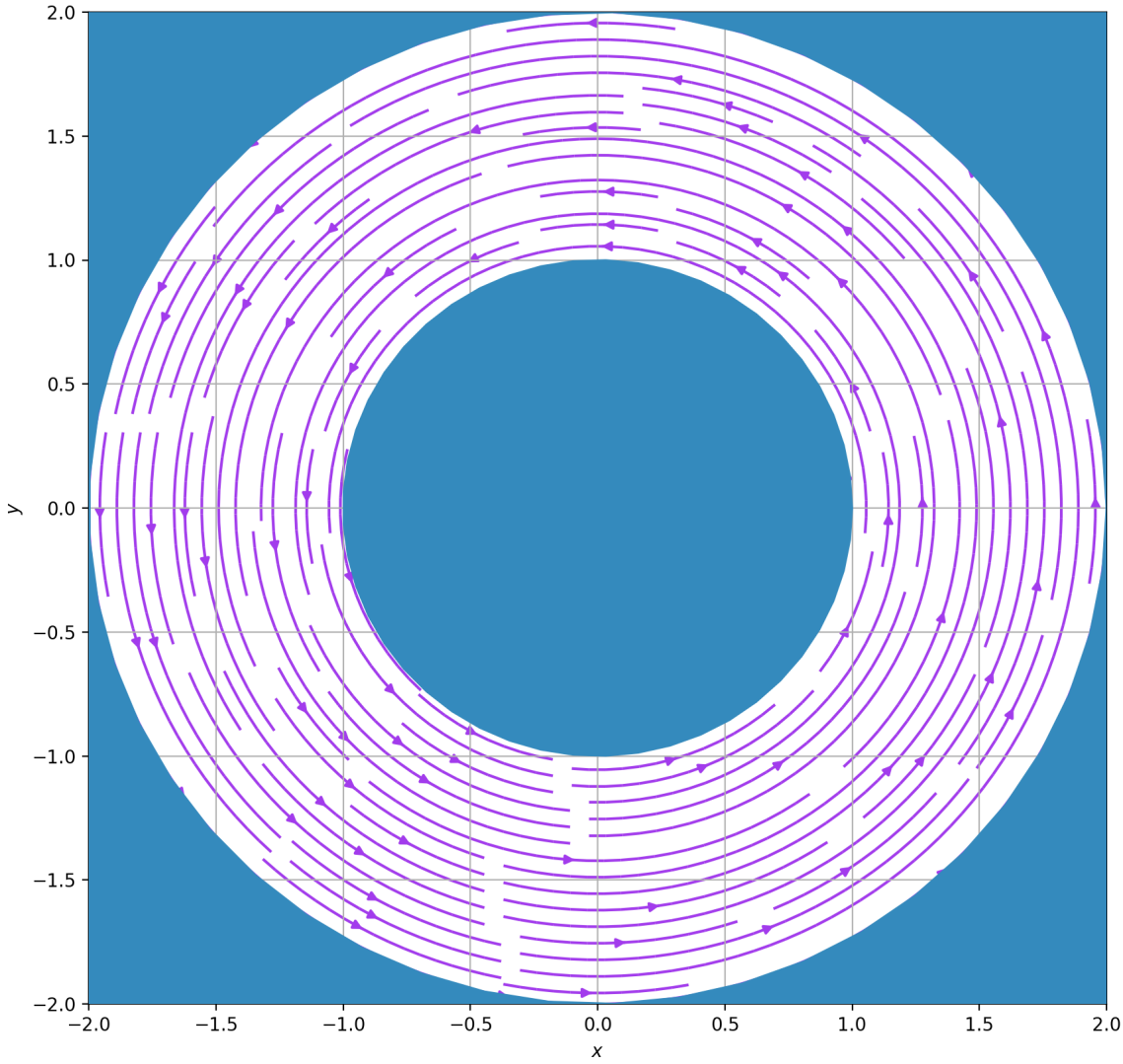


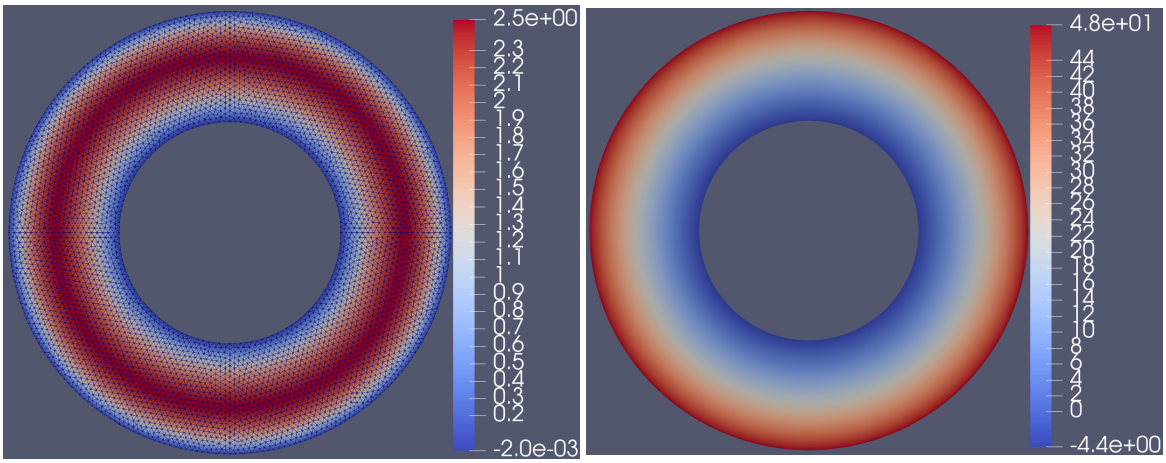
Figure 2.9: Electromagnetic field **b** for Example 2.

From Fig 2.9 it can be seen the magnetic field has closed lines. Therefore, for methods which involve Γ_{in} we use the stabilisation variants. However, from the numerical investigation of Example 1 from Table 2.2 we found that we can change Γ_{in} to Γ_{out} and get a similar error. This strongly suggests that Γ_{in} can represent a point on each streamline. Thus we will numerically investigate the consequence of setting $\Gamma_{in} := \{x, y \in \mathbb{R}, x > 0, y = 0\}$.

We will use the solution

$$u = (1 + \varepsilon)(x^2 + y^2 - 1)(4 - x^2 - y^2), \tag{2.63}$$

which is visualised with its source term f in Figure 2.10 at $\varepsilon = 0.1$.



(a) Exact solution u with mesh.

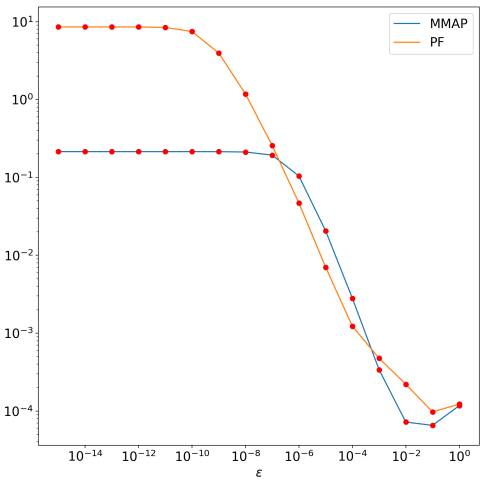
(b) Source term f.

Figure 2.10: Example 2 with $\varepsilon = 0.1$.

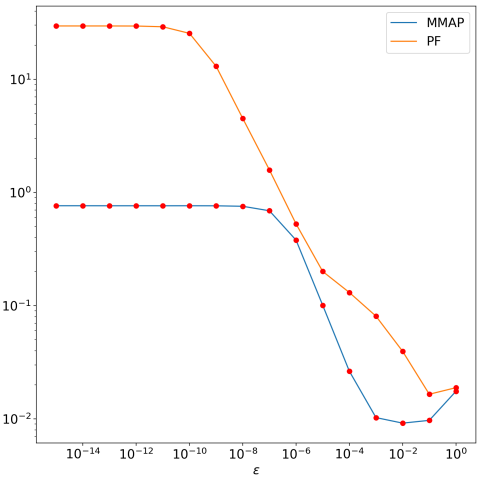
In Figure 2.11 we have numerical calculations for varying ε for Example 2 with degrees 36944 of freedom. As can be seen by Figures 2.11a and 2.11b when there are closed field lines the methods (PF) and $(MMAP)$ fail to produce accurate solutions for $\varepsilon \ll 1$.

However, in Figures 2.11c and 2.11d we change $\Gamma_{in} := \emptyset$ to $\Gamma_{in} := \{x, y \in \mathbb{R}, x > 0, y = 0\}$. This change makes a negligible difference to the (PF) method. It is currently unclear why there is no difference, the code has been checked multiple times but no error has been found. But, the change leads to a significant improvement for the $(MMAP)$. Therefore, we propose a new variant of the $(MMAP)$ where the definition of the set $\mathcal{L}$ is $\{\lambda \in \mathcal{H}^1(\Omega) : \lambda|_{\Gamma_{CL} \cup \Gamma_D} = 0\}$.

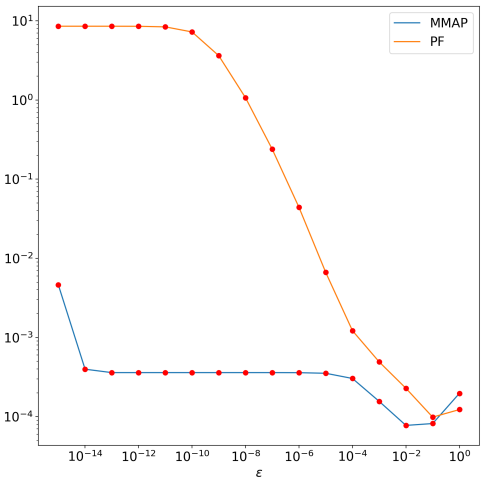
The new $(MMAP)$ variant is not discussed further because Figures 2.11e and 2.11f show methods $(MMAP\_STAB)$ and (PF_STAB) outperform the $(MMAP)$ variant. Thus for the following examples, we only consider methods $(MMAP\_STAB)$ and (PF_STAB).



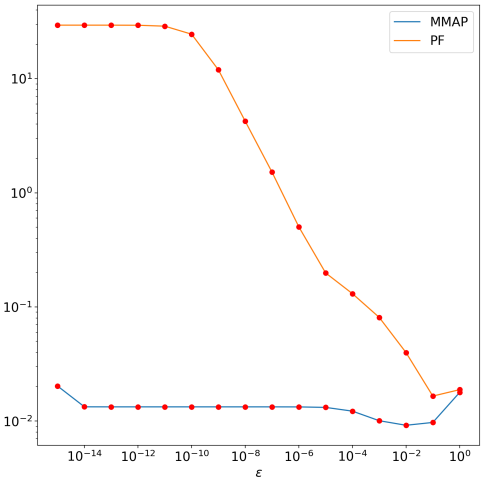
(a) L2 Error.



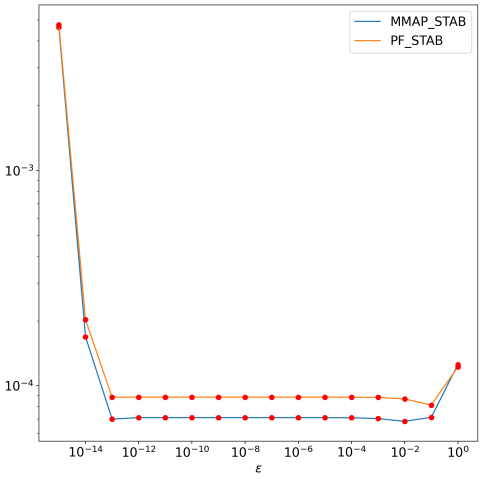
(b) H1 Error.



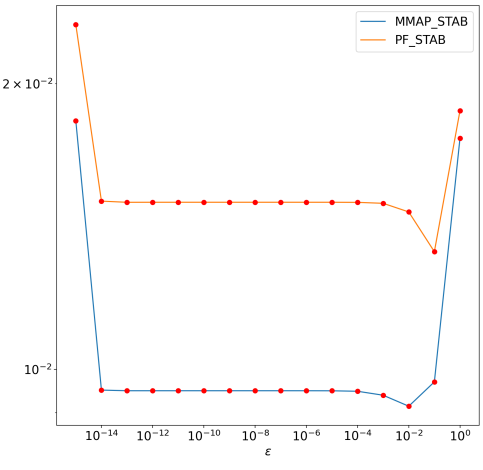
(c) L2 Error, with constraint.



(d) H1 Error, with constraint.



(e) L2 Error, with stabilisation.



(f) H1 Error, with stabilisation.

Figure 2.11: Numerical demonstration for solver $(MMAP), (MMAP\_STAB), (PF)$ and (PF_STAB) on a 20×20 quadrilateral grid using CG order 2 for Example 2. Also, $x = \varepsilon$ and $y =$ Error.

