A mixed Finite Element approach to the Magnetohydrodynamics problem

by

Michael Wathen

M.Sci., The University of Birmingham, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

March 2014

Abstract

Preface

Table of Contents

List of Tables

List of Figures

Acknowledgements

Dedication

Chapter 1

Introduction

stationary incompressible and resistive magnetohydrodynamics (MHD) system:

$$-\nu\,\Delta \boldsymbol{u} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + \nabla p - \kappa\,(\nabla\times\boldsymbol{b})\times\boldsymbol{b} = \boldsymbol{f} \qquad \text{in } \Omega, \tag{1.0.1a}$$

$$\nabla\cdot\boldsymbol{u} = 0 \qquad \text{in } \Omega, \tag{1.0.1b}$$

$$\kappa\nu_m\,\nabla\times(\nabla\times\boldsymbol{b}) + \nabla r - \kappa\,\nabla\times(\boldsymbol{u}\times\boldsymbol{b}) = \boldsymbol{g} \qquad \text{in } \Omega, \tag{1.0.1c}$$

$$\nabla\cdot\boldsymbol{b} = 0 \qquad \text{in } \Omega. \tag{1.0.1d}$$

Here, Ω is a bounded simply-connected Lipschitz polyhedron in $\mathbb{R}^3$, with a connected boundary $\partial\Omega$. The unknowns are the velocity $\boldsymbol{u}$, the hydrodynamic pressure p, the magnetic field $\boldsymbol{b}$, and the Lagrange multiplier r associated with the divergence constraint on the magnetic field. The functions $\boldsymbol{f}$ and $\boldsymbol{g}$ represent external force terms.

The equations (1.0.1) are characterized by three dimensionless parameters: the hydrodynamic Reynolds number $\mathrm{Re} = \nu^{-1}$, the magnetic Reynolds number $\mathrm{Rm} = \nu_m^{-1}$, and the coupling number κ. For further discussion of these parameters and their typical values, we refer the reader to [**?** **?** **?**]. We consider the following homogeneous and inhomogeneous Dirichlet

boundary conditions:

$$\boldsymbol{u} = \boldsymbol{0} \qquad \text{on } \partial\Omega, \tag{1.0.2a}$$

$$\boldsymbol{n} \times \boldsymbol{b} = \boldsymbol{0} \qquad \text{on } \partial\Omega, \tag{1.0.2b}$$

$$r = 0 \qquad \text{on } \partial\Omega, \tag{1.0.2c}$$

with $\boldsymbol{n}$ being the unit outward normal on $\partial\Omega$. By taking the divergence of the magnetostatic equation (1.0.1c), we obtain the Poisson problem

$$\Delta r = \nabla \cdot \boldsymbol{g} \quad \text{in } \Omega, \qquad r = 0 \quad \text{on } \partial\Omega. \tag{1.0.3}$$

Since $\boldsymbol{g}$ is divergence-free in physically relevant applications, the multiplier r is typically zero and its primary purpose is to ensure stability; see also [**?**].

Our goal in this paper is to derive a scalable numerical solution procedure for solving (1.0.1)-(1.0.2). We first present in Section 3 the finite element discretization and the structure of the matrices that arise throughout the nonlinear iterations. In Section **??** we introduce the proposed preconditioners for the indefinite linear systems that arise; our ideas are based on combining preconditioners for the incompressible Navier-Stokes and Maxwell sub-systems appearing in the MHD system (1.0.1a)–(1.0.1d) while taking into account the presence of coupling terms. Spectral analysis is performed in Section **??**. [**DS: More precise and quote earlier papers**]. In Section **??** we consider circumstances where the incompressible Navier-Stokes and the Maxwell systems can be decoupled; in such cases the precondi-

tioning approach may be significantly simplified. In Section **??** we provide preliminary results in two dimensions that show that our numerical solution techniques are feasible and reasonably scalable. Finally, we offer some concluding remarks in Section **??**.

Chapter 2

FEniCS

The FEniCS project started in 2003 and the aim was create software that automates a finite element discretisation and solution of differential equations. The core libraries used within FEniCS are DOLFIN [13, 14], FFC [11, 12, 16], FIAT [9, 10], Instant, UFC [2, 3] and UFL [1, 4]. Along the these core packages FEniCS has a few extra optional packages that can be found here `http://fenicsproject.org/applications/`.

