Implementation of quasi-2D magnetohydrodynamic mixed convection solver for incompressible flows in liquid metal channels

Eduardo Iraola de Acevedo

Department of Physics,
Technical University of Catalonia,
Barcelona, Spain

January 31, 2021

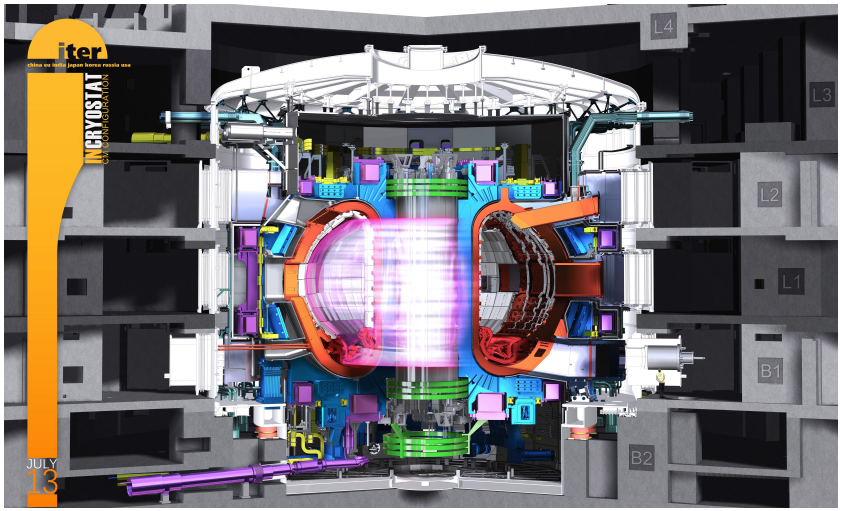
Table of Contents

- Introduction
- The existing solver `mhdFoam`
- The quasi-2D approximation and its implementation in the Q2DmhdFoam solver.
- Benchmark
- Conclusions

Context: Nuclear Fusion

Nuclear fusion reactors need breeding blankets around the core:

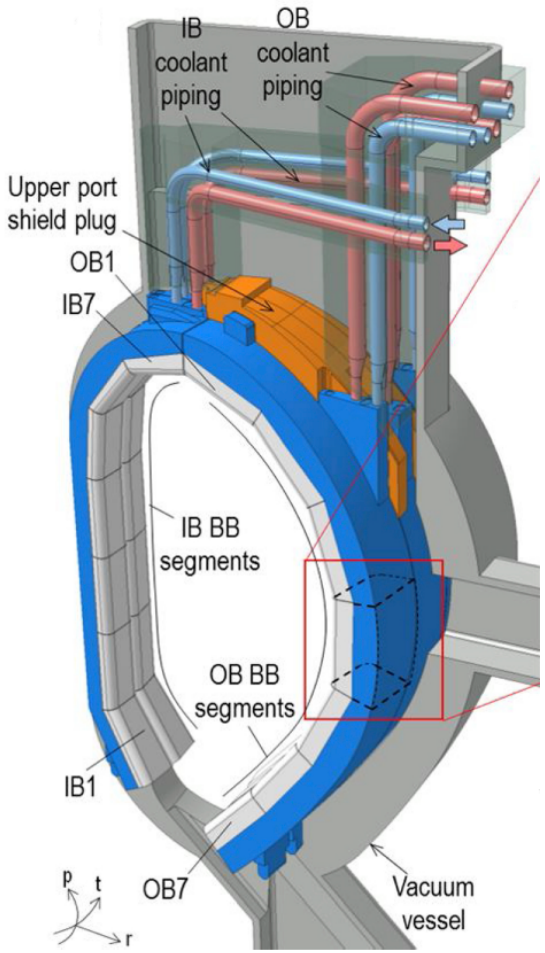
- To breed tritium, the fuel for the fusion reaction.
- To serve as a coolant (and transfer the resulting heat to produce electricity).
- To shield the external components from the radiation of the core.



Source: ITER.org

Context: Nuclear Fusion

- Some of the existing concepts for breeding blankets are liquid metal formed by a lead–lithium alloy.
- An example of these breeding blankets is the DCLL or Dual Coolant Lithium Lead, in which channels of liquid metal flow surrounding the reactor.
- Smaller channels with helium collaborate with the liquid metal in the coolant function.



Source: Federici et al. An overview of the EU breeding blanket design strategy as an integral part of the DEMO design effort. (2019).