2.6 Example 3, Magnetic Islands

From [2] we get an example on a square domain with non-zero boundary conditions on Γ_D and with closed and open field lines. We will solve the PDE (1.1) with $\Omega = [0,1]^2, \Gamma_D = \{y = 0 \text{ or } y = 1\}$ and $\Gamma_N = \{x = 0 \text{ or } x = 1\}$. We choose a magnetic field such that

$$\mathbf{b} = \frac{\mathbf{B}}{|\mathbf{B}|}, \mathbf{B} = \begin{bmatrix} -\cos(\pi y) \\ 4a\sin(4\pi x) \end{bmatrix}, \tag{2.64}$$

which is visualised in Figure 2.12. Additionally, after some ODE analysis, we get the streamlines for this vector field. Which are

$$\sin(\pi y) - a\cos(4\pi x) = C, \tag{2.65}$$

with $-a \leq C \leq 1 + a$.

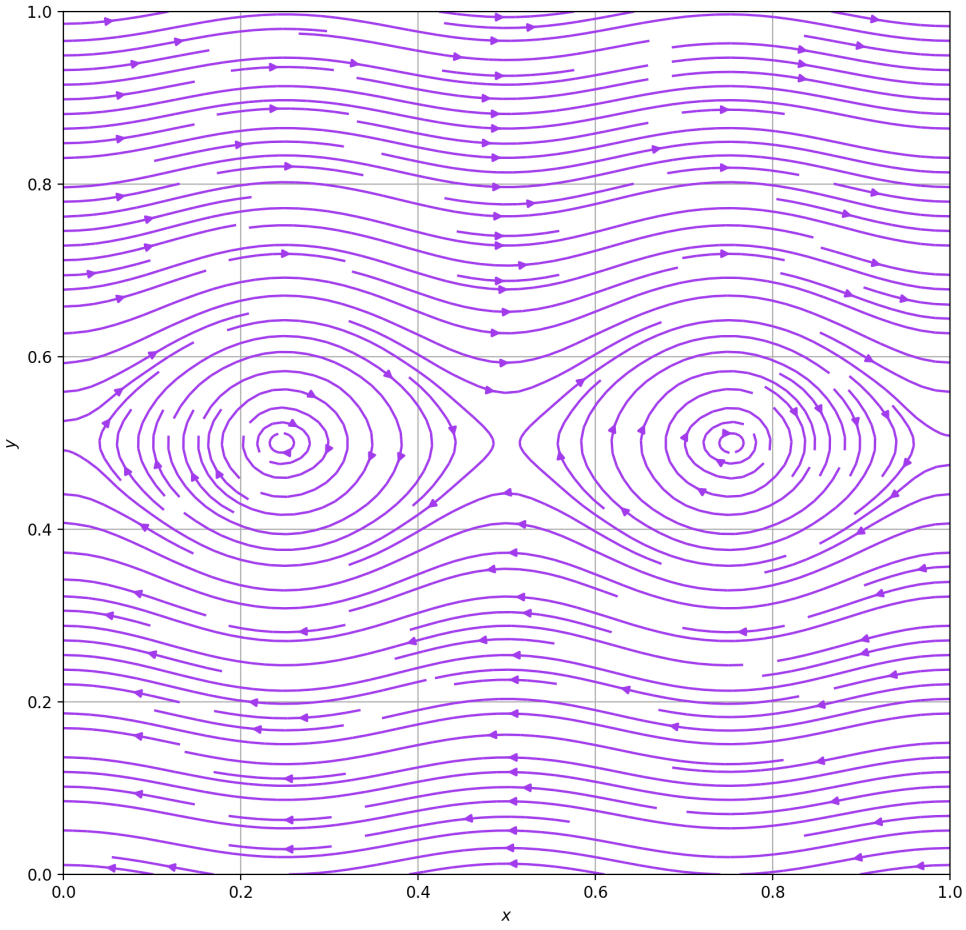


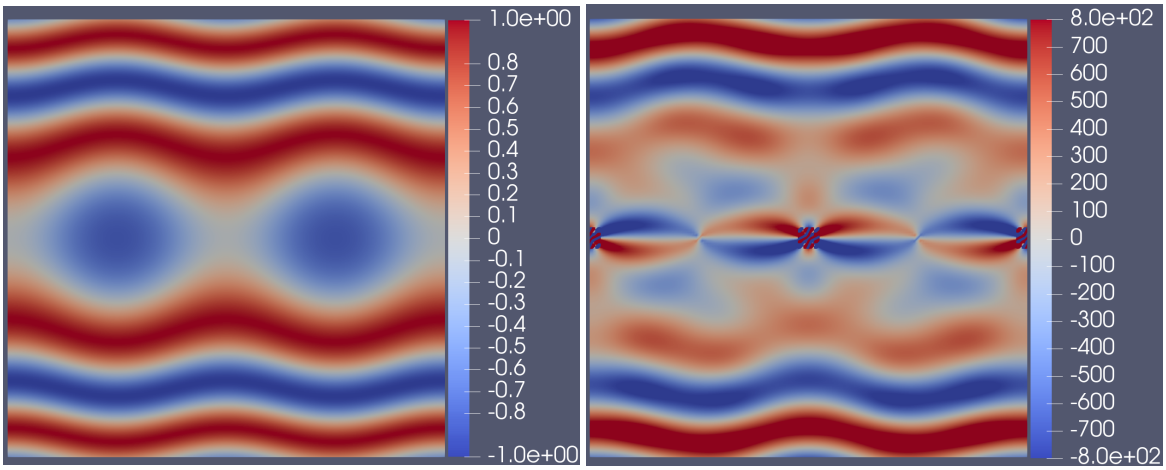
Figure 2.12: Electromagnetic field **b** for Example 3.

This example was proposed by [2] but it does not satisfy the restrictions we impose. The restrictions are $\mathbf{b}\cdot\mathbf{n} = 0$ on Γ_D this failed restriction can be seen in Figure 2.12 and $u = 0$ on Γ_D this failed restriction can be seen in Figure 2.13a. Thus, our numerical solvers will struggle to solve this example, this problem is rectified in Example 4. Additionally, [2] rectified this problem by defining $\Gamma_D := \{\mathbf{x} \in \partial\Omega : \mathbf{b} \cdot \mathbf{n} = 0\}$.

We will use the exact solution

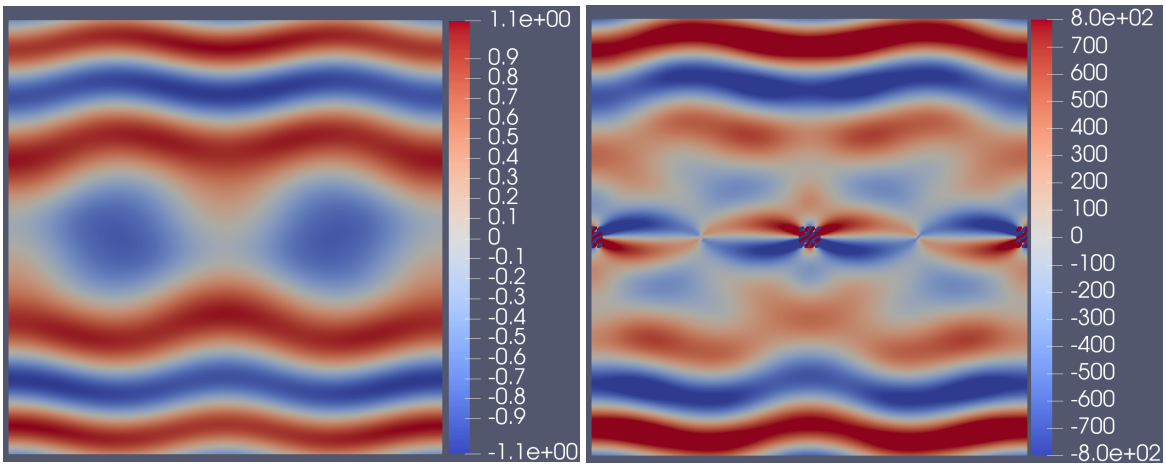
$$u = \sin(10\sin(\pi y) - 10a\cos(4\pi x)) + \varepsilon\cos(2\pi x)\sin(10\pi y), \tag{2.66}$$

which is visualised with its source term f in Figure 2.13 at $\varepsilon = 10^{-10}$. Furthermore, we visualise this for $\varepsilon = 0.1$.



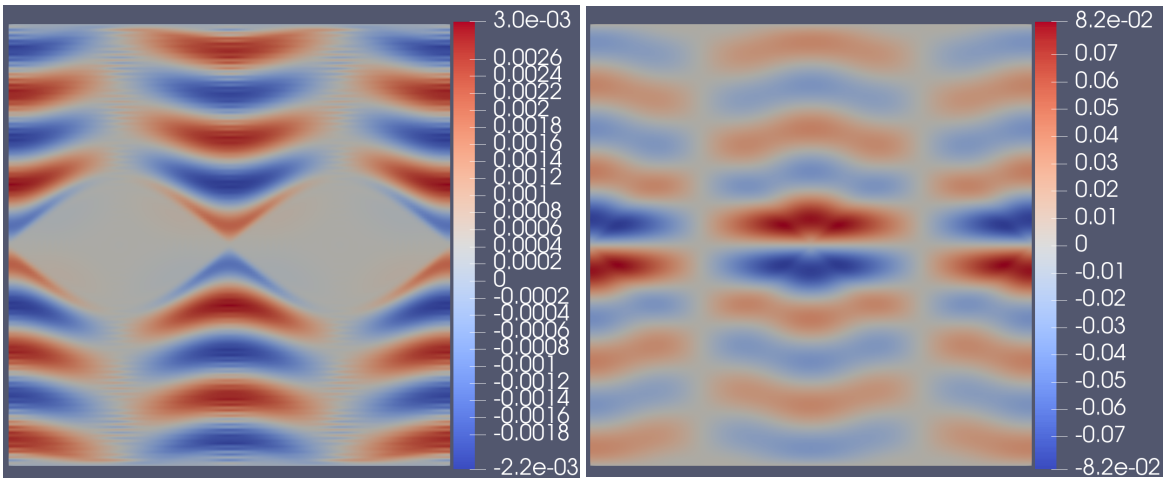
(a) Exact solution u. (b) Source term f.

Figure 2.13: Example 3 with $\varepsilon = 10^{-10}$.



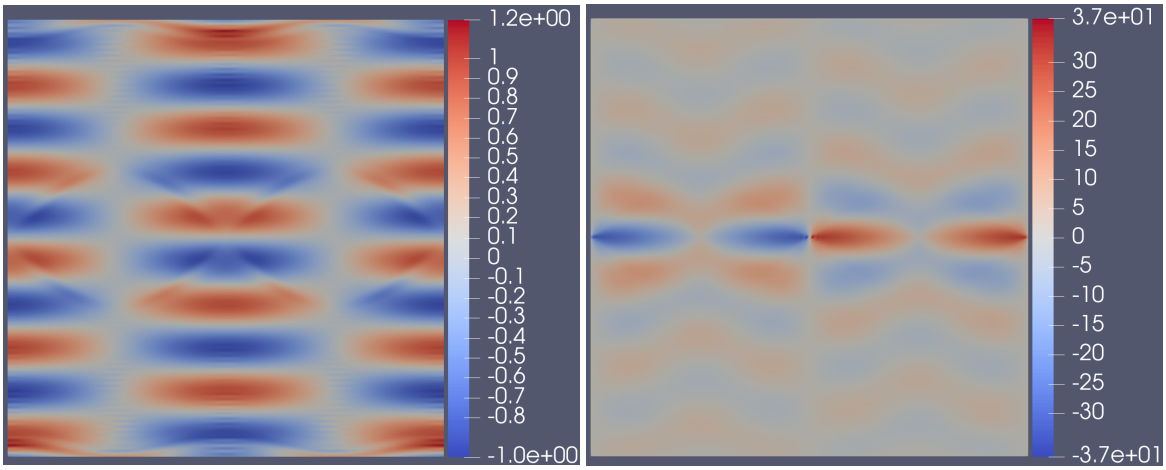
(a) Exact solution u. (b) Source term f.

Figure 2.14: Example 3 with $\varepsilon = 0.1$.



(a) L2 $= 2.6 \times 10^{-2}$ for $(MMAP\_STAB)$. (b) L2 $= 2.6 \times 10^{-2}$ for (PF_STAB).

Figure 2.15: Visualisation of Error $u_e - u_h$ for Example 3 with $\varepsilon = 10^{-10}$, CG order 2 on a 80×80 quadrilateral grid with dof= 51842.



(a) Solves q defined in equation (1.25) for $(MMAP\_STAB)$. (b) Solves q defined in equation (1.33) for (PF_STAB).

Figure 2.16: Visualisation of q defined in sections 1.2.6 and 1.3.2 for Example 3 with $\varepsilon = 10^{-10}$, CG order 2 on a 80×80 quadrilateral grid with dof= 51842.