2.1 Overview

One of the key ideas of FEniCS was to be able to write an easy to use software package for the solution of partial differential equations (PDEs). The main underlying code base for FEniCS is written in C++. and Python

FEniCS supports large range of a different finite element function space. Thus allowing FEniCS to be used from electromagnetic problems to fluid. The full list of support elements can be found in table 2.1.

This breath introduction to FEniCS will go through how to set up a very simple test case, namely the Poisson equation. For this simple problem we will consider both Dirichlet and Neumann boundary conditions and show how to set up the PDE in its variational form.

Name	Usage
Argyris*	“ARG”
Arnold-Winther*	“AW”
Brezzi-Douglas-Fortin-Marini*	“BDFM”
Brezzi-Douglas-Marini	“BDM”
Bubble	“B”
Crouzeix-Raviart	“CR”
Discontinuous Lagrange	“DG”
Hermite*	“HER”
Lagrange	“CG”
Mardal-Tai-Winther*	“MTW”
Morley*	“MOR”
Nedelec 1st kind H(curl)	“N1curl”
Nedelec 2nd kind H(curl)	“N2curl”
Quadrature	“Q”
Raviart-Thomas	“RT”

Table 2.1: Avaliable finite element function space (*only partly supported)

2.2 Poisson example

For a simple example to see how FEniCS works consider the Poisson equation. Defining $\Omega \subset \mathbb{R}^n$ to be the domain and the boundary as $\partial\Omega = \Gamma_D \cup \Gamma_N$ where Γ_D and Γ_N correspond to the Dirichlet and Neumann boundaries respectively. Thus, the Poisson equation with corresponding boundary conditions reads as:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{in } \Gamma_D \\ \nabla u \cdot n &= g \quad \text{in } \Gamma_N \end{aligned} \tag{2.2.1}$$

For this example we will be considering the following function, domains and boundaries to be:

- $f = (x - 1/2)^2 - 4(y - 1/2)^4$
- $g = 5x$
- $\Omega = [0,1] \times [0,1]$
- $\Gamma_D = \{(0,y) \cup (1,y) \quad | \quad 0 \leq y \leq 1\}$
- $\Gamma_N = \{(x,0) \cup (x,1) \quad | \quad 0 \leq x \leq 1\}$

Defining the following trial and test function spaces $\mathcal{V}$ and $\bar{\mathcal{V}}$ as

$$\begin{aligned} \mathcal{V} &= H^1_{u_0}(\Omega) = \left\{ u \in H^1(\Omega) \,:\, u = u_0 \text{ on } \partial\Omega \right\}, \\ \bar{\mathcal{V}} &= H^1_0(\Omega) = \left\{ u \in H^1(\Omega) \,:\, u = 0 \text{ on } \partial\Omega \right\}. \end{aligned} \tag{2.2.2}$$

Thus the variational formulation of (2.2.1) depends on finding $u \in \mathcal{V}$ such that

$$a(u,v) = L(v) \quad \forall v \in \bar{\mathcal{V}}, \tag{2.2.3}$$

where

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dx \quad \text{and} \quad L(v) = \int_\Omega f v dx + \int_{\Gamma_N} g v \, ds. \tag{2.2.4}$$

2.2.1 Defining mesh and function space

For this example we want to consider a uniform unit square mesh. This is created by using the following python code

```
1 mesh = UnitSquareMesh(32,32)
```

Once the mesh is create we no turn to the function space. For this model problem we want to consider the function space $\mathcal{V}$ and $\bar{\mathcal{V}}$ defined by (2.2.2).

Therefore, the discrete finite element space use is the following space

$$\boldsymbol{V}_h = \{\, \boldsymbol{u} \in H_1(\Omega) \,:\, \boldsymbol{u}|_K \in \mathcal{P}_1(K),\, K \in \mathcal{T}_h \,\},$$

where $\mathcal{T}_h = \{K\}$ regular and quasi-uniform triangles. This function space is create in FEniCS using the following command

```
V = FunctionSpace(mesh,'CG',1)
```