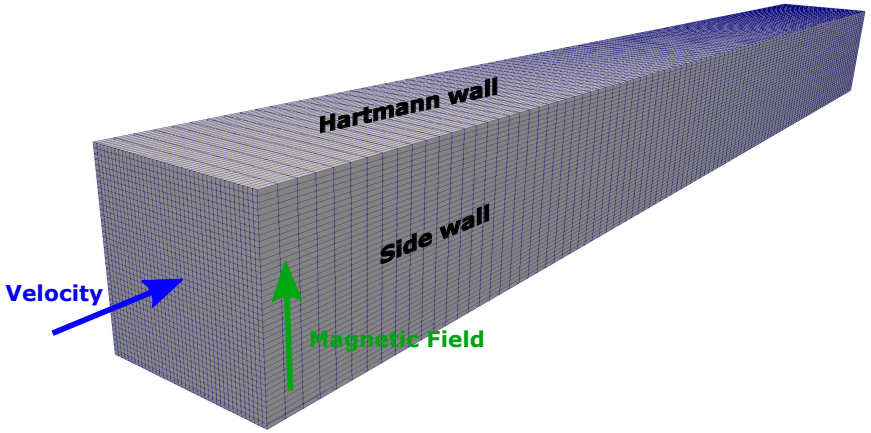
Context: Nuclear Fusion

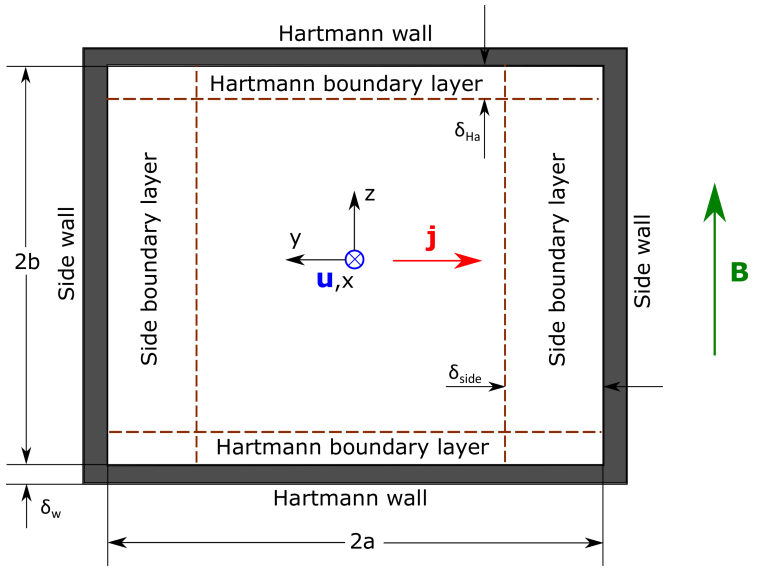
However, there is problem with liquid metals in fusion:

- The liquid metal bears free charge that is moving.
- The magnetic field of the reactor interacts with this charge.
- Magnetic forces appear. These forces modify the flow

New field of study:

Magnetohydrodynamics (MHD)
It analyzes the coupling between the magnetic field and the rest of the fields (velocity).





MHD: Governing equations

Conservation of Mass:

$$\nabla \cdot \boldsymbol{v} = 0$$

Conservation of Momentum:

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\boldsymbol{v} = -\frac{\nabla p}{\rho} + \nu \nabla^2 \boldsymbol{v} + \frac{1}{\mu_{\mathrm{m}} \rho}\left(-\nabla \frac{\boldsymbol{B}^2}{2} + (\boldsymbol{B} \cdot \nabla)\boldsymbol{B}\right) + \beta \boldsymbol{g}(T - T_{\mathrm{o}})$$

Gauss's law for magnetism:

$$\nabla \cdot \boldsymbol{B} = 0$$

Equation for the magnetic field:

$$\frac{\partial \boldsymbol{B}}{\partial t} = \nabla \times (\boldsymbol{v} \times \boldsymbol{B}) + \eta \nabla^2 \boldsymbol{B}$$

Energy equation:

$$\frac{\partial T}{\partial t} + (\boldsymbol{v} \cdot \nabla) T = \frac{k}{\rho c_p} \nabla^2 T + q_{\mathrm{o}} e^{-\left(\frac{y}{a}+1\right)m}$$

The mhdFoam solver

There is an implementation of the MHD equations in OpenFOAM 2006 and foam-extend 4.1

The solver is called mhdFoam.

(We will use foam-extend but similar procedures would apply to OpenFOAM)

- We will quickly check how these equations are implemented.
- We will run the tutorial of mhdFoam
- And we will modify it to see the boundaries of its domain in different directions.

The mhdFoam solver

The equations are implemented in the main file of the solver:

`$FOAM_SOLVERS/electromagnetics/mhdFoam/mhdFoam.C`

Momentum Equation

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvc::div(phiB, 2.0*DBU*B)
  - fvm::laplacian(nu, U)
  + fvc::grad(DBU*magSqr(B))
);

solve(UEqn == -fvc::grad(p));
```

We can see that:

- The term phiB is a way to adapt the implementation of the divergence.
- DBU=constant= $\frac{1}{2\mu_m\rho}$
- **No coupling with temperature**.

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v}\cdot\nabla)\boldsymbol{v} = -\frac{\nabla p}{\rho} + \nu\nabla^2\boldsymbol{v} + \frac{1}{\mu_m\rho}\left(-\nabla\frac{\boldsymbol{B}^2}{2} + (\boldsymbol{B}\cdot\nabla)\boldsymbol{B}\right)$$

The mhdFoam solver

Magnetic Field Equation