2.7 Example 4, Magnetic Islands

We now introduce an example on a unit square where Γ_D has zero boundary conditions. This is similar to Example 3 but Example 4 satisfies $\mathbf{b} \cdot \mathbf{n} = 0$ on Γ_D this can be seen in Figure 2.17. This shows its vector field $\mathbf{b}$ on $y = 0$ and $y = 1$ is parallel to the boundary. Additionally, this has open and closed field lines. Thus on $\Omega = [0,1]^2, \Gamma_D = \{y = 0 \text{ or } y = 1\}$ and $\Gamma_N = \{x = 0 \text{ or } x = 1\}$ we solve the PDE (1.1) with the magnetic field

$$\mathbf{b} = \frac{\mathbf{B}}{|\mathbf{B}|}, \mathbf{B} = \begin{bmatrix} -\cos(\pi y) \\ 4a \sin(4\pi x) \sin(\pi y) \end{bmatrix}, \tag{2.67}$$

where the electromagnetic field $\mathbf{b}$ is shown in Figure 2.17. This vector field has streamlines

$$\sin(\pi y) \exp^{-a \cos(4\pi x)} = C, \tag{2.68}$$

with $0 \leq C \leq e^a$.

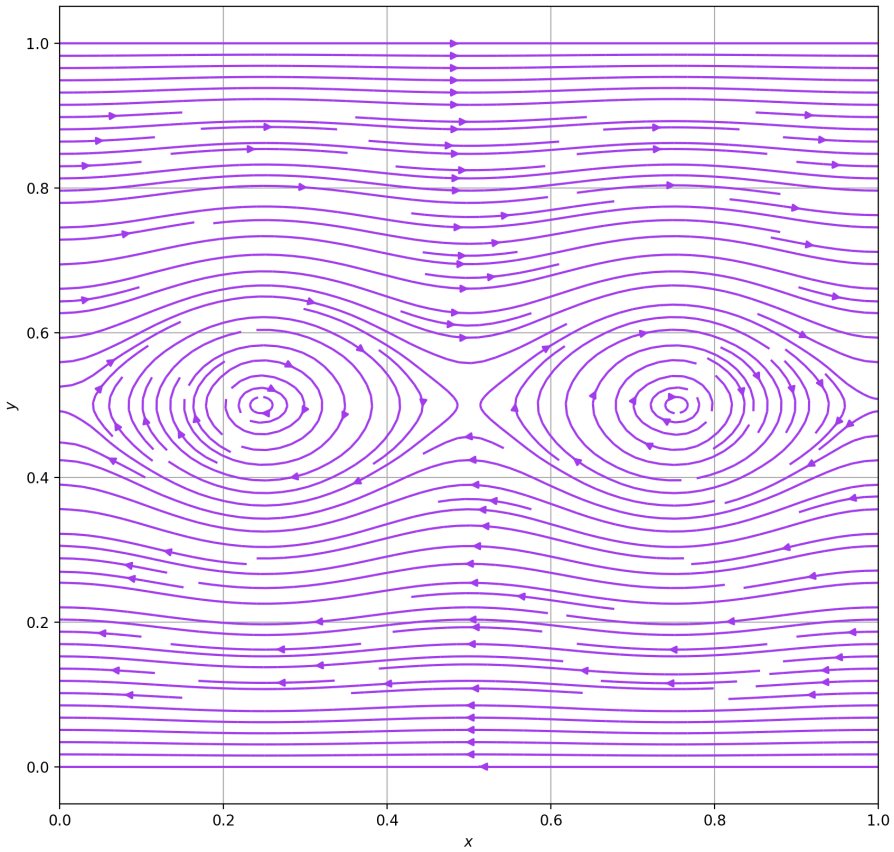
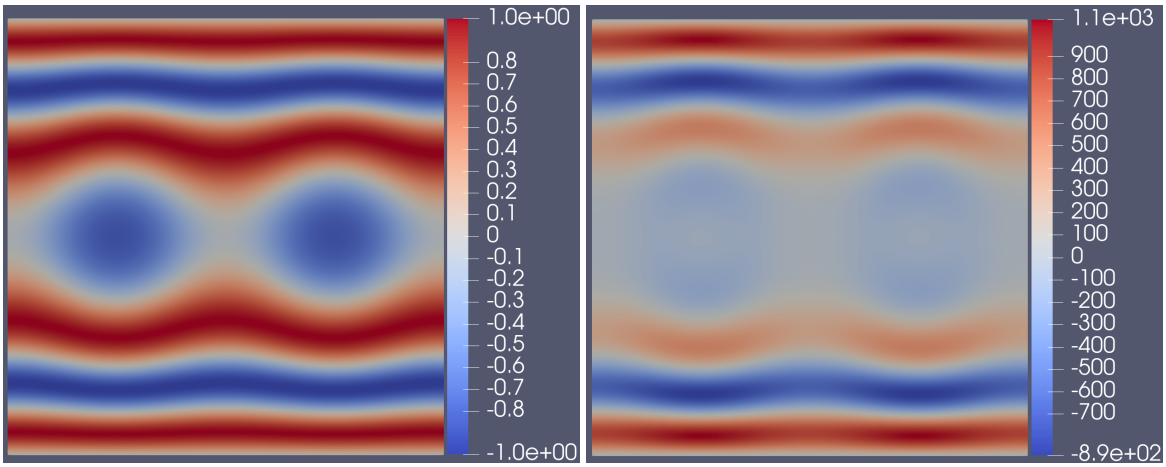


Figure 2.17: Electromagnetic field $\mathbf{b}$ for Example 4.

The exact solution is

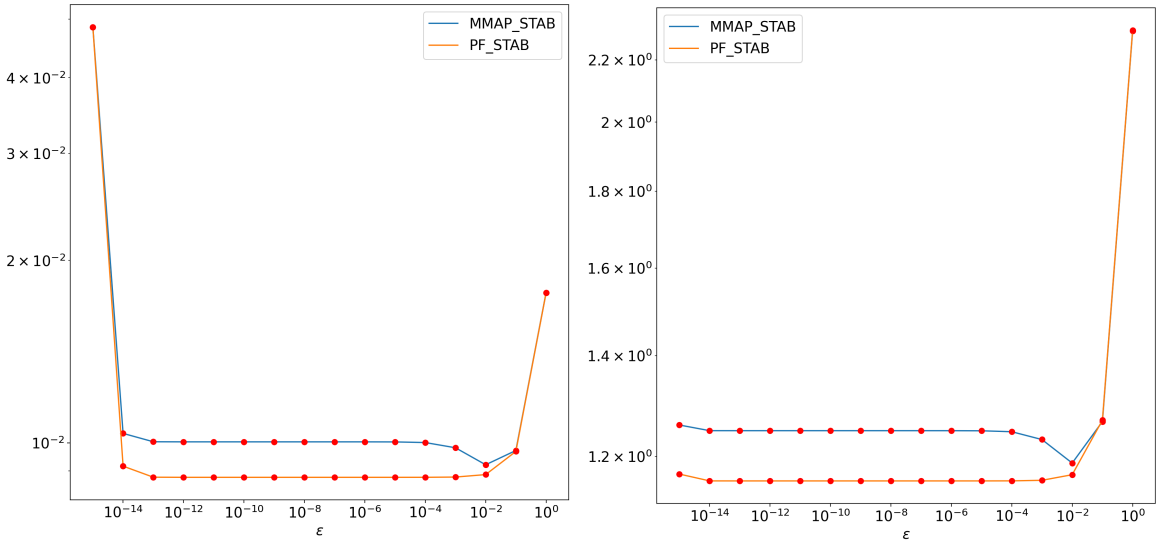
$$u = \sin(10\sin(\pi y)\exp^{-a\cos(4\pi x)}) + \varepsilon\sin(10\sin(\pi y)\exp^{-a\cos(4\pi x)}), \tag{2.69}$$

which is shown in Figure 2.18.



(a) Exact solution u. (b) Source term f.

Figure 2.18: Example 4 with $\varepsilon = 10^{-10}$.

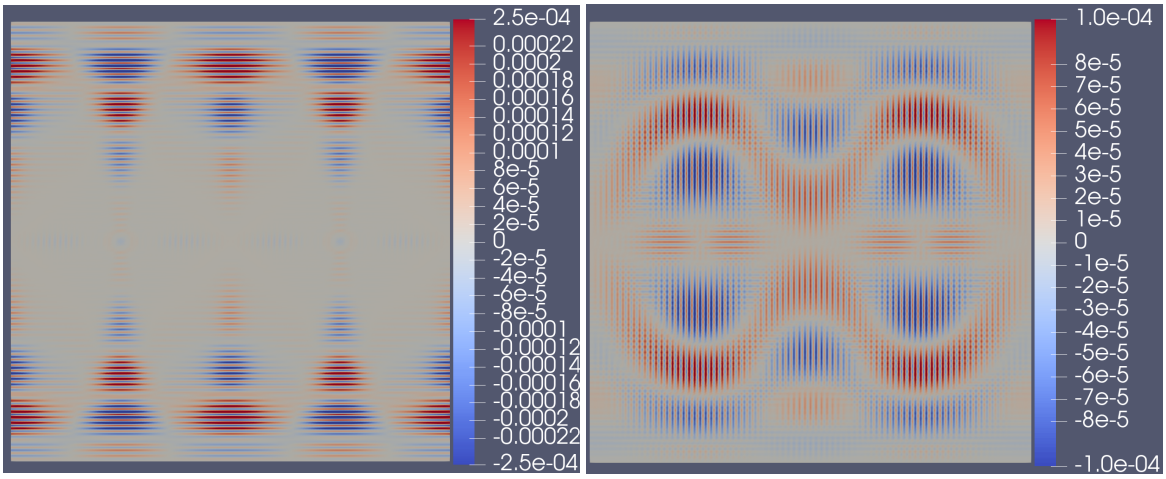


(a) L2 Error. (b) H1 Error.

Figure 2.19: Numerical solution of Example 4 for solver ($MMAP\_STAB$) and (PF_STAB) with varying ε on 20×20 quadrilateral grid with CG order 2. This has 3362 degrees of freedom.

Error		L2 Error		H1 Error	
Size	dof	PF_STAB	MMAP_STAB	PF_STAB	MMAP_STAB
10×10	882	2.33×10^{-2}	3.91×10^{-2}	1.97×10^{0}	2.16×10^{0}
20×20	3362	8.78×10^{-3}	1.00×10^{-2}	1.16×10^{0}	1.25×10^{0}
40×40	13122	1.14×10^{-3}	1.26×10^{-3}	2.98×10^{-1}	3.31×10^{-1}
80×80	51842	1.45×10^{-4}	1.56×10^{-4}	7.57×10^{-2}	8.32×10^{-2}
160×160	206082	1.84×10^{-5}	1.92×10^{-5}	1.93×10^{-2}	2.04×10^{-2}

Table 2.3: Error for varying quadrilateral grid size for Example 4 using (PF_STAB) and ($MMAP\_STAB$) with order 2 Lagrange finite element and $\varepsilon = 10^{-10}$.

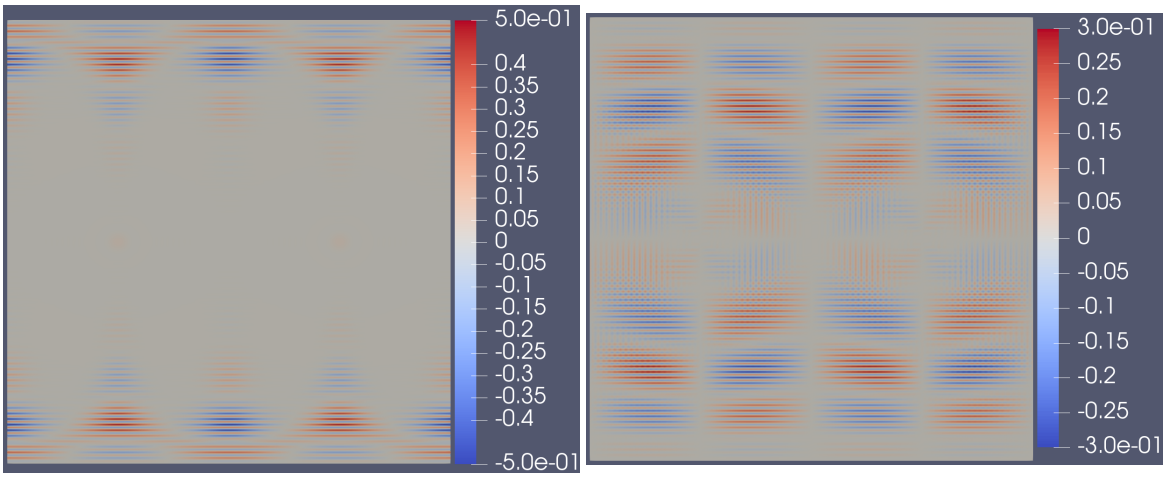


(a) L2 $= 1.56 \times 10^{-4}$ for ($MMAP\_STAB$). (b) L2 $= 1.45 \times 10^{-4}$ for (PF_STAB).