2.2.2 Defining subdomains

Separating different parts of the boundary is easily done within FEniCS. For this our simple Possion model the boundary is split up into two parts. The top and bottom defines the Dirichlet boundary condition whilst the left and right portions of the boundary define the Neumann conditions. Before partitioning the domain into the Dirichlet and Neumann boundaries we need to define the different sections of the boundary. This is done using classes as follows:

```
# Defining boundary classes
class Left(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[0], 0.0)

class Right(SubDomain):
    def inside(self, x, on_boundary):
        return near(x[0], 1.0)
```

```
9
10 class Bottom(SubDomain):
11     def inside(self, x, on_boundary):
12         return near(x[1], 0.0)
13
14 class Top(SubDomain):
15     def inside(self, x, on_boundary):
16         return near(x[1], 1.0)
17
18 # Initialize sub-domain instances
19 left = Left()
20 top = Top()
21 right = Right()
22 bottom = Bottom()
```

Using the classes above we group the Dirichlet and Neumann boundaries together. These can then be used to set up the boundary conditions for our problem.

```
1 boundaries = FacetFunction("size_t", mesh)
2 boundaries.set_all(0)
3 left.mark(boundaries, 1)
4 right.mark(boundaries, 1)
5 top.mark(boundaries, 2)
6 bottom.mark(boundaries, 2)
```

2.2.3 Boundary Conditions and weak formulation

The boundaries have been split into their separate domains so now we can think about defining the boundary conditions, bilinear and linear (2.2.3) forms.

First consider the homogeneous Dirichlet boundary:

$$u = 0 \quad \text{in } \Gamma_D.$$

The Dirichlet boundary condition is created by using the `DirichletBC` class. When defining Dirichlet boundary conditions on separate subdomains we need to use four inputs to the `DirichletBC` class. The first one is the function space that the boundary condition applies to, the value on the boundary and then the last two arguments define which part of the boundary the boundary condition is defined on.

```
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundaries,1)
```

To define the variational form of the problem we first need to determine the trial function `u` and test function `v` belonging to the function space $\mathcal{V}$. Furthermore, the source term f and Neumann condition g are required in the linear form $L(v)$. Both are defined using the `Expression` class. Together with the trial and test functions the variational form can be created using the following code:

```
# Define variational problem
u = TrialFunction(V)
```

```
v = TestFunction(V)
f = Expression("(pow(x[0]-0.5,2)-4*pow(x[1]-0.5,4))")
g = Expression("(5*x[0])")
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds(2)
```

2.2.4 Assembly and solving

Now the variational (or weak form) have been stated the next step is to solve the problem. This can be done in two was within FEniCS

1. Assemble system and solve
2. Solve without assembly.

First, consider assembling then solve the system. This can be done by using the `assemble_system` class. This class takes the variational form as the inputs and the Dirichlet boundary condition.

```
A, b = assemble_system(a,L,bc)
```

The final solving step is the same for both ways. To do this we first define a function `u` in the corresponding function space ($\mathcal{V}$) that will represent the solution. Next, calling the `solve` function with either the matrix `A` and vector `b` as the arguments or the variational form will compute the solution using sparse direct solvers which is the default solving method for FEniCS

```
u = Function(V)
solve(A,u.vector(),b)
solve(a == L, u, bc)
```

Optional solution parameters

For simplicity above the `solve` used FEniCS default parameters. However, this may not necessarily be the most efficient so may want to change these parameters. It is well known for example that multigrid is the most efficient solve for the Poisson problem. Here is how to set this up.

```
solve(a == L, u, bc,
            solver_parameters=dict(linear_solver="cg",
            preconditioner="amg"))
```

For more optional parameters use the following commands:

```
print list_krylov_solver_methods()
print list_krylov_solver_preconditioners()
print list_linear_algebra_backends()
print list_linear_solver_methods()
print list_lu_solver_methods()
```

Instead of using FEniCS's inbuilt solving function you can use other external packages. For instance you can link you your FEniCS code with Trilinos [7, 8], PETSc [5, 6] or your own code.