```
fvVectorMatrix BEqn
(
    fvm::ddt(B)
  + fvm::div(phi, B)
  - fvc::div(phiB, U)
  - fvm::laplacian(DB, B)
);

BEqn.solve();
```

$$\frac{\partial \boldsymbol{B}}{\partial t} = \nabla \times (\boldsymbol{v} \times \boldsymbol{B}) + \eta \nabla^2 \boldsymbol{B}$$

- There is a sort of magnetic diffusivity defined $\mathrm{DB} = 1/(\sigma \mu_{\mathrm{m}})$.
- The curl term is expanded into the two terms: $(\boldsymbol{B} \cdot \nabla)\boldsymbol{v}$ and $(\boldsymbol{v} \cdot \nabla)\boldsymbol{B}$. (Thanks to the fundamental laws: $\nabla \cdot \boldsymbol{B} = 0$ and $\nabla \cdot \boldsymbol{v} = 0$!)

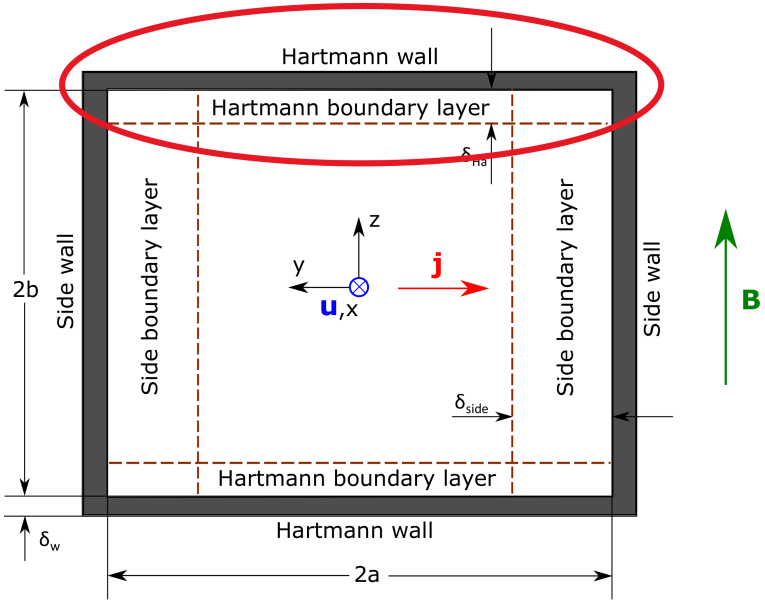
The mhdFoam tutorial: hartmann

We will now take a look at the hartmann tutorial. This is a case of 2D flow along two parallel plates. The plates represent the hartmann walls.

Copy the tutorial files into your run directory:

```
cd $FOAM_RUN
cp -r $FOAM_TUTORIALS/electromagnetics/mhdFoam/hartmann .
cd hartmann
```

Take a look at the file structure.



The mhdFoam tutorial: hartmann

Take a look at the file structure.

```
hartmann
├── 0
│   ├── B
│   ├── p
│   ├── pB
│   └── U
├── constant
│   ├── polymesh
│   │   └── blockMeshDict
│   └── transportProperties
└── system
    ├── controlDict
    ├── fvSchemes
    ├── fvSolution
    └── sampleDict
```

pB takes the role of a magnetic pressure, and is solved in a PISO algorithm (in the code: B-PISO) in a similar way to the normal pressure.

The mhdFoam tutorial: hartmann

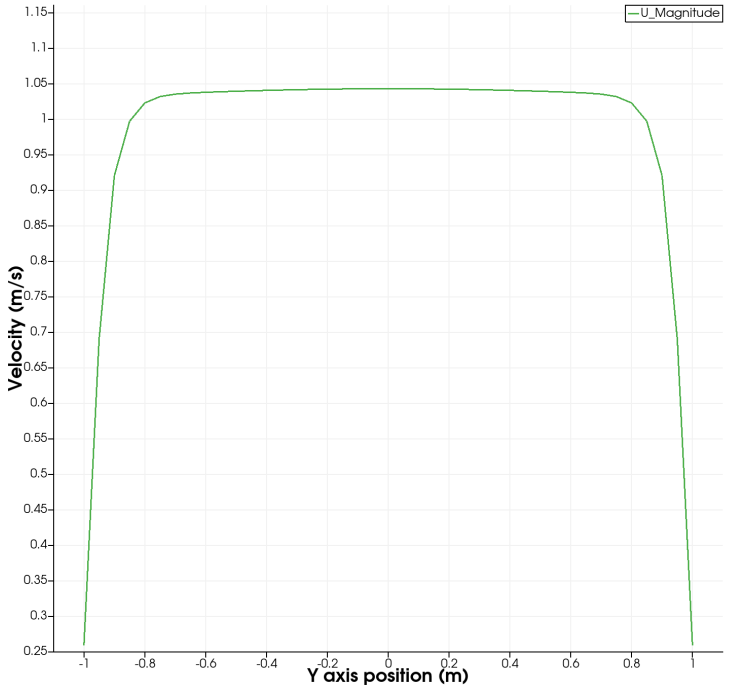
Construct the mesh and run the case:

```
blockMesh
mhdFoam
```

To plot the velocity profile we can do:

```
foamCalc components U
sample
#There is a sample dictionary
gnuplot
plot 'postProcessing/sets/2/line_centreProfile_Ux.xy'
```

We should see the flat profile typical of MHD flow with electrically isolating walls.



The mhdFoam tutorial: hartmann 3D

Now we will take advantage of the hartmann tutorial to set up a three-dimensional case:

```
cd $FOAM_RUN
cp -r $FOAM_TUTORIALS/electromagnetics/mhdFoam/hartmann hartmann3D
cd hartmann3D
```

Modified constant/polyMesh/blockMeshDict **file for the** hartmann3D **case.**

```
vertices
(
    (0 -1 -1)
    (20 -1 -1)
    (20 1 -1)
    (0 1 -1)
    (0 -1 1)
    (20 -1 1)
    (20 1 1)
    (0 1 1)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (100 40 40) simpleGrading (1 1 1)
);
```

The mhdFoam tutorial: hartmann 3D

And change the patch type in the 3rd dimension so that OpenFOAM knows that this is a 3D case.

```
edges
(
);

boundary
(
    ...
    frontAndBack
    {
        type patch;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);

mergePatchPairs
(
);
```

The mhdFoam tutorial: hartmann 3D

Now we change the boundary conditions of the fields in the time directory accordingly to the previous modifications.

Modified magnetic field file (0/B) for the hartmann3D case.

```
internalField   uniform (0 0 20);

boundaryField
{

    ...

    frontAndBack
    {
        type            fixedValue;
        value           uniform (0 0 20);
    }
}
```

(NOTE: we are changing the direction of the magnetic field to match it to the cases we'll see later.)

The mhdFoam tutorial: hartmann 3D

We now set a no-slip condition to the new walls and zeroGradient boundary conditions to the rest of the fields.

Modified frontAndBack velocity boundary condition (0/U).

```
boundaryField
{
    ...
    frontAndBack
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
}
```

Modified frontAndBack pressure boundary condition (0/p and 0/pB).

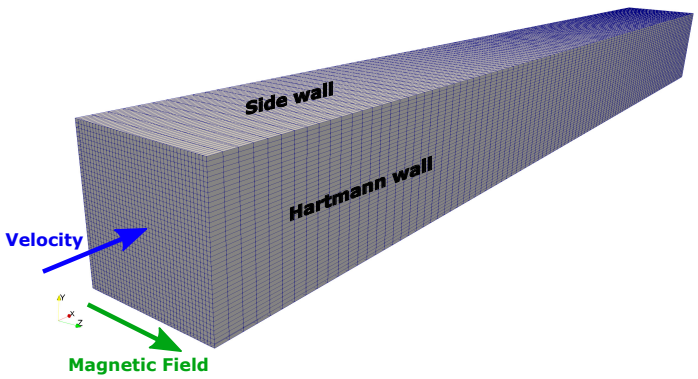
```
boundaryField
{
    ...
    frontAndBack
    {
        type            zeroGradient;
    }
}
```

The mhdFoam tutorial: hartmann 3D

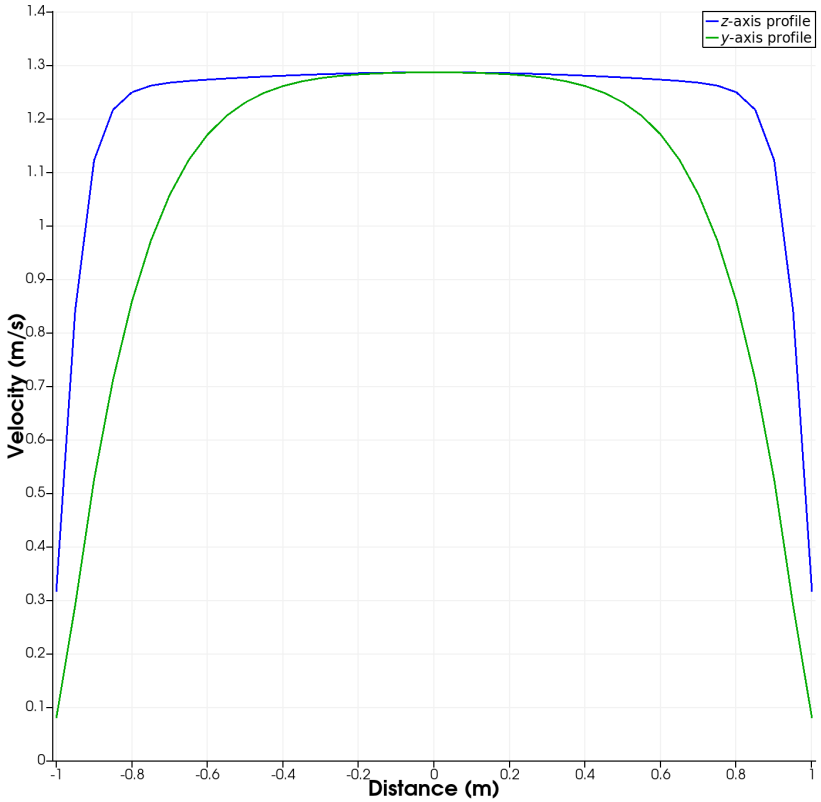
We are ready to mesh and run the case:

```
blockMesh
mhdFoam
paraFoam
```

The case can take several minutes (one processor).



- The side b.l. is $\parallel$ to $\boldsymbol{B}$.
- The hartmann b.l. is $\perp$ to $\boldsymbol{B}$. The velocity gradient is steeper there.



Summary of mhdFoam

What have we learnt so far?

- mhdFoam can be used to do rigorous simulations of MHD.
- mhdFoam doesn't account for temperature coupling.
  A passive scalar solver could be run over the solution of mhdFoam. But we cannot see the influence of the temperature in the velocity field.
- 3D simulations of mhdFoam can get expensive. A rigorous velocity–temperature–magnetic field solver would be even more expensive.
- Boundary layers have strong gradients in MHD. Thus they are thin and they need fine meshes.

New MHD solver

Let's say we are interested in a new solver that can fulfill the following requirements:

- MHD solver.
- Computationally cheap.
- Able to couple temperature with velocity.
- Able to include a built-in exponential heat deposition term without the need for preprocessing tools.

Sommeria and Moreau[1] showed a two-dimensional approach that can fit these expectations.
We will take a look at their approach.

[1] J. Sommeria and R. Moreau, "Why, how, and when, MHD turbulence becomes two-dimensional," J. Fluid Mech., vol. 118, no. May, pp. 507–518, 1982.

New MHD solver

In their work, Sommeria and Moreau assimilated the 3rd dimension in a liquid metal channel with the presence of magnetic fields to a two-dimensional problem. It works the following way:

- The removed dimension is that **parallel to the magnetic field**.
- The magnetic field equation is not solved. Now, the influence of the magnetic field is represented by a magnetic term in the **momentum equation**.

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\boldsymbol{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{v} - \frac{1}{\tau_{\mathrm{Ha}}}\boldsymbol{v}$$

- This new term is the one that contains the Lorentz force and slows down the flow, specifically in the core region (remember the flattened velocity profiles).
- The **Hartmann braking time**, τ_{Ha} contains the magnitude of the magnetic field.

$$\tau_{\mathrm{Ha}} = \frac{b}{B}\sqrt{\frac{\rho}{\sigma \nu}}$$

This approach is therefore called quasi-2D approximation. We are solving a 2D case, but accounting for certain influences from the 3rd dimension.

New MHD solver

Furthermore, the Boussinesq approach can be used to add the influence of temperature gradients in the velocity field.

The Boussinesq hypothesis lets us use a constant density ρ_o in all the terms of the momentum equation except in the buoyancy term. This way we can avoid the use of an equation of state.

The momentum and energy equations are coupled by the buoyancy term:

$$\beta \boldsymbol{g}(T - T_o)$$

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\boldsymbol{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{v} - \frac{1}{\tau_{\mathrm{Ha}}}\boldsymbol{v} + \beta \boldsymbol{g}(T - T_o)$$

The boussinesqBuoyantFoam solver implements this feature.
(`foam-extend 4.1`)

New MHD solver: Q2DmhdFoam

We will implement a new solver, called `Q2DmhdFoam`.

- The new solver will share features with `mhdFoam` and `boussinesqBuoyantFoam`.
- It will be based on the approximation from Sommeria and Moreau.
- It will account for a heat deposition term to simulate a heat source that decays exponentially, following the formula:

$$\dot{q} = q_o e^{-\left(\frac{y}{a}+1\right)m}$$

  (We will check the parameters meaning later)
- We will use the `boussinesqBuoyantFoam` as a starting point. It is a transient solver for buoyancy-driven laminar flow of incompressible flows using the Boussinesq Model.

Implementation of Q2DmhdFoam

We will start by copying our base solver, boussinesqBuoyantFoam, into the user directory:

```
foam
cp -r --parents applications/solvers/heatTransfer/boussinesqBuoyantFoam \
$WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR
```

Then, we can rename the directory according to the name of the new solver:

```
cd $WM_PROJECT_USER_DIR/applications/solvers/heatTransfer
mv boussinesqBuoyantFoam Q2DmhdFoam
cd Q2DmhdFoam
```

Implementation of Q2DmhdFoam

Now, we do all the steps to change the old name: `boussinesqBuoyantFoam` inside the files.