Figure 2.20: Visualisation of Error $u_e - u_h$ for Example 4 with $\varepsilon = 10^{-10}$, CG order 2 on a 80×80 quadrilateral grid with dof= 51842.

Since Example 4 is highly anisotropic the (PF_STAB) method performs slightly better than the ($MMAP\_STAB$) method. This can be seen in the numerical calculations shown in Figure 2.19. Also, from Tables 2.1 and 2.3 we can infer higher density meshes lead to better solutions we discuss how we can take advantage of this in section 3.1.3.

In Figure 2.19b and Table 2.3 it states a large H1 Error. To fix this we could consider a Hermite finite element to reduce the H1 Error. This is discussed in section 3.1.4.



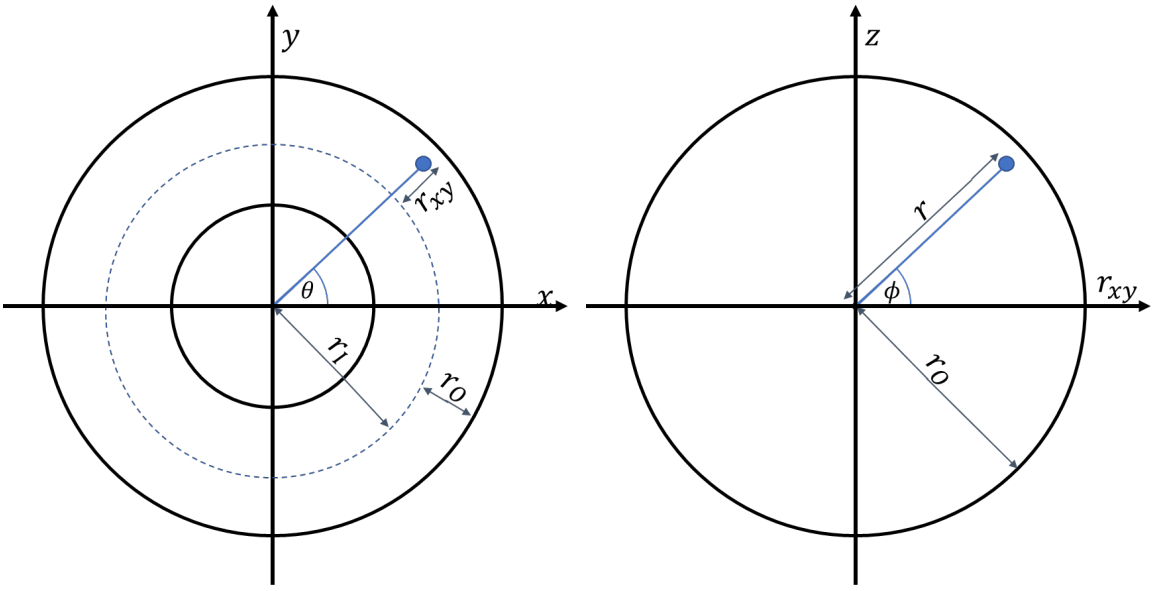
(a) Solves q defined in equation (1.25) for $(MMAP\_STAB)$.

(b) Solves q defined in equation (1.33) for (PF_STAB).

Figure 2.21: Visualisation of q defined in sections 1.2.6 and 1.3.2 for Example 4 with $\varepsilon = 10^{-10}$, CG order 2 on a 80×80 quadrilateral grid.

2.8 Formulation of Toroidal Coordinates

Here we discuss how to parameterise a torus and to find its inverted map. We use Figures 2.22a and 2.22b located below to help us derive the parameterisation.



(a) Cross section of a torus with $z = 0$.

(b) Cross section of torus.

Figure 2.22: Visual aids for the derivation of the Toroidal parameterisation.

From visual inspection of Figure 2.22a, we get

$$x = (r_I + r_{xy})\cos(\theta), \tag{2.70}$$

$$y = (r_I + r_{xy})\sin(\theta). \tag{2.71}$$

Additionally, from Figure 2.22b we get

$$r_{xy} = r\cos(\phi), \tag{2.72}$$

$$z = r\sin(\phi). \tag{2.73}$$

Thus joining the equations (2.70) to (2.73) we get the toroidal parametrisation

$$x = (r_I + r\cos(\phi))\cos(\theta), \tag{2.74}$$

$$y = (r_I + r\cos(\phi))\sin(\theta), \tag{2.75}$$

$$z = r\sin(\phi), \tag{2.76}$$

where $0 \leq r \leq r_O$ and $0 \leq \phi, \theta \leq 2\pi$. Now we calculate the inverse of this map by considering $x^2 + y^2 + z^2$.

$$x^2 + y^2 + z^2 = (r_I + r\cos(\phi))^2 + r^2\sin^2(\phi), \tag{2.77}$$

$$= r_I^2 + 2r_I r\cos(\phi) + r^2, \tag{2.78}$$

$$= r_I^2 + 2r_I\sqrt{r^2 - z^2} + r^2. \tag{2.79}$$

This is a quadratic in disguise thus after some algebraic manipulation we get 4 possible solutions

$$r = \pm\sqrt{r_I^2 + x^2 + y^2 + z^2 \pm 2r_I\sqrt{x^2 + y^2}}. \tag{2.80}$$

However, by using enforcing $r \geq 0$ we remove two solutions. We find the final solution by substitution. We use the substitution $(x, y, z) = (r_I, 0, 0)$ where $r = 0$. With $+$ we get $r = 2r_I$ and for $-$ we get $r = 0$. Thus we have

$$r = \sqrt{r_I^2 + x^2 + y^2 + z^2 - 2r_I\sqrt{x^2 + y^2}}. \tag{2.81}$$

For completeness will we calculate θ and ϕ. To get θ and ϕ we do a similar process used for calculating the argument of complex numbers. We note $r_{xy} = x^2 + y^2 - r_I$ thus we get

$$\theta = \text{atan2}(y, x), \tag{2.82}$$

$$\phi = \text{atan2}(z, x^2 + y^2 - r_I), \tag{2.83}$$

where atan2 is a common variation of the arctan function.

We use the inverse of the map because calculating the differential operators leads to large expressions. When we need to solve a PDE on a domain which can be parameterised it is usually easier to turn the source term f into Cartesian coordinates so we deal with the Cartesian differential operators.

2.9 Example 5, Torus

We now look at an example in 3D where the domain Ω is a torus with inner radius $r_I = 1$, outer radius $r_O = 0.5$ and is centred at the origin. It will have zero Dirichlet boundary conditions. For this example, we will use toroidal coordinates which are explained in section 2.8 and demonstrate how to find the inverse map. We invert back to Cartesian coordinates because the toroidal Laplacian is very complicated. Thus the definition of r is

$$r = \sqrt{r_I^2 + x^2 + y^2 + z^2 - 2r_I\sqrt{x^2 + y^2}}. \tag{2.84}$$

Where r denotes the minimum distance from the circumference of a circle centred at the origin with radius r_I and has $z = 0$. We have a magnetic field

$$\mathbf{b} = \frac{\mathbf{B}}{|\mathbf{B}|} = \begin{bmatrix} -y \\ x \\ 0 \end{bmatrix} / \sqrt{x^2 + y^2}, \mathbf{B} = \begin{bmatrix} -y \\ x \\ 0 \end{bmatrix}. \tag{2.85}$$

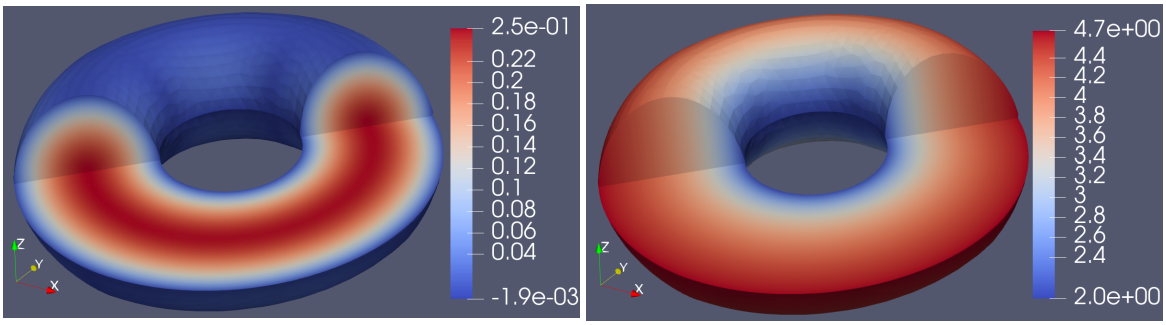
Therefore, we have the streamlines following the equation

$$x^2 + y^2 = R^2, \text{ with } r_I - \sqrt{r_O^2 - z^2} \leq R \leq r_I + \sqrt{r_O^2 - z^2}. \tag{2.86}$$

In this case $1 - \sqrt{0.25 - z^2} \leq R \leq 1 + \sqrt{0.25 + z^2}$. Also, we will have the exact solution

$$u = \begin{cases} (1+\varepsilon)((0.5)^2 - r), \\ (1+\varepsilon)\left(0.25 - \sqrt{r_I^2 + x^2 + y^2 + z^2 - 2r_I\sqrt{x^2 + y^2}}\right). \end{cases} \tag{2.87}$$

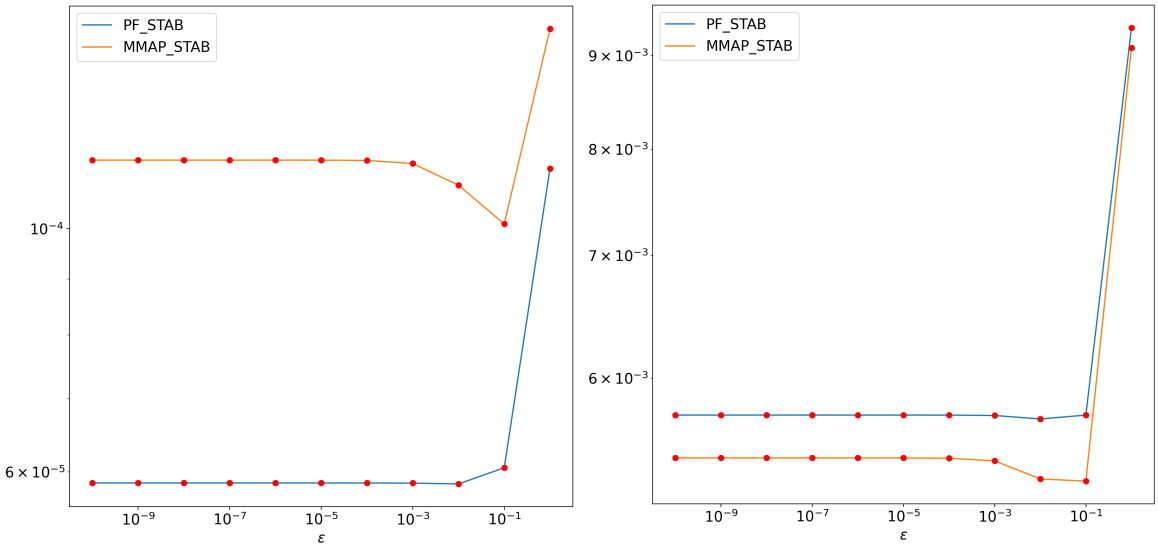
This u is displayed in Figure 2.23 with its source term f and $\varepsilon = 10^{-10}$.



(a) Exact solution u.

(b) Source term f.

Figure 2.23: Example 5 with $\varepsilon = 10^{-10}$.



(a) L2 Error.

(b) H1 Error.

Figure 2.24: Numerical solution of Example 5 for solver ($MMAP\_STAB$) and (PF_STAB) with varying ε on torus mesh with CG order 2. This has 69976 degrees of freedom.

The numerical calculations in Figure 2.24 show the (PF_STAB) method and the ($MMAP\_STAB$) method work with closed streamlines in $3D$. However, since the finite element is a tetrahedron and involves volume integrals it takes longer to calculate the entries to the linear system than a domain in $2D$.