2.2.5 Visualisation

Once you have solved your problem the next step is to either visualise or save the result. To output the solution in a VTK file you need to create a file with `.pvd` suffix. This enables you to use external software (for example ParaView []) to visualise the solution. Alternatively FEniCS's has an interface for the VTK visualisation tools. We do this by using the `plot` command.

```
# Save solution in VTK format
file = File("poisson.pvd")
file << u

# Plot solution
plot(u, interactive=True)
```

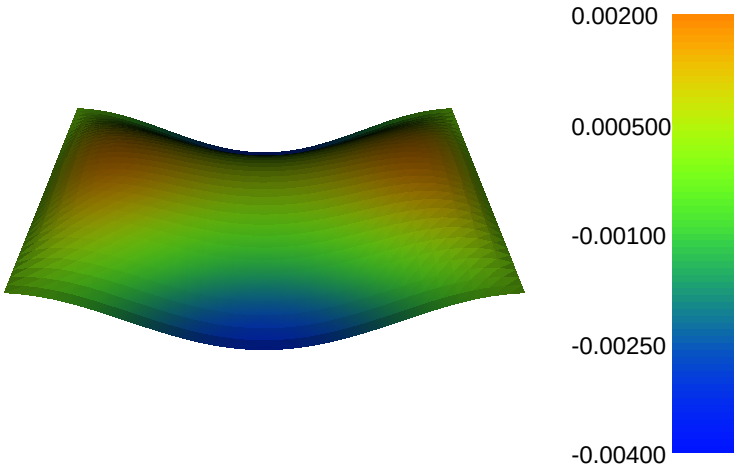

Figure 2.1: Numerical solution to (2.2.1)

Chapter 3

Finite element discretization

The section will introduce the mixed finite element discretisation of (1.0.1) together with the principle properties of the discretisation.

3.1 Variational formulation

Before writing (1.0.1) in the weak form, we first need to denote the inner product of $L^2(\Omega)$ or $L^2(\Omega)^3$ as $(\cdot,\cdot)_\Omega$. Using the Sobolev spaces

$$\begin{aligned}
\boldsymbol{V} &= H_0^1(\Omega)^d = \left\{ \boldsymbol{u} \in H^1(\Omega)^d \,:\, \boldsymbol{u} = \boldsymbol{0} \text{ on } \partial\Omega \right\}, \\
Q &= H_0^1(\Omega) = \{\, p \in H^1(\Omega) \,:\, (p\,,1)_\Omega = 0 \,\}, \\
\boldsymbol{C} &= H_0(\mathrm{curl};\Omega) = \left\{ \boldsymbol{b} \in L^2(\Omega)^d \,:\, \nabla \times \boldsymbol{b} \in L^2(\Omega)^d,\ \boldsymbol{n} \times \boldsymbol{b} = \boldsymbol{0} \text{ on } \partial\Omega \right\}, \\
S &= H_0^1(\Omega) = \{\, r \in H^1(\Omega) \,:\, r = 0 \text{ on } \partial\Omega \,\},
\end{aligned} \tag{3.1.1}$$

the weak formulation of the incompressible MHD system (1.0.1) and the boundary conditions (1.0.2) depends on finding $(\boldsymbol{u}, p, \boldsymbol{b}, r) \in \boldsymbol{V} \times Q \times \boldsymbol{C} \times S$

such that

$$
\begin{aligned}
A(\boldsymbol{u},\boldsymbol{v}) + O(\boldsymbol{u};\boldsymbol{u},\boldsymbol{v}) + C(\boldsymbol{b};\boldsymbol{v},\boldsymbol{b}) + B(\boldsymbol{v},p) &= (\boldsymbol{f},\boldsymbol{v})_\Omega, && \text{(3.1.2a)}\\
B(\boldsymbol{u},q) &= 0, && \text{(3.1.2b)}\\
M(\boldsymbol{b},\boldsymbol{c}) - C(\boldsymbol{b};\boldsymbol{u},\boldsymbol{c}) + D(\boldsymbol{c},r) &= (\boldsymbol{g},\boldsymbol{c})_\Omega, && \text{(3.1.2c)}\\
D(\boldsymbol{b},s) &= 0, && \text{(3.1.2d)}
\end{aligned}
$$

for all $(\boldsymbol{v},q,\boldsymbol{c},s) \in \boldsymbol{V}\times Q\times \boldsymbol{C}\times S$. The individual variational forms are given by