```
wclean
mv boussinesqBuoyantFoam.C Q2DmhdFoam.C
```

Check all the occurrences of the word `boussinesqBuoyantFoam`.

```
grep -r boussinesqBuoyantFoam
```

And perform the replacements.

```
sed -i s/boussinesqBuoyantFoam/Q2DmhdFoam/g Q2DmhdFoam.C
sed -i s/boussinesqBuoyantFoam/Q2DmhdFoam/g Make/files
```

At this point, we should have a functional solver named `Q2DmhdFoam` that has the same functionality as `boussinesqBuoyantFoam` solver. This can be checked by compiling the solver:

```
wmake
```

Implementation of Q2DmhdFoam

readTransportProperties.H

The `boussinesqBuoyantFoam` solver was not prepared for electromagnetic properties. We here will use the `readTransportProperties.H` file to read these properties from `transportProperties`.

Take a look at the current `readtransportProperties.H`.

We keep this part of the code and **modify** only the acquisition of the kinematic viscosity. We will make the user input the kinematic viscosity (instead of the dynamic viscosity to avoid confusion with magnetic permeability) as follows

```
dimensionedScalar nu
    (
        transportProperties.lookup("nu")
    );
```

Implementation of Q2DmhdFoam

readTransportProperties.H

Then we **add** the rest of the parameters to the file (see description in next slide):

```
dimensionedScalar Cp
(
    transportProperties.lookup("Cp")
);
dimensionedScalar a
(
    transportProperties.lookup("a")
);
dimensionedScalar b
(
    transportProperties.lookup("b")
);
dimensionedScalar sigma
(
    transportProperties.lookup("sigma")
);
dimensionedScalar tauHa
(
    b*sqrt(rho0/(sigma*nu))
);
dimensionedScalar q0
(
    transportProperties.lookup("q0")
);
dimensionedScalar m
(
    transportProperties.lookup("m")
);
scalar coord
(
    readScalar(transportProperties.lookup("coord")));
```

Implementation of Q2DmhdFoam

readTransportProperties.H

New parameters:

- `Cp`: fluid specific heat.
- `a`: half-length of the rectangular channel in the direction perpendicular to the magnetic field.
- `b`: half-length of the rectangular channel in the direction of the magnetic field.
- `sigma`: fluid electrical conductivity.
- `tauHa`: Hartmann braking time.
- `q0`: base heat deposition.
- `m`: exponential shape parameter.
- `coord`: parameter used to indicate the direction of the heat deposition.

The heat parameters apply to the following exponential term

$$\dot{q} = q_o e^{-\left(\frac{y}{a}+1\right)m}$$

that we will add to the energy equation.

Implementation of Q2DmhdFoam

createFields.H

The magnetic field must be initialized in `createFields.H`. We will add the next piece of code to this file.

Code added to `createFields.H` (I)

```
Info<< "Reading field B\n" << endl;
volScalarField B
(
    IOobject
    (
        "B",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

Implementation of Q2DmhdFoam

createFields.H

Also, the heat deposition source field must be calculated.

Code added to `createFields.H` (II)

```
Info<< "Calculating field sourceT\n" << endl;
// Infer the sourceT BCs from the pressure (set all to zeroGradient)
wordList sourceTBCTypes
(
    p.boundaryField().size(),
    zeroGradientFvPatchScalarField::typeName
);
volScalarField y = mesh.C().component(vector::Y);
if (coord == 1)
{
    y = mesh.C().component(vector::X);
}
if (coord == 2)
{
    y = mesh.C().component(vector::Y);
}
if (coord == 3)
{
    y = mesh.C().component(vector::Z);
}
```

Implementation of Q2DmhdFoam

createFields.H

Also, the heat deposition source field must be calculated.

Code added to `createFields.H` (III)

```
volScalarField sourceT
(
    IOobject
    (
        "sourceT",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    q0*exp(- (y/a + 1) * m),
    sourceTBCTypes
);
```

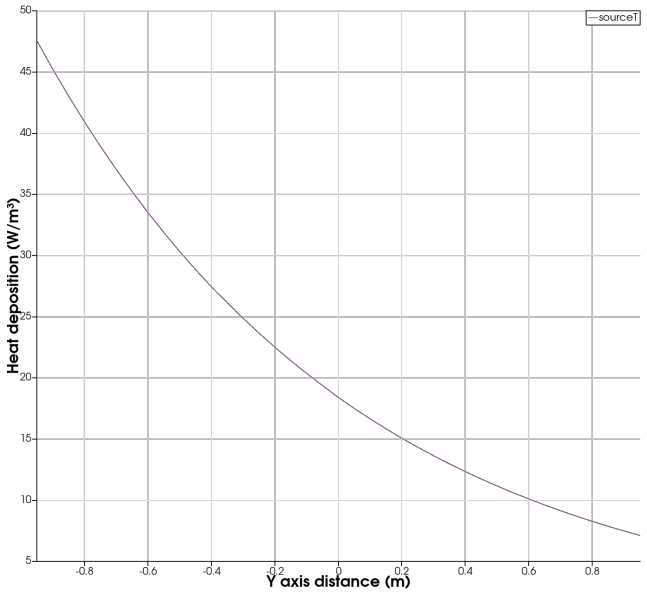
Implementation of Q2DmhdFoam

createFields.H

In the previous slides:

- We initialized the heat deposition field, `sourceT`.
- We configured its boundary conditions through the `sourceTBCTypes`.
- We determined the direction of the deposition through the variable `coord`.
- And calculated the heat deposition throughout the domain according to the formula

$$\dot{q} = q_o e^{-\left(\frac{y}{a}+1\right)m}$$



Implementation of Q2DmhdFoam

Main file: Q2DmhdFoam.C

The configuration of the main file mainly consists in adding the magnetic term to the momentum equation and the heat deposition term to the energy equation.

The momentum equation is modified as follows:

Momentum equation in `Q2DmhdFoam.C`.

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
 ==
   -beta*(T - T0)*g
   -fvm::Sp(mag(B)/tauHa, U)
);

solve(UEqn == -fvc::grad(p));
```

Implementation of Q2DmhdFoam

Main file: Q2DmhdFoam.C

The energy equation is modified as follows:

Energy equation in `Q2DmhdFoam.C`

```
// Solve energy equation
solve
(
    fvm::ddt(T)
  + fvm::div(phi, T)
  - fvm::laplacian(DT, T)
  - sourceT/(rho0*Cp)
);
```

Implementation of Q2DmhdFoam

Main file: Q2DmhdFoam.C

Finally, we want to add the following include files to allow adjustable time step depending on the Courant number:

Extra included files in `Q2DmhdFoam.C`

```
int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
    pisoControl piso(mesh);
#   include "readTransportProperties.H"
#   include "readGravitationalAcceleration.H"
#   include "createFields.H"
#   include "initContinuityErrs.H"
#   include "createTimeControls.H"
#   include "CourantNo.H"
#   include "setInitialDeltaT.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;
    while (runTime.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

#       include "readTimeControls.H"
#       include "CourantNo.H"
#       include "setDeltaT.H"
...
```

Implementation of Q2DmhdFoam

At this point, we can compile the code with the `wmake` command and verify everything is correct.

Configuring a Q2DmhdFoam case

The first tutorial case will come from the combination of the `mhdFoam` tutorial: the `hartmann` tutorial, and the `boussinesqBuoyantFoam` case: the `heatedCavity` case, tuned to fit the new solver.

```
cp -r $FOAM_TUTORIALS/heatTransfer/boussinesqBuoyantFoam/heatedCavity/ \
liquidMetalChannel
cd liquidMetalChannel

cp -r \
$FOAM_TUTORIALS/electromagnetics/mhdFoam/hartmann/constant/polyMesh/blockMeshDict \
constant/polyMesh
blockMesh
```

The generated mesh must be like the one for the `hartmann` tutorial.

Configuring a Q2DmhdFoam case

Configuration of the 0 directory.

```
cp $FOAM_TUTORIALS/electromagnetics/mhdFoam/hartmann/0/B 0/B
cp $FOAM_TUTORIALS/electromagnetics/mhdFoam/hartmann/0/U 0/U

sed -i s/top/inlet/g 0/*
sed -i s/bottom/outlet/g 0/*
sed -i s/left/upperWall/g 0/*
sed -i s/right/lowerWall/g 0/*
```

Configuring a Q2DmhdFoam case

The pressure file, 0/p needs further modifications.

Initial conditions in the pressure file.

```
...
boundaryField
{
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

Configuring a Q2DmhdFoam case

And we will also modify the temperature values.

Initial conditions in the temperature file.

```
...
boundaryField
{
    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            fixedValue;
        value           uniform 0;
    }

    lowerWall
    {
        type            fixedValue;
        value           uniform 0;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Configuring a Q2DmhdFoam case

Then we can add the necessary physical properties to `transportProperties`.

`transportProperties` file after modifications.

```
/*---------------------------------- C++ ----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.1                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

rho0            rho0  [1 -3 0 0 0 0 0] 1.0;
nu              nu  [0 2 -1 0 0 0 0] 1.0;
T0              T0  [0 0 0 1 0 0 0] 0.0;
Cp              Cp  [0 2 -2 -1 0 0 0] 1;
k               k  [1 1 -3 -1 0 0 0] 0.0001;
beta            beta  [0 0 0 -1 0 0 0] 1e-3;
a               a  [0 1 0 0 0 0 0] 1;
b               b  [0 1 0 0 0 0 0] 1;
sigma           sigma  [-1 -3 3 0 0 2 0] 1;
q0              q0  [1 -1 -3 0 0 0 0] 0;
m               m  [0 0 0 0 0 0 0 ] 1;
coord           2;  //X_axis:1, Y_axis:2, Z_axis:3

// ************************************************************************* //
```

Configuring a Q2DmhdFoam case

We temporarily remove gravity from the simulation.

g file

```
/*---------------------------------* C++ *---------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.1                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -2 0 0 0 0];
value           ( 0 0 0 );
// ************************************************************************* //
```

Configuring a Q2DmhdFoam case

Regarding the `controlDict` file, we will copy that of the `hartmann` tutorial and change its default solver to `Q2DmhdFoam`.