Chapter 3

Conclusion and Future Work

3.1 Methods For Investigation

3.1.1 Line Integration

The paper [12] states we can use line integration to solve PDE (1.1). The idea of line integration is to use the fact that the solution should be approximately constant along streamlines for $\varepsilon \ll 1$. Therefore, we only have to calculate the PDE on Γ_{in}. We demonstrate the idea of this method for Example 1 when $\alpha = 0$. Thus, the magnetic field $\mathbf{b} = [1, 0]$. When substituting this $\mathbf{b}$ into (1.1) we get

$$\begin{cases} \varepsilon^{-1} u_{xx} + u_{yy} = f(x, y), & \in \Omega, \\ u_x = 0, & \text{on } \Gamma_N, \\ u = 0, & \text{on } \Gamma_D. \end{cases} \tag{3.1}$$

Now we take line integrals along the vector field and take the limit of $\varepsilon \to 0$ in (3.1) to get

$$\begin{cases} u_{xx} = 0, & \in \Omega, \\ u_x = 0, & \text{on } x = 0, \\ -\int_0^1 u_{yy} dx = \int_0^1 f(x, y) dx, & \forall y \in [0, 1], \\ u = 0, & \text{on } y = 0 \text{ or } y = 1. \end{cases} \tag{3.2}$$

This problem can be solved. From the first two equations in (3.2) we get the solution for u is constant along $\mathbf{b}$. Thus u is independent of x. From this fact and the last

two equations we get

$$\begin{cases} -u_{yy} = \int_0^1 f(x,y)dx, & \text{on } x = 0, \\ u = 0, & \text{on } y = 0 \text{ or } y = 1. \end{cases} \tag{3.3}$$

Now we demonstrate this works. For Example 1 with $\alpha = 0$ we get

$$f(x,y) = \sin(\pi y)\pi^2(1 + (4+\varepsilon)\cos(2\pi x)). \tag{3.4}$$

Thus after some integration, we get

$$u_{yy} = -\int_0^1 f(x,y)dx = -\int_0^1 \sin(\pi y)\pi^2(1+(4+\varepsilon)\cos(2\pi x))dx = -\pi^2 sin(\pi y). \tag{3.5}$$

Which implies $u = \sin(\pi y)$, this is missing the ε order term from (2.60) because we have solved the limit problem. Therefore we get the value of u as $\varepsilon \to 0$. Additionally, the paper [12] describes how to use this method for a vector field of the form $\mathbf{b} = [\cos(\theta), \sin(\theta)]$, then they parameterise θ to get an arbitrary vector field. Also, as Example 1 is a popular example to use to demonstrate methods for solving PDE (1.1), they cover $(\alpha = 0)$ and $(\alpha = 2, m = 1)$ in their numerical demonstrations. Furthermore, our (PF) and $(MMAP)$ seem to have smaller L2 Error for the same examples, this is hard to compare because they used a finite difference method to numerically solve their PDE.

However, it is not clear how a stabilisation technique can be used to solve vector fields with closed field lines. At the end of their paper, they stated their future work would be to solve this problem. One potential solution could be to take a point on each streamline and then join the points together for streamlines that touch. Then solve the reduced dimension PDE.

3.1.2 Mapping of f

The idea behind this method is given f for PDE (1.1) when $\varepsilon = 10^{-15}$. We use this f and solve the PDE for $\varepsilon = 10^{-1}$, this reduces the anisotropic strength and should reduce the error. We use the code below to quickly test this hypothesis on Example 4 using the $(MMAP\_STAB)$ method.

```
E4 = Example_4(eps = 1e-15, Size = 40)
epss = 10**(np.arange(-15,1,1,dtype=float))
u_e = E4[5]
for eps in epss:
    u_h = Solve_MMAP_STAB_Method(*E4, Order = 2, eps = eps)
    print("eps: "+str(eps) +", L2: %.5f" % norm(u_e-u_h, "L2"))
    print("eps: "+str(eps) +", H1: %.5f" % norm(u_e-u_h, "H1"))
```

Table 3.1 contains some output of the code. It shows doing this method slightly reduces H1 Error but slightly increases L2 Error. Therefore, this suggests this technique does not work. Also, by looking at our numerical demonstrations of the ($MMAP\_STAB$) method shown in Figure 2.19 it can be seen varying ε does not change the error. This shows that for the ($MMAP\_STAB$) method the difficulty comes from having a vector field $\mathbf{b}$ that is highly anisotropic.

ε	L2 Error	H1 Error
10^{-15}	0.02880	0.34581
10^{-10}	0.02880	0.34581
10^{-5}	0.02880	0.34564
10^{-4}	0.02880	0.34415
10^{-3}	0.02880	0.31149
10^{-2}	0.02880	0.31481
10^{-1}	0.02884	0.31149
10^{0}	0.02904	0.31333

Table 3.1: Error for solving with ($MMAP\_STAB$) with ε on a 40×40 quadrilateral grid with an order 2 Lagrange Element (dof= 13122) using f from $\varepsilon = 10^{-15}$.

Future work would involve finding a mapping for the f generated by the PDE (1.1) when $\varepsilon = 10^{-15}$ and the f generated by the PDE when $\varepsilon = 10^{-1}$. This would lead to a reduced error because solving Example 4 with the ($MMAP\_STAB$) method on a 40×40 quadrilateral grid with order 2 Lagrange elements with $\varepsilon = 0.1$ we get an L2 Error of 0.001249. This is an order of magnitude lower than the L2 Error of 0.02880 from Table 3.1 at $\varepsilon = 10^{-15}$.

3.1.3 More Finite Elements

Future work would involve looking at different finite elements. Currently, our numerical calculations have been done with an order 2 Lagrange finite element. This means our solutions will be continuous but the solutions derivative will be discontinuous. This leads to the H1 error being large for the highly anisotropic examples. We can counter this by having a Hermite finite element which will give a continuous solution and the derivative will be continuous.

We have the basis functions for a Hermite finite element stated at [11]. Also, they

can be calculated using a similar method discussed in section 2.2. In Figure 3.1 we show the basis functions for the Hermite finite element on a unit interval.

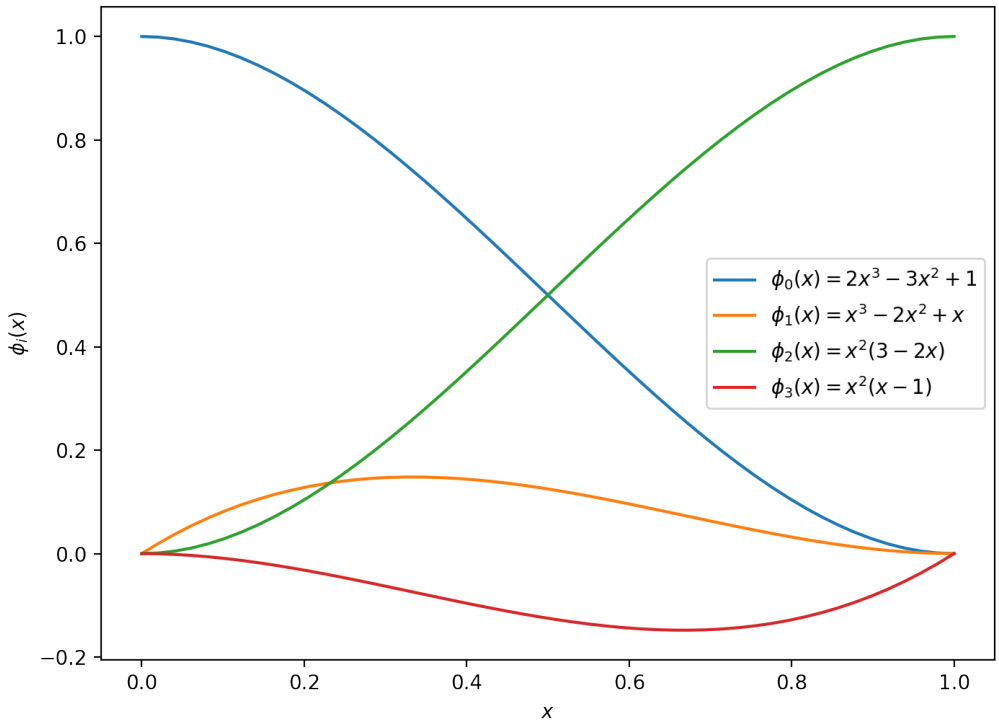


Figure 3.1: Hermite basis functions on interval $[0, 1]$.

3.1.4 Parallel Implementation

A parallel implementation would speed up the calculation of the solution. Thus in the same amount of time as the sequential Firedrake implementation, a higher density mesh can be used and this leads to a solution with less error as shown in Tables 2.1 and 2.3.

When using a local formulation it is trivial to calculate and add the contribution from each finite element to the global matrix in parallel.

What is not so clear is how to solve a linear system in parallel. First, we note that a matrix calculated from a finite element problem is sparse. In [13] it discusses how to implement a parallel GMRES (generalised minimal residual method) algorithm to solve sparse linear systems. Also, in [14] it states in its abstract that a cluster of 12 GPUs is 8 times faster than a cluster of 12 CPUs for solving a sparse linear system.

We now demonstrate a parallel GMRES algorithm with restarts. For our tests we use the sparse linear system thermal1 [15] which is shown in Figure 3.2. We denote $\mathbb{K}$ as the matrix and $\mathbf{d}$ as the vector in the data set thermal1. This symmetric matrix has 82654 rows and 82654 columns with $574,458$ non-zero entries. The matrix data contains the linear system from a finite element method on a steady-state thermal problem. For our CPU execution, we use the GMRES function from SciPy [16] and our GPU execution uses the GMRES function from [17] which provides a high-level interface for linear algebra algorithms on sparse matrices. It requires CUDA [18] to work.

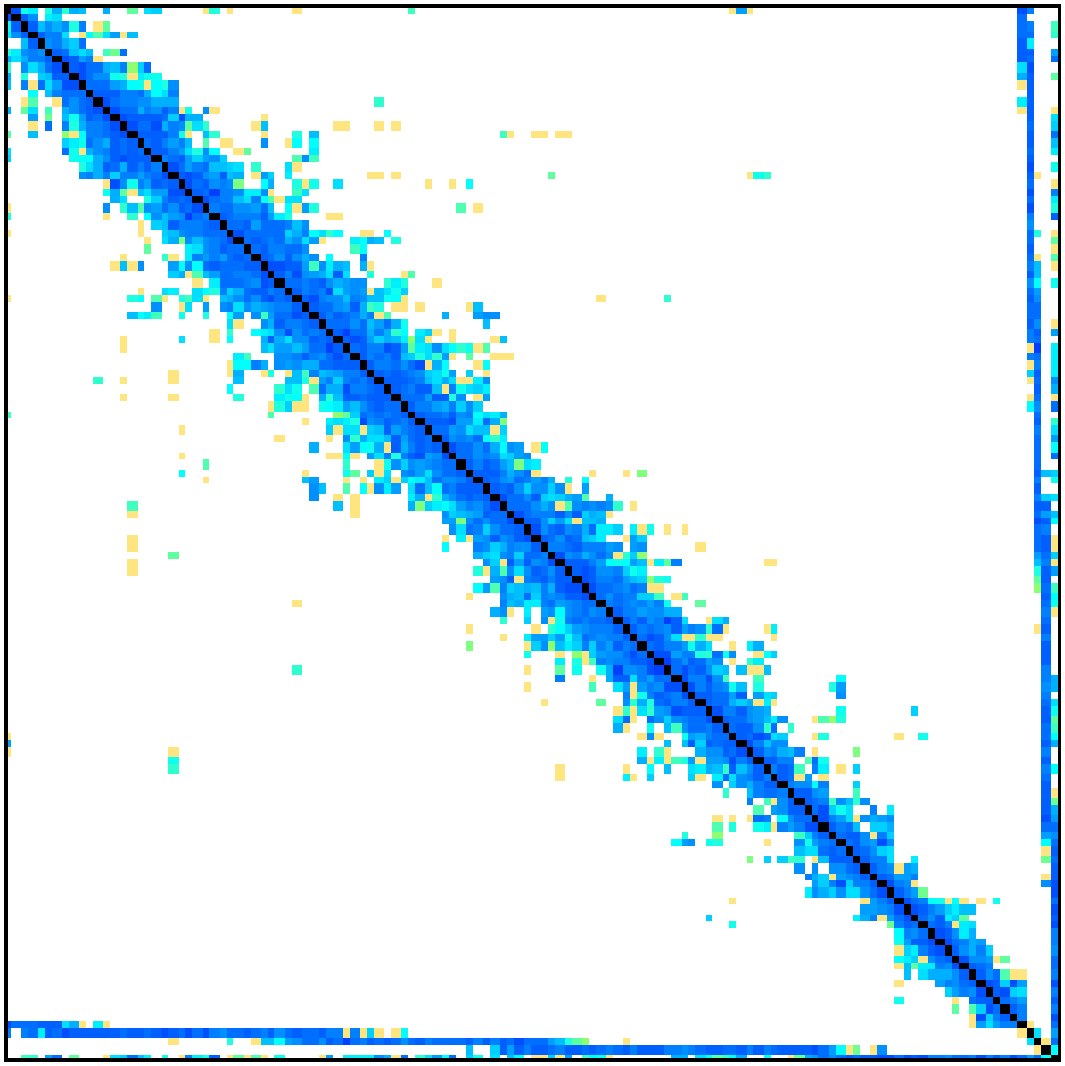


Figure 3.2: Visual representation of $\mathbb{K}$ from data set thermal1 [15]. Darker squares represent a higher density of non-zero entries.