$$
\begin{aligned}
A(\boldsymbol{u},\boldsymbol{v}) &= \int_\Omega \nu\,\nabla\boldsymbol{u} : \nabla\boldsymbol{v}\,d\boldsymbol{x}, & O(\boldsymbol{w};\boldsymbol{u},\boldsymbol{v}) &= \int_\Omega (\boldsymbol{w}\cdot\nabla)\boldsymbol{u}\cdot\boldsymbol{v}\,d\boldsymbol{x},\\
B(\boldsymbol{u},q) &= -\int_\Omega (\nabla\cdot\boldsymbol{u})\,q\,d\boldsymbol{x}, & M(\boldsymbol{b},\boldsymbol{c}) &= \int_\Omega \kappa\nu_m(\nabla\times\boldsymbol{b})\cdot(\nabla\times\boldsymbol{c})\,d\boldsymbol{x},\\
D(\boldsymbol{b},s) &= \int_\Omega \boldsymbol{b}\cdot\nabla s\,d\boldsymbol{x}, & C(\boldsymbol{d};\boldsymbol{v},\boldsymbol{b}) &= \int_\Omega \kappa\,(\boldsymbol{v}\times\boldsymbol{d})\cdot(\nabla\times\boldsymbol{b})\,d\boldsymbol{x}.
\end{aligned}
$$

To verify convergence results we introduce the following integral norms:

$$
\begin{aligned}
\|f\|_{L_2(\Omega)} &= \left(\int_\Omega (f,f)\;dx\right)^{\frac{1}{2}},\\
\|f\|_{H(curl,\Omega)} &= \left(\|f\|^2_{L_2(\Omega)} + \|\nabla\times f\|^2_{L_2(\Omega)}\right)^{\frac{1}{2}},\\
\|f\|_{H^1(\Omega)} &= \left(\|f\|^2_{L_2(\Omega)} + \|\nabla f\|^2_{L_2(\Omega)}\right)^{\frac{1}{2}}
\end{aligned}
\qquad (3.1.3)
$$

3.2 Mixed finite element discretisation

Consider the domain Ω to be divided up into regular and quasi-uniform triangles ($d=2$) or tetrahedral ($d=3$) $\mathcal{T}_h = \{K\}$ with mesh size h. Following

for function spaces defined in (3.1.1) then the finite element solution will lie in finite element functions spaces given here:

$$\begin{aligned}
\boldsymbol{V}_h &= \\
Q_h &= \\
\boldsymbol{C}_h &= \{\, \boldsymbol{b} \in H_0(\mathrm{curl};\Omega) \,:\, \boldsymbol{b}|_K \in \mathcal{P}_{k-1}(K)^d \oplus \boldsymbol{R}_k(K),\, K \in \mathcal{T}_h \,\}, \\
S_h &= \{\, r \in H_0^1(\Omega) \,:\, r|_K \in \mathcal{P}_k(K),\, K \in \mathcal{T}_h \,\}.
\end{aligned} \tag{3.2.1}$$

fluid equations ..... For the magnetic field space ($\boldsymbol{C}_h$) we use the curl-conforming Nédélec elements [15] of the first kind. These choices of finite elements spaces $\boldsymbol{V}_h$, $\boldsymbol{C}_h$, Q_h and S_h means we have conforming subspaces to our Sobolev spaces $\boldsymbol{V}$, $\boldsymbol{C}$, Q and S, respectively. Thus, the finite element solution depends on finding $(\boldsymbol{u}_h, p_h, \boldsymbol{b}_h, r_h) \in \boldsymbol{V}_h \times Q_h \times \boldsymbol{C}_h \times S_h$ such that

$$A(\boldsymbol{u}_h, \boldsymbol{v}) + O(\boldsymbol{u}_h; \boldsymbol{u}_h, \boldsymbol{v}) + C(\boldsymbol{b}_h; \boldsymbol{v}, \boldsymbol{b}_h) + B(\boldsymbol{v}, p_h) = (\boldsymbol{f}, \boldsymbol{v}), \tag{3.2.2a}$$

$$B(\boldsymbol{u}_h, q) = 0, \tag{3.2.2b}$$

$$M(\boldsymbol{b}_h, \boldsymbol{c}) - C(\boldsymbol{b}_h; \boldsymbol{u}_h, \boldsymbol{c}) + D(\boldsymbol{c}, r_h) = (\boldsymbol{g}, \boldsymbol{c}), \tag{3.2.2c}$$

$$D(\boldsymbol{b}_h, s) = 0, \tag{3.2.2d}$$

for all $(\boldsymbol{v}, q, \boldsymbol{c}, s) \in \boldsymbol{V}_h \times Q_h \times \boldsymbol{C}_h \times S_h$.

For the convection term ($O(\cdot;\cdot,\cdot)$) we would like to ensure the property

$$O(\boldsymbol{w}; \boldsymbol{u}, \boldsymbol{u}) = 0. \tag{3.2.3}$$

To ensure this property we integrate by parts (3.2.3) to obtain

$$\begin{aligned}\int_\Omega (\boldsymbol{w}\cdot\nabla)\boldsymbol{u}\cdot\boldsymbol{u}\,d\boldsymbol{x} = &-\frac{1}{2}\int_\Omega \nabla\cdot\boldsymbol{w}\boldsymbol{u}\cdot\boldsymbol{u}\,d\boldsymbol{x}\\ &+\frac{1}{2}\int_{\partial\Omega}\boldsymbol{w}\cdot\boldsymbol{n}|\boldsymbol{u}|^2\,ds,\end{aligned} \tag{3.2.4}$$

where $\boldsymbol{n}$ is the unit outward normal on $\partial\Omega$. Therefore, we write the non-linear convection term as

$$O(\boldsymbol{w};\boldsymbol{u},\boldsymbol{v}) = \int_\Omega (\boldsymbol{w}\cdot\nabla)\boldsymbol{u}\cdot\boldsymbol{v}\,d\boldsymbol{x} + \frac{1}{2}\int_\Omega \nabla\cdot\boldsymbol{w}\boldsymbol{u}\cdot\boldsymbol{v}\,d\boldsymbol{x} - \frac{1}{2}\int_{\partial\Omega}\boldsymbol{w}\cdot\boldsymbol{n}\boldsymbol{u}\cdot\boldsymbol{v}\,ds.$$

However, the forms for A, M, B, D and C stay the same as on the continuous level.

3.2.1 Matrix representation

To define the matrix representation from the variational form (3.2.2) we need to introduce the basis function for the finite elements spaces (3.2.1):

$$\boldsymbol{V}_h = \text{span}\langle\boldsymbol{\psi}_j\rangle_{j=1}^{n_u}, \qquad Q_h = \text{span}\langle\alpha_i\rangle_{i=1}^{m_u}, \tag{3.2.5}$$

$$\boldsymbol{C}_h = \text{span}\langle\boldsymbol{\phi}_j\rangle_{j=1}^{n_b}, \qquad S_h = \text{span}\langle\beta_i\rangle_{i=1}^{m_b}. \tag{3.2.6}$$

The aim is to find the coefficient vectors $u = (u_1,\ldots,u_{n_u}) \in \mathbb{R}^{n_u}$, $p = (p_1,\ldots,p_{m_u}) \in \mathbb{R}^{m_u}$, $b = (b_1,\ldots,b_{n_b}) \in \mathbb{R}^{n_b}$, and $r = (r_1,\ldots,r_{m_b}) \in \mathbb{R}^{m_b}$ of the finite element functions $(\boldsymbol{u}_h, p_h, \boldsymbol{b}_h, r_h)$. This is done by writing writing

the bilinear forms in (3.2.2) in there corresponding matrix forms

$$
\begin{aligned}
A_{i,j} &= A(\boldsymbol{\psi}_j, \boldsymbol{\psi}_i), && 1 \le i, j \le n_u, \\
B_{i,j} &= B(\boldsymbol{\psi}_j, \alpha_i), && 1 \le i \le m_u,\ 1 \le j \le n_u, \\
D_{i,j} &= D(\boldsymbol{\phi}_j, \beta_i), && 1 \le i \le m_b,\ 1 \le j \le n_b, \\
M_{i,j} &= M(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i), && 1 \le i, j \le n_b, \\
f_i &= (\boldsymbol{f}, \boldsymbol{\psi}_i)_\Omega, && 1 \le i \le n_u, \\
g_i &= (\boldsymbol{g}, \boldsymbol{\phi}_i)_\Omega, && 1 \le i \le n_b.
\end{aligned}
$$

For the two non-linear forms (O and C) we need to introduce two finite element functions $\boldsymbol{w} \in \boldsymbol{V}_h$ and $d \in \boldsymbol{C}_h$. Therefore, the convection and coupling terms are defined as:

$$
\begin{aligned}
O(w)_{i,j} &= O_h(\boldsymbol{w}; \boldsymbol{\psi}_j, \boldsymbol{\psi}_i), \quad 1 \le i, j \le n_u, \\
C(d)_{i,j} &= C(\boldsymbol{d}; \boldsymbol{\psi}_j, \boldsymbol{\phi}_i), \qquad 1 \le i \le n_b,\ 1 \le j \le n_u.
\end{aligned}
$$