Now we run our parallel CPU implementation on an 8 Core i7-11800H Processor [19] with code in Appendix F. Also, we run our GPU code on a NVIDIA GeForce RTX 3050 Ti Laptop GPU [20] with code in Appendix E. In both cases we represent the matrix in COO format [21] and use no preconditioner. The results are shown in Table 3.2.

	Time Seconds		Residual $\lvert\mathbb{K}\mathbf{x} = \mathbf{d}\rvert$	
tol	Python SciPy	CUDA CUSP	Python SciPy	CUDA CUSP
10^{-2}	0.0976	0.0766	$1.681E-01$	$1.681E-01$
10^{-3}	0.6493	0.4654	$1.711E-02$	$1.706E-02$
10^{-4}	9.6704	5.5791	$1.715E-03$	$1.715E-03$
10^{-5}	45.5871	30.7765	$1.715E-04$	$1.715E-04$

Table 3.2: Comparison of solving linear system created by the data set thermal1 [15] using SciPy [16] GMRES 8 Core CPU implementation and CUDA CUSP [17] GMRES GPU implementation.

From Table 3.2 the GPU implementation is faster. However, when running the CPU implementation, it used 100% of the CPU cores. The GPU implementation achieved a maximum of 30% of its computational power. This could be because SciPy is a popular library and thus constantly updated to optimise its performance and take advantage of the latest hardware. While CUSP was last updated in 2018 and required file tweaks to make it compatible with the latest version of CUDA [18].

However, the most likely reason is the matrix is not large enough to take advantage of the 2560 cores on this GPU. For larger systems, the GPU implementation will perform better than its CPU implementation.

Also, the gaming industry is driving hardware development for GPUs. So GPUs are going to get more powerful. Therefore, future work would involve finding better parallel algorithms to solve sparse linear systems and better communication protocols between cores. Then it will be coded in CUDA for simply implementation by third parties.

Currently, Firedrake does not support calculations on the GPU this can be seen by the open issue on GitHub located at [22]. Other potential future work would involve helping integrate parallel algorithms for Firedrake.

3.2 Conclusion

In this thesis, we have discussed existing methods to solve anisotropic diffusion problems. We proposed a new method (PF_STAB). When the problem is highly anisotropic the (PF_STAB) method performs slightly better than the best existing method ($MMAP\_STAB$). Also, these methods do not require grid alignment. However, it is unclear what value to assign σ.

Moreover, methods (PF) and ($MMAP$) do not work on closed field lines and we have to find Γ_{in}.But we found for the ($MMAP$) method we can use a modified version of $\mathcal{L}$ to get a reasonable error on closed field lines. More research is needed to know if the selected points on the streamlines have to form a continuous line.

The (SP) method is the best for solving $\varepsilon \approx 1$ and it contains one PDE so there are fewer degrees of freedom than other methods. However, the method fails to produce a sufficiently small error for $\varepsilon \ll 1$ as it is not well-posed.

We found no reason to use the (AP) method when the ($MMAP$) method produces the same error for fewer degrees of freedom. Additionally, when $\varepsilon \ll 1$ discretisations from ($MMAP$) and (LM) are almost identical. However, the (LM) method fails to calculate an acceptable error for larger ε. Thus there is no reason to use the (LM) method. The same reasoning is used for the stabilisation variants.

We have shown it is possible to solve partial differential equations via the finite element method on a GPU cluster. In a sequential implementation, assembling the linear system of equations takes longer than finding the solution to the linear system, as integration over finite elements takes time. However, with a parallel implementation, the bottlenecks become data communication and processes that can not be run in parallel. As assembling the linear system in parallel is straightforward funding and research should go into finding parallel linear system solvers that spread the computational load more efficiently. Then we can use robust discretisations (PF_STAB) and ($MMAP\_STAB$) on a high-density mesh to get an accurate solution to a highly anisotropic diffusion problem in a reasonable time.

Further research can be done into finding better discretisations but it is unclear if significantly better methods can be found. With research into parallel implementation, you will also get the benefit of other industries driving development for better parallel hardware.

Thus future work would be focused on using heterogeneous computing techniques and fully utilising the hardware available to solve the PDE.

Finally, the code used for the methods, the examples and generating data can be

found on GitHub at [1]. This means results can be easily reproduced. To generate the data in the graphs and tables we used the functions in "makePics.py" located at [1].

Bibliography

[1] *Fusion Power Repository*. (Last Visited 14/08/2022). 2022. URL: https://github.com/Hitthesurf/Fusion_Power.

[2] Fabrice Deluzet and Jacek Narski. "A Two Field Iterated Asymptotic-Preserving Method for Highly Anisotropic Elliptic Equations". In: *Multiscale Model. Simul.* 17.1 (2019), 434–459. ISSN: 1540-3459. DOI: 10.1137/17M115205X. URL: https://doi.org/10.1137/17M115205X.

[3] Pierre Degond et al. "Duality-based Asymptotic-Preserving method for highly anisotropic diffusion equations". In: *Communications in Mathematical Sciences* 10 (Aug. 2010). DOI: 10.4310/CMS.2012.v10.n1.a2.

[4] *Firedrake*. (Last Visited 14/08/2022). 2022. URL: https://firedrakeproject.org/.

[5] Pierre Degond et al. "An asymptotic-preserving method for highly anisotropic elliptic equations based on a Micro–Macro decomposition". In: *Journal of Computational Physics* 231.7 (2012), pp. 2724–2740. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2011.11.040. URL: https://www.sciencedirect.com/science/article/pii/S0021999111006966.

[6] Jacek Narski and Maurizio Ottaviani. "Asymptotic Preserving scheme for strongly anisotropic parabolic equations for arbitrary anisotropy direction". In: *Computer Physics Communications* 185.12 (2014), pp. 3189–3203. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2014.08.018. URL: https://www.sciencedirect.com/science/article/pii/S0010465514002999.

[7] *FEniCSx*. (Last Visited 14/08/2022). 2022. URL: https://fenicsproject.org/.

[8] *ParaView*. [Online; accessed 26-August-2022]. URL: https://www.paraview.org/.

[9] *Ipyvolume*. [Online; accessed 26-August-2022]. URL: https://ipyvolume.readthedocs.io/en/latest/.

[10] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[11] The DefElement contributors. *DefElement: an encyclopedia of finite element definitions*. https://defelement.com. [Online; accessed 14-August-2022]. 2022.

[12] Min Tang and Yihong Wang. "An asymptotic preserving method for strongly anisotropic diffusion equations based on field line integration". In: *Journal of Computational Physics* 330 (2017), pp. 735–748. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2016.10.062. URL: https://www.sciencedirect.com/science/article/pii/S0021999116305708.

[13] Morgan Görtz. *Parallelization Of The GMRES Method*. [Online; accessed 14-August-2022]. URL: https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8889583&fileOId=8890947.

[14] Jacques M. Bahi, Raphaël Couturier, and Lilia Ziane Khodja. "Parallel GMRES Implementation for Solving Sparse Linear Systems on GPU Clusters". In: *Proceedings of the 19th High Performance Computing Symposia*. HPC '11. Boston, Massachusetts: Society for Computer Simulation International, 2011, 12–19.

[15] D. Schmid. *Matrix: Schmid/thermal1*. [Online; accessed 14-August-2022]. 2006. URL: https://www.cise.ufl.edu/research/sparse/matrices/Schmid/thermal1.html.

[16] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[17] Steven Dalton et al. *Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations*. Version 0.5.0. 2014. URL: http://cusplibrary.github.io/.

[18] *NVIDIA CUDA*. [Online; accessed 26-August-2022]. URL: https://developer.nvidia.com/cuda-toolkit.

[19] *Intel® Core™ i7-11800H Processor*. [Online; accessed 26-August-2022]. URL: https://www.intel.com/content/www/us/en/products/sku/213803/intel-core-i711800h-processor-24m-cache-up-to-4-60-ghz/specifications.html.

[20] *NVIDIA GeForce RTX 3050 Ti Mobile*. [Online; accessed 26-August-2022]. URL: https://www.techpowerup.com/gpu-specs/geforce-rtx-3050-ti-mobile.c3778.

[21] Hoang-Vu Dang and Bertil Schmidt. "The Sliced COO Format for Sparse Matrix-Vector Multiplication on CUDA-enabled GPUs". In: *Procedia Computer Science* 9 (2012). Proceedings of the International Conference on Computational Science, ICCS 2012, pp. 57–66. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2012.04.007. URL: https://www.sciencedirect.com/science/article/pii/S1877050912001287.

[22] *GPU support for Firedrake 1605*. [Online; accessed 26-August-2022]. URL: https://github.com/firedrakeproject/firedrake/pull/1605.

Appendix A

Example of Using Our Code

The code below shows how to use our code to calculate a solution to Example 1 with $\alpha = 2$ and $m = 10$ in Python with Firedrake [4].

```
from Solvers import *
from Examples import *
from firedrake import *

E1 = Example_1(al = 2, m=10, eps = 1e-10, Size = 160)
u_e = E1[5]
u_h = Solve_MMAP_Method(*E1, Order = 2, eps = 1e-10, save = True)
#Save creates a PVD file with u_h, f, ... in output_graph/
L2_error = norm(u_h - u_e, "L2")
H1_error = norm(u_h - u_e, "H1")
print("MMAP eps 1e-10 ||u - u_exact||_L2: %.1e" % L2_error)
print("MMAP eps 1e-10 ||u - u_exact||_H1: %.1e" % H1_error)
```

This will output the following text to the command line

```
MMAP, Z.dim() = 206082, Order = 2, eps = 1e-10.
MMAP eps 1e-10 ||u - u_exact||_L2: 6.4e-05
MMAP eps 1e-10 ||u - u_exact||_H1: 1.1e-02
```

From this code, it is trivial to change the example and method used.

Appendix B

Calculation of Streamlines

Here we present the method we used to calculate the streamlines for the vector field **b** (2.58) in Example 1. A similar approach can be used to calculate the streamlines for the other examples covered in this paper. First, we have

$$\frac{dy}{dx} = \frac{\dot{y}}{\dot{x}} = \frac{\alpha m\pi(y^2 - y)\sin(m\pi x)}{\alpha(2y-1)\cos(m\pi x) + \pi}, \tag{B.1}$$

after some algebraic manipulation, we get

$$df = (\alpha m\pi(y^2 - y)\sin(m\pi x))dx + (\alpha(2y-1)\cos(m\pi x) + \pi)dy = 0. \tag{B.2}$$