Thus, the numerical solution to (1.0.1) depends on solving

$$
\begin{pmatrix}
A + O(u) & B^T & C^T(b) & 0 \\
B & 0 & 0 & 0 \\
-C(b) & 0 & M & D^T \\
0 & 0 & D & 0
\end{pmatrix}
\begin{pmatrix} u \\ p \\ b \\ r \end{pmatrix}
=
\begin{pmatrix} f \\ 0 \\ g \\ 0 \end{pmatrix}. \tag{3.2.7}
$$

where the vectors $u \in \mathbb{R}^{n_u}$, $p \in \mathbb{R}^{m_u}$, $b \in \mathbb{R}^{n_b}$, and $r \in \mathbb{R}^{m_b}$ are the coefficients of the finite element functions.

3.3 Oseen/Picard iterations

From (1.0.1) we can see that the MHD system is a non-linear model. Therefore there is a need to apply a non-linear solve to this problem. A common choice to deal with the non-linearity within Navier-Stokes is to perform Oseen or Picard iterations. This involves linearising about the convection term and solving for the increment updates. This is indeed what we will do for the full MHD system. Given the solution (u_k, p_k, b_k, r_k) then we solve for $(\delta u_k, \delta p_k, \delta b_k, \delta r_k)$ to update the solution:

$$u_{k+1} = u_k + \delta u_k, \tag{3.3.1}$$

$$p_{k+1} = p_k + \delta p_k, \tag{3.3.2}$$

$$b_{k+1} = b_k + \delta b_k, \tag{3.3.3}$$

$$r_{k+1} = r_k + \delta r_k. \tag{3.3.4}$$

Form this point onwards we will be dropping the iterations k. This means we will be solving for $(\delta u, \delta p, \delta b, \delta r)$ but still update our solution vector (u, p, b, r).

Using (3.3.1) then the variational form (3.2.2) becomes

$$\begin{aligned} A(\delta \boldsymbol{u}_h, \boldsymbol{v}) + O(\boldsymbol{u}; \delta \boldsymbol{u}_h, \boldsymbol{v}) + C(\boldsymbol{b}_h; \boldsymbol{v}, \delta \boldsymbol{u}_h) + B(\boldsymbol{v}, \delta p_h) &= R_u(\boldsymbol{u}_h, \boldsymbol{b}_h, p_h; \boldsymbol{v}) \\ B(\delta \boldsymbol{u}_h, q) &= R_p(\boldsymbol{u}_h; q), \\ M(\delta \boldsymbol{b}_h, \boldsymbol{c}) + D(\boldsymbol{c}, \delta r_h) - C(\boldsymbol{b}_h; \delta \boldsymbol{u}_h, \boldsymbol{v}) &= R_b(\boldsymbol{u}_h, \boldsymbol{b}_h, r_h; \boldsymbol{c}), \\ D(\delta \boldsymbol{b}_h, s) &= R_r(\boldsymbol{b}_h; s). \end{aligned} \tag{3.3.5}$$

where right hand side linear forms correspond to the residual at the previous iteration defined by:

$$
\begin{aligned}
R_u(\boldsymbol{u}_h, \boldsymbol{b}_h, p_h; \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v})_\Omega - A_h(\boldsymbol{u}_h, \boldsymbol{v}) - O_h(\boldsymbol{u}_h; \boldsymbol{u}_h, \boldsymbol{v}) - C(\boldsymbol{b}_h; \boldsymbol{v}, \boldsymbol{b}_h) - B(\boldsymbol{v}, p_h), \\
R_p(\boldsymbol{u}_h; q) &= -B(\boldsymbol{u}_h, q), \\
R_b(\boldsymbol{u}_h, \boldsymbol{b}_h, r_h; \boldsymbol{c}) &= (\boldsymbol{g}, c)_\Omega - M(\boldsymbol{b}_h, \boldsymbol{c}) + C(\boldsymbol{b}_h; \boldsymbol{u}_h, \boldsymbol{c}) - D(\boldsymbol{c}, r_h), \\
R_r(\boldsymbol{b}_h; s) &= -D(\boldsymbol{b}_h, s).
\end{aligned}
$$