This is in exact form so we must solve the following equations

$$\frac{\partial f}{\partial x} = \alpha m\pi(y^2 - y)\sin(m\pi x), \tag{B.3}$$

$$\frac{\partial f}{\partial y} = (\alpha(2y-1)\cos(m\pi x) + \pi). \tag{B.4}$$

Thus the streamlines follow the function

$$\pi y + \alpha(y^2 - y)\cos(m\pi x) = C. \tag{B.5}$$

Appendix C

Code File: Solvers.py

```
#Solvers
from firedrake import *
import numpy as np

def SAVE(to_save, f, b, mesh, u_h, u_e, Order, name, Extra):
    path_name_file = "output_graph/"+name+"_solution.pvd"
    U = FunctionSpace(mesh, "CG", Order)
    vecU = VectorFunctionSpace(mesh, "CG", Order)
    b_h = Function(vecU).interpolate(b)
    f_h = Function(U).interpolate(f)
    f_h.rename("f_h")
    b_h.rename("b_h")
    to_save.append(f_h)
    to_save.append(b_h)
    if Extra:
        u_e_h = Function(U).interpolate(u_e)
        the_error = Function(U).interpolate(u_h-u_e_h)
        grad_u_e_h = Function(vecU).interpolate(grad(u_h))
        u_e_h.rename("u_e_h")
        the_error.rename("the_error")
        grad_u_e_h.rename("grad_u_e_h")
        to_save.append(u_e_h)
        to_save.append(the_error)
        to_save.append(grad_u_e_h)
    File(path_name_file).write(*to_save)

def Solve_MMAP_Method(f, b, mesh, dOmegaD, dOmegaIN, dOmegaDVal=0,
                      A_para = 1, A_perp = None, Order = 1,
                      eps = 1e-15, save = False,
```

```
                    BC_As_Exact = True):
    V = FunctionSpace(mesh, "CG", Order)
    L = FunctionSpace(mesh, "CG", Order)
    Z = V*L

    print(f"MMAP, Z.dim() = {Z.dim()}, Order = {Order},"+
          f" eps = {eps}.")
    if (eps==0):
        print("LM, As eps=0.")


    I = Identity(mesh.geometric_dimension())
    if A_perp == None:
        A_perp = I

    #Set boundary conditions
    bc0 = DirichletBC(Z.sub(0), dOmegaDVal, dOmegaD)
    bc1 = DirichletBC(Z.sub(1), dOmegaDVal, dOmegaD)
    bcs = [bc0, bc1]
    if (len(dOmegaIN)>0):
        bc2 = DirichletBC(Z.sub(1), 0, dOmegaIN)
        bcs.append(bc2)

    grad_para = lambda alpha: dot(outer(b,b), grad(alpha))
    grad_perp = lambda alpha: dot(I-outer(b,b), grad(alpha))
    a_perp = lambda alpha, beta: inner(A_perp*grad_perp(alpha),
                                       grad_perp(beta))
    a_para = lambda alpha, beta: inner(A_para*grad_para(alpha),
                                       grad_para(beta))

    z = Function(Z)
    u, q = split(z)
    u_, q_ = split(TestFunction(Z))

    F = (
        + a_perp(u, u_)*dx
        + a_para(q, u_)*dx
        - inner(f, u_)*dx
        + a_para(u, q_)*dx
        )

    if eps>0:
        F = (
```

```
            + a_perp(u, u_)*dx
            + a_para(q, u_)*dx
            - inner(f, u_)*dx
            + a_para(u, q_)*dx
            - eps*a_para(q, q_)*dx
            )

    solve(F==0,z,bcs)
    (u_h, q_h) = z.split()
    u_h.rename("u_h")
    q_h.rename("q_h")

    #Save Data
    if save:
        to_save = [u_h, q_h]
        name = "MMAP"
        if eps == 0:
            name = "LM"
        SAVE(to_save, f, b, mesh, u_h, dOmegaDVal,
             Order, name, BC_As_Exact)
    return u_h



def Solve_MMAP_STAB_Method(f, b, mesh, dOmegaD, dOmegaIN,
                           dOmegaDVal=0, A_para = 1, A_perp = None,
                           Order = 1, eps = 1e-15, sigma = 0.1,
                           save = False, BC_As_Exact = True):
    V = FunctionSpace(mesh, "CG", Order)
    Z = V*V

    print(f"MMAP_STAB, Z.dim() = {Z.dim()}, Order = {Order},"+
          f" eps = {eps}.")
    if (eps==0):
        print("LM_STAB, As eps=0.")
    I = Identity(mesh.geometric_dimension())
    if A_perp == None:
        A_perp = I

    #Set boundary conditions
    bc0 = DirichletBC(Z.sub(0), dOmegaDVal, dOmegaD)
    bc1 = DirichletBC(Z.sub(1), dOmegaDVal, dOmegaD)
    bcs = [bc0, bc1]
```

```
116
117     grad_para = lambda alpha: dot(outer(b,b), grad(alpha))
118     grad_perp = lambda alpha: dot(I-outer(b,b), grad(alpha))
119     a_perp = lambda alpha, beta: inner(A_perp*grad_perp(alpha),
120                                        grad_perp(beta))
121     a_para = lambda alpha, beta: inner(A_para*grad_para(alpha),
122                                        grad_para(beta))
123
124     z = Function(Z)
125     u, q = split(z)
126     u_, q_ = split(TestFunction(Z))
127
128
129
130     F = (
131         + a_perp(u, u_)*dx
132         + a_para(q, u_)*dx
133         - inner(f, u_)*dx
134         + a_para(u, q_)*dx
135         - sigma*inner(q,q_)*dx
136         )
137
138     if eps>0:
139         F = (
140             + a_perp(u, u_)*dx
141             + a_para(q, u_)*dx
142             - inner(f, u_)*dx
143             + a_para(u, q_)*dx
144             - eps*a_para(q, q_)*dx
145             - sigma*inner(q,q_)*dx
146             )
147
148     solve(F==0,z,bcs)
149     (u_h, q_h) = z.split()
150     u_h.rename("u_h")
151     q_h.rename("q_h")
152     if save:
153         to_save = [u_h, q_h]
154         name = "MMAP_STAB"
155         if eps == 0:
156             name = "LM_STAB"
157         SAVE(to_save, f, b, mesh, u_h, dOmegaDVal,
158              Order, name, BC_As_Exact)
```

```
159     return u_h
160
161
162 def Solve_Limit_Method(f, b, mesh, dOmegaD, dOmegaIN, dOmegaDVal=0,
163                        A_para = 1, A_perp = None, Order = 1,
164                        eps = None, save = False,
165                        BC_As_Exact = True):
166     return Solve_MMAP_Method(f, b, mesh, dOmegaD, dOmegaIN,
167                              dOmegaDVal=dOmegaDVal,
168                              A_para = A_para, A_perp = A_perp,
169                              Order = Order, eps = 0, save = save,
170                              BC_As_Exact = BC_As_Exact)
171
172 def Solve_Limit_STAB_Method(f, b, mesh, dOmegaD, dOmegaIN,
173                             dOmegaDVal=0, A_para = 1,
174                             A_perp = None, Order = 1, eps = None,
175                             sigma = 0.1, save = False,
176                             BC_As_Exact = True):
177     return Solve_MMAP_STAB_Method(f, b, mesh, dOmegaD, dOmegaIN,
178                                   dOmegaDVal=dOmegaDVal,
179                                   A_para = A_para, A_perp = A_perp,
180                                   Order = Order, eps = 0,
181                                   sigma = sigma, save = save,
182                                   BC_As_Exact = BC_As_Exact)
183
184
185
186 def Solve_Singular_Pert_Method(f, b, mesh, dOmegaD, dOmegaIN,
187                                dOmegaDVal=0, A_para = 1,
188                                A_perp = None, Order = 1,
189                                eps = 1e-15, save = False,
190                                BC_As_Exact = True):
191     U = FunctionSpace(mesh, "CG", Order)
192
193     print(f"SP, Z.dim() = {U.dim()}, Order = {Order},"+
194           f" eps = {eps}.")
195     I = Identity(mesh.geometric_dimension())
196     if A_perp == None:
197         A_perp = I
198
199     #Set boundary conditions
200     bc0 = DirichletBC(U, dOmegaDVal, dOmegaD)
201     bcs = [bc0]
```

```
202
203     grad_para = lambda alpha: dot(outer(b,b), grad(alpha))
204     grad_perp = lambda alpha: dot(I-outer(b,b), grad(alpha))
205     a_perp = lambda alpha, beta: inner(A_perp*grad_perp(alpha),
206                                        grad_perp(beta))
207     a_para = lambda alpha, beta: inner(A_para*grad_para(alpha),
208                                        grad_para(beta))
209
210     u = TrialFunction(U)
211     psi = TestFunction(U)
212     u_h = Function(U)
213
214     a = ((1/eps)*a_para(u,psi) + a_perp(u, psi))*dx
215     F = inner(f, psi)*dx
216
217     solve(a==F, u_h, bcs)
218     u_h.rename("u_h")
219
220     #Save Data
221     if save:
222         to_save = [u_h]
223         SAVE(to_save, f, b, mesh, u_h, dOmegaDVal,
224              Order, "SP", BC_As_Exact)
225     return u_h
226
227 def Solve_AP_Method(f, b, mesh, dOmegaD, dOmegaIN, dOmegaDVal=0,
228                     A_para = 1, A_perp = None, Order = 1,
229                     eps = 1e-15, save = False, BC_As_Exact = True):
230     V = FunctionSpace(mesh, "CG", Order)
231     L = FunctionSpace(mesh, "CG", Order)
232     Z = V*L*V*V*L
233
234     print(f"AP, Z.dim() = {Z.dim()}, Order = {Order},"+
235           f" eps = {eps}.")
236     I = Identity(mesh.geometric_dimension())
237     if A_perp == None:
238         A_perp = I
239
240     #Set boundary conditions
241     bc0 = DirichletBC(Z.sub(0), dOmegaDVal, dOmegaD)
242     bc1 = DirichletBC(Z.sub(1), dOmegaDVal, dOmegaD)
243
244     bc3 = DirichletBC(Z.sub(2), dOmegaDVal, dOmegaD)
```

```
    bc4 = DirichletBC(Z.sub(3), dOmegaDVal, dOmegaD)
    bc5 = DirichletBC(Z.sub(4), dOmegaDVal, dOmegaD)

    bcs = [bc0, bc1, bc3, bc4, bc5]
    if (len(dOmegaIN)>0):
        bc2 = DirichletBC(Z.sub(1), 0, dOmegaIN)
        bc6 = DirichletBC(Z.sub(4), 0, dOmegaIN)
        bcs.append(bc2)
        bcs.append(bc6)

    grad_para = lambda alpha: dot(outer(b,b), grad(alpha))
    grad_perp = lambda alpha: dot(I-outer(b,b), grad(alpha))
    a_perp = lambda alpha, beta: inner(A_perp*grad_perp(alpha),
                                       grad_perp(beta))
    a_para = lambda alpha, beta: inner(A_para*grad_para(alpha),
                                       grad_para(beta))

    z = Function(Z)
    p, lam, q, l, mu = split(z)
    eta, kappa, xi, chi, tau = split(TestFunction(Z))

    F = (
        + a_perp(p, eta)*dx #1
        + a_perp(q, eta)*dx
        + a_para(eta, lam)*dx
        - inner(f, eta)*dx
        + a_para(p, kappa)*dx #2
        + a_para(q, xi)*dx #3
        + eps*a_perp(q, xi)*dx
        + eps*a_perp(p, xi)*dx
        + inner(l, xi)*dx
        - eps*inner(f, xi)*dx
        + inner(q, chi)*dx #4
        + a_para(chi, mu)*dx
        + a_para(l, tau)*dx #5
        )

    solve(F==0,z,bcs)
    (p_h, lam_h, q_h, l_h, mu_h) = z.split()
    u_h = Function(V).interpolate(p_h + q_h)
    u_h.rename("u_h")
    #Save Data
    if save:
```

```
288         to_save = [u_h]
289         SAVE(to_save, f, b, mesh, u_h, dOmegaDVal,
290              Order, "AP", BC_As_Exact)
291     return u_h
292 
293 def Solve_PF_Method(f, b, mesh, dOmegaD, dOmegaIN, dOmegaDVal=0,
294                     A_para = 1, A_perp = None, Order = 1,
295                     eps = 1e-15, save = False, BC_As_Exact = True):
296     U = FunctionSpace(mesh, "CG", Order)
297     Q = FunctionSpace(mesh, "CG", Order)
298     Z = U*Q
299 
300     print(f"PF, Z.dim() = {Z.dim()}, Order = {Order},"+
301           f" eps = {eps}.")
302     I = Identity(mesh.geometric_dimension())
303     if A_perp == None:
304         A_perp = I
305 
306     #Set boundary conditions
307     bc0 = DirichletBC(Z.sub(0), dOmegaDVal, dOmegaD)
308     bcs = [bc0]
309     if (len(dOmegaIN)>0):
310         bc2 = DirichletBC(Z.sub(1), 0, dOmegaIN)
311         bcs.append(bc2)
312 
313     #grad_para = lambda alpha: dot(outer(b,b), grad(alpha))
314     grad_perp = lambda alpha: dot(I-outer(b,b), grad(alpha))
315     a_perp = lambda alpha, beta: inner(A_perp*grad_perp(alpha),
316                                        grad_perp(beta))
317     #a_para = lambda alpha, beta: inner(A_para*grad_para(alpha),
318                                        #grad_para(beta))
319 
320     z = Function(Z)
321     u, q = split(z)
322     v, w = split(TestFunction(Z))
323 
324     F = (
325         + inner(grad_perp(u), grad_perp(v))*dx
326         + inner(q*b, grad(v))*dx
327         - inner(f, v)*dx
328         - eps * inner(q, w)*dx
329         + inner(grad(u), w*b)*dx
330         )
```

```
331
332     solve(F == 0, z, bcs)
333     (u_h, q_h) = z.split()
334     u_h.rename("u_h")
335     q_h.rename("q_h")
336     if save:
337         to_save = [u_h, q_h]
338         SAVE(to_save, f, b, mesh, u_h, dOmegaDVal,
339              Order, "PF", BC_As_Exact)
340     return u_h
341
342 def Solve_PF_STAB_Method(f, b, mesh, dOmegaD, dOmegaIN,
343                          dOmegaDVal=0, A_para = 1, A_perp = None,
344                          Order = 1, eps = 1e-15, sigma = 0.1,
345                          save = False, BC_As_Exact = True):
346     U = FunctionSpace(mesh, "CG", Order)
347     Q = FunctionSpace(mesh, "CG", Order)
348     Z = U*Q
349
350     print(f"PF_STAB, Z.dim() = {Z.dim()}, Order = {Order},"+
351           f" eps = {eps}.")
352     I = Identity(mesh.geometric_dimension())
353     if A_perp == None:
354         A_perp = I
355
356     #Set boundary conditions
357     bc = DirichletBC(Z.sub(0), dOmegaDVal, dOmegaD)
358
359     #grad_para = lambda alpha: dot(outer(b,b), grad(alpha))
360     grad_perp = lambda alpha: dot(I-outer(b,b), grad(alpha))
361     a_perp = lambda alpha, beta: inner(A_perp*grad_perp(alpha),
362                                        grad_perp(beta))
363     #a_para = lambda alpha, beta: inner(A_para*grad_para(alpha),
364                                        #grad_para(beta))
365
366     z = Function(Z)
367     u, q = split(z)
368     v, w = split(TestFunction(Z))
369
370     F = (
371         + inner(grad_perp(u), grad_perp(v))*dx
372         + inner(q*b, grad(v))*dx
373         - inner(f, v)*dx
```

```
374         - eps * inner(q, w)*dx
375         - sigma*inner(q, w)*dx
376         + inner(grad(u), w*b)*dx
377         )
378 
379     solve(F == 0, z, bc)
380     (u_h, q_h) = z.split()
381     u_h.rename("u_h")
382     q_h.rename("q_h")
383     if save:
384         to_save = [u_h, q_h]
385         SAVE(to_save, f, b, mesh, u_h, dOmegaDVal,
386              Order, "PF_STAB", BC_As_Exact)
387     return u_h
388 
389 def Solve_DN_Method(f, b, mesh, dOmegaD, dOmegaIN, dOmegaDVal=0,
390                     A_para = 1, A_perp = None, Order = 1,
391                     eps = 1e-15, eps_0 = 1e-7, Repeats = 8):
392     V = FunctionSpace(mesh, "CG", Order)
393     print(f"DN, Z.dim() = {V.dim()}, Order = {Order},"+
394           f" eps = {eps}.")
395     u_e = Function(V)
396     if (dOmegaDVal==0):
397         u_e.interpolate(Constant(0))
398     else:
399         u_e.interpolate(dOmegaDVal)
400 
401     #v, w = TestFunctions(Z)
402     v = TestFunction(V)
403     w = TestFunction(V)
404     I = as_matrix([[1,0],[0,1]])
405 
406     bc0 = DirichletBC(V, dOmegaDVal, dOmegaD)
407     bcs = [bc0]
408 
409     grad_para = lambda u: outer(b,b)*grad(u)
410     grad_perp = lambda u: (I-outer(b,b))*grad(u)
411     a_para = lambda u, v: inner(grad_para(u),grad(v))*dx
412     a_perp = lambda u, v: inner(grad_perp(u),grad(v))*dx
413     a_eps_0 = lambda u, v : a_para(u, v) + eps_0 * a_perp(u, v)
414 
415     u = Function(V).interpolate(Constant(0))
416     q = Function(V).interpolate(Constant(0))
```

```
417
418     counter = 0
419     L2_errors = []
420     H1_errors = []
421     while counter < Repeats:
422         counter += 1
423         u_ = TrialFunction(V) #under score denotes n+1
424         solve(a_eps_0(u_, v) == inner(eps_0*f,v)*dx+
425                                 (eps-eps_0)*a_para(q,v), u, bcs)
426
427         q_ = TrialFunction(V)
428         solve(a_eps_0(q_, w) == inner(f,w)*dx+eps*a_perp(q,w)-
429                                 a_perp(u,w), q, bcs)
430
431         L2_error = norm(u - u_e, "L2")
432         H1_error = norm(u - u_e, "H1")
433         L2_errors.append(L2_error)
434         H1_errors.append(H1_error)
435         print("Counter: "+str(counter) +
436               ", ||u - u_exact||_L2: %.1e" % L2_error)
437         print("Counter: "+str(counter) +
438               ", ||u - u_exact||_H1: %.1e" % H1_error)
439     return u, L2_errors, H1_errors
```

Appendix D

Code File: Examples.py

```
#Examples
from firedrake import *
import numpy as np

def Example_1(al = 0., m = 1., Size = 40, eps = 1e-15):
    # Example with no clossed feild lines
    #Create Mesh
    mesh = UnitSquareMesh(Size,Size, quadrilateral= True)
    x, y = SpatialCoordinate(mesh)

    #Provide Boundary Conditions
    dOmegaD = [3,4]
    dOmegaIN = [1]

    #Calc Vector Field
    b_1 = al*(2*y-1)*cos(m*pi*x)+pi
    b_2 = pi*al*m*(y**2-y)*sin(m*pi*x)
    B = as_vector([b_1,b_2])
    b = B/sqrt(dot(B,B))

    #Provide Exact Solution
    u_e0 = sin(pi*y+al*(y**2-y)*cos(m*pi*x))
    u_e1 = cos(2*pi*x)*sin(pi*y)
    u_e = u_e0 + eps*u_e1

    #Calc f
    I = Identity(mesh.geometric_dimension())
    A_para = 1
    A_perp = I
```

```
    A_eps = (1/eps)*A_para*outer(b,b)
    A_eps += (I-outer(b,b)) * A_perp * (I-outer(b, b))
    f = -div(dot(A_eps, grad(u_e)))
    return f, b, mesh, dOmegaD, dOmegaIN, u_e


def Example_2(eps = 1e-15, Restrict_CL =True):
    # Annulus (Clossed Field Lines)
    # 1<r<2
    mesh = Mesh("Mesh/Annulus_Side_Mesh.msh")
    x, y = SpatialCoordinate(mesh)

    #Provide Boundary Conditions
    dOmegaD = [11, 12]
    dOmegaIN = []
    if Restrict_CL:
        dOmegaIN.append(1) #MagSplit Num

    #Calc Vector Field
    b_1 = -y
    b_2 = x
    B = as_vector([b_1,b_2])
    b = B/sqrt(dot(B,B))

    #Provide Exact Solution
    u_e0 = (x**2+y**2-1)*(4-x**2-y**2)
    u_e1 = (x**2+y**2-1)*(4-x**2-y**2)
    u_e = u_e0 + eps*u_e1


    #Calc f
    I = Identity(mesh.geometric_dimension())
    A_para = 1
    A_perp = I
    A_eps = (1/eps)*A_para*outer(b,b)
    A_eps += (I-outer(b,b)) * A_perp * (I-outer(b, b))
    f = -div(dot(A_eps, grad(u_e)))
    return f, b, mesh, dOmegaD, dOmegaIN, u_e

def Example_3(a = 0.05, Size = 40,
              eps = 1e-15, Restrict_CL = False):
    #Magnetic Island (Clossed Feild Lines) from DN paper
    #Create Mesh
```

```
73      mesh = UnitSquareMesh(Size,Size, quadrilateral= True)
74      if (Restrict_CL):
75          #Load mesh
76          mesh = Mesh("Mesh/Square_Mesh.msh")
77
78
79      x, y = SpatialCoordinate(mesh)
80
81      #Provide Boundary Conditions
82      dOmegaD = [3,4]
83      dOmegaIN = [1]
84      if Restrict_CL:
85          dOmegaIN.append(9)
86
87      #Calc Vector Field
88      b_1 = -cos(pi*y)
89      b_2 = 4*a*sin(4*pi*x)
90      B = as_vector([b_1,b_2])
91      b = B/sqrt(dot(B,B))
92
93      #Provide Exact Solution
94      u_e0 = sin(10*(sin(pi*y)-a*cos(4*pi*x)))
95      u_e1 = cos(2*pi*x)*sin(10*pi*y)
96      u_e = u_e0 + eps*u_e1
97
98      #Calc f
99      I = Identity(mesh.geometric_dimension())
100     A_para = 1
101     A_perp = I
102     A_eps = (1/eps)*A_para*outer(b,b)
103     A_eps += (I-outer(b,b)) * A_perp * (I-outer(b, b))
104     f = -div(dot(A_eps, grad(u_e)))
105     return f, b, mesh, dOmegaD, dOmegaIN, u_e
106
107 def Example_4(a = 0.05, Size = 40,
108               eps = 1e-15, Restrict_CL = False):
109     #Magnetic Island (Clossed Feild Lines) adaption of example 3
110     # Enforces vector field to have b dot n = 0, on dOmegaD
111     #Create Mesh
112     mesh = UnitSquareMesh(Size,Size, quadrilateral= True)
113     if (Restrict_CL):
114         #Load mesh
115         mesh = Mesh("Mesh/Square_Mesh.msh")
```

```
116
117
118     x, y = SpatialCoordinate(mesh)
119
120     #Provide Boundary Conditions
121     dOmegaD = [3,4]
122     dOmegaIN = [1]
123     if Restrict_CL:
124         dOmegaIN.append(9)
125
126     #Calc Vector Field
127     b_1 = -cos(pi*y)
128     b_2 = 4*a*sin(4*pi*x)*sin(pi*y)
129     B = as_vector([b_1,b_2])
130     b = B/sqrt(dot(B,B))
131
132     #Provide Exact Solution
133     u_e0 = sin(10*sin(pi*y)*e**(-a*cos(4*pi*x)))
134     u_e1 = u_e0
135     u_e = u_e0 + eps*u_e1
136
137     #Calc f
138     I = Identity(mesh.geometric_dimension())
139     A_para = 1
140     A_perp = I
141     A_eps = (1/eps)*A_para*outer(b,b)
142     A_eps += (I-outer(b,b)) * A_perp * (I-outer(b, b))
143     f = -div(dot(A_eps, grad(u_e)))
144     return f, b, mesh, dOmegaD, dOmegaIN, u_e
145
146 def Example_5(eps = 1e-15, Restrict_CL = True):
147     #Torus (Clossed Field Lines)
148     r_I = 1
149     mesh = Mesh("Mesh/Torus_Side_Mesh.msh")
150     x, y, z = SpatialCoordinate(mesh)
151
152
153     #Privide Boundary Conditions
154     dOmegaD = [1]
155     dOmegaIN = []
156     if Restrict_CL:
157         dOmegaIN.append(9)
158
```

```
159     #Calc Vector Field
160     b_1 = -y
161     b_2 = x
162     b_3 = Constant(0)
163     B = as_vector([b_1,b_2,b_3])
164     b = B/sqrt(dot(B,B))
165 
166     #Provide Exact Solution
167     #r = sqrt(r_I**2+x**2+y**2+z**2-2*r_I*sqrt(x**2+y**2))
168     r_s = r_I**2+x**2+y**2+z**2-2*r_I*sqrt(x**2+y**2)
169     u_e01 = -r_s + 0.25
170     u_e = u_e01 + eps*u_e01
171 
172     #Calc f
173     I = Identity(mesh.geometric_dimension())
174     A_para = 1
175     A_perp = I
176     A_eps = (1/eps)*A_para*outer(b,b)
177     A_eps += (I-outer(b,b)) * A_perp * (I-outer(b, b))
178     f = -div(dot(A_eps, grad(u_e)))
179     return f, b, mesh, dOmegaD, dOmegaIN, u_e
```

Appendix E

Code File: CUSP_Par_GMRES.cu

```
#include <cusp/coo_matrix.h>
#include <cusp/monitor.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/gmres.h>
#include <iostream>
#include "My_StopWatch.hpp"


// where to perform the computation
typedef cusp::device_memory MemorySpace;
// which floating point type to use
typedef float ValueType;
int main(void)
{
  StopWatch My_Watch;
    // create an empty sparse matrix structure (coo format)
  cusp::coo_matrix<int, ValueType, MemorySpace> A;

    // load data from matrix file
  cusp::io::read_matrix_market_file(A, "thermal1.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<ValueType, MemorySpace> x(A.num_rows);
    cusp::array1d<ValueType, MemorySpace> b(A.num_rows);
  cusp::io::read_matrix_market_file(b, "thermal1_b.mtx");


    // set stopping criteria:
  std::cout<<"Start Solve"<<"\n";
```

```
30   int its = 10000;
31   float tols[4] = {1e-02, 1e-03, 1e-04, 1e-05};
32   bool verbose = true;
33   for (int i=0; i<4; ++i)
34   {
35     float tol = tols[i];
36     std::cout<<"-----START-tol="<<tol<<"----------"<<std::endl;
37     // set initial guess
38     thrust::fill( x.begin(), x.end(), ValueType(0));
39
40       cusp::monitor<ValueType> monitor(b, its, tol, 0, verbose);
41     int restart = 50;
42     // solve the linear system A * x = b with the GMRES
43     //Start Timer
44     My_Watch.Start();
45     cusp::krylov::gmres(A, x, b,restart, monitor);
46     My_Watch.Stop();
47     std::cout<<"Done Solve: " << My_Watch.Time() << std::endl;
48   }
49
50     return 0;
51 }
```

Appendix F

Code File: Python_Seq_GMRES.py

```
#Named Sequential, but utilises all cores on CPU
from scipy.io import mmread
from scipy.sparse.linalg import *
import numpy as np
import time
A = mmread('thermal1.mtx')
b = mmread('thermal1_b.mtx')
tols = [1e-02, 1e-03, 1e-04, 1e-05]
for tol in tols:
    print("----tol="+str(tol)+"----")
    start = time.time()
    x, exitCode = gmres(A, b, tol=tol, atol=0, restart = 50)
    end = time.time()
    print("Time s: "+str((end-start)))
    print("|Ax-b| = "+str(np.linalg.norm(A.dot(x)-b[:,0])))
```