From (3.2.7) we formed the non-linearised problem in its matrix form. Using (3.3.5) the new linear system is as follows:

$$
\begin{pmatrix}
A + O(u) & B^T & C^T(b) & 0 \\
B & 0 & 0 & 0 \\
-C(b) & 0 & M & D^T \\
0 & 0 & D & 0
\end{pmatrix}
\begin{pmatrix}
\delta u \\ \delta p \\ \delta b \\ \delta r
\end{pmatrix}
=
\begin{pmatrix}
r_u \\ r_p \\ r_b \\ r_r
\end{pmatrix}, \tag{3.3.6}
$$

with

$$
\begin{aligned}
r_u &= f - Au - O(u)u - C(b)^T b - B^T p, \\
r_p &= -Bu, \\
r_b &= g - Mu + C(b)b - D^T r, \\
r_r &= -Db.
\end{aligned}
$$

The final challenge is to find an effective linear solver for this problem.

Chapter 4

Linear Solver

In many applications one of the main problems is to find the solution of a linear system. This system may arise form an optimisation or, like here, a discretised PDE. In this chapter we will go through the two methods which this is done.

4.1 Direct Methods

For

4.2 Iterative Methods

Chapter 5

Preconditioning

In this chapter we will go through the preconditioning strategies for Navier-Stokes, Maxwell's and finally the full MHD problem.

5.1 Navier-Stokes

5.1.1 Least Squares Commutator

5.1.2 Pressure Convection-Diffusion

5.2 Maxwell

5.3 MHD

5.3.1 Decoupling

No decoupling

Magnetics decoupling

Complete decoupling

Chapter 6

Results

In this chapter we will go through the preconditioning strategies for Navier-Stokes, Maxwell's and finally the full MHD problem.

6.1 Navier-Stokes

6.1.1 Least Squares Commutator

6.1.2 Pressure Convection-Diffusion

6.2 Maxwell

6.3 MHD

6.3.1 Decoupling

No decoupling

Magnetics decoupling

Complete decoupling

Chapter 7

Conclusion

Bibliography

[1] Martin S. Alnæs. *UFL: a Finite Element Form Language*, chapter 17. Springer, 2012.

[2] Martin S. Alnæs, Anders Logg, and Kent-Andre Mardal. *UFC: a Finite Element Code Generation Interface*, chapter 16. Springer, 2012.

[3] Martin S. Alnæs, Anders Logg, Kent-Andre Mardal, Ola Skavhaug, and Hans Petter Langtangen. Unified framework for finite element assembly. *International Journal of Computational Science and Engineering*, 4(4):231–244, 2009.

[4] Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software*, To appear, 2013.

[5] Satish Balay, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory, 2013.

[6] Satish Balay, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2014.

[7] Michael Heroux, Roscoe Bartlett, Vicki Howle Robert Hoekstra, Jonathan Hu, Tamara Kolda, Richard Lehoucq, Kevin Long, Roger Pawlowski, Eric Phipps, Andrew Salinger, Heidi Thornquist, Ray Tuminaro, James Willenbring, and Alan Williams. An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[8] Michael A. Heroux and James M. Willenbring. Trilinos Users Guide. Technical Report SAND2003-2952, Sandia National Laboratories, 2003.

[9] Robert C. Kirby. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software*, 30(4):502–516, 2004.

[10] Robert C. Kirby. *FIAT: Numerical Construction of Finite Element Basis Functions,*, chapter 13. Springer, 2012.

[11] Robert C. Kirby and Anders Logg. A compiler for variational forms. *ACM Transactions on Mathematical Software*, 32(3), 2006.

[12] Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. *FFC: the FEniCS Form Compiler*, chapter 11. Springer, 2012.

[13] Anders Logg and Garth N. Wells. Dolfin: Automated finite element computing. *ACM Transactions on Mathematical Software*, 37(2), 2010.

[14] Anders Logg, Garth N. Wells, and Johan Hake. *DOLFIN: a C++/Python Finite Element Library*, chapter 10. Springer, 2012.

[15] Jean-Claude Nédélec. Mixed finite elements in 3. *Numerische Mathematik*, 35(3):315–341, 1980.

[16] Kristian B. Ølgaard and Garth N. Wells. Optimisations for quadrature representations of finite element tensors through automated code generation. *ACM Transactions on Mathematical Software*, 37, 2010.