```
cp $FOAM_TUTORIALS/electromagnetics/mhdFoam/hartmann/system/controlD
system/controlDict
sed -i s/mhdFoam/Q2DmhdFoam/g system/controlDict
```

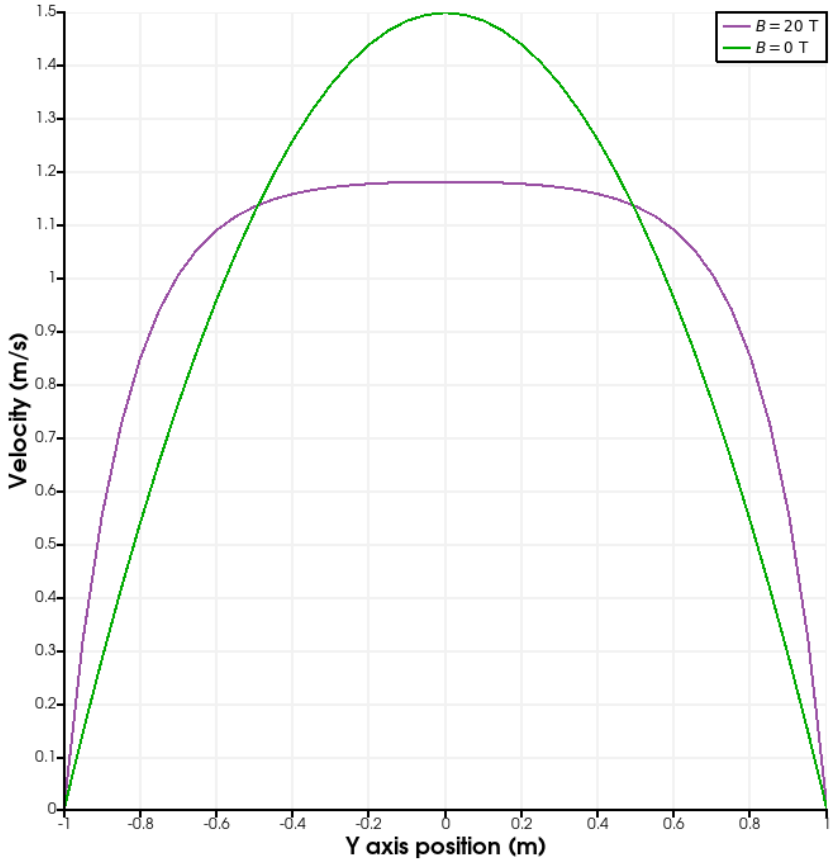
We are also interested in changing the laplacian scheme in `system/fvSchemes`.

```
sed -i s/"(mu|rho0)"/nu/g system/fvSchemes
```

The case is set up and the `Q2DmhdFoam` solver can be run. In the results, it shall be seen how the typical parabolic laminar profile is flattened in the presence of a magnetic field.
The next slide shows a comparison between the velocity profile with the configuration so far (purple) and the same case with no magnetic field (green).

Configuring a Q2DmhdFoam case

Comparison between the velocity profile with the configuration so far (purple) and the same case with no magnetic field (green).

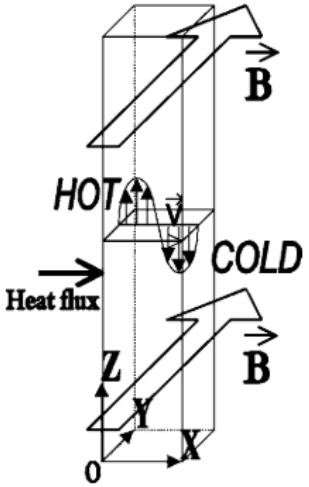


Benchmark: Tagawa

This section proposes a benchmark based on the work of Tagawa et al.[2] for `Q2DmhdFoam`.

Benchmark case: infinitely long vertical square duct

- 15 cm of side length.
- Walls with fixed temperatures
- The only driving force is the thermal stress.
- The flow is assumed to reach steady state.
- The magnetic and temperature constraints are input using the Hartmann and Grashof numbers, respectively



$$\mathrm{Ha} = Bb\sqrt{\frac{\sigma}{\mu}} \qquad \mathrm{Gr} = \frac{g\beta\Delta T a^3}{\nu^2}$$

[2] T. Tagawa, G. Authié, and R. Moreau, "Buoyant flow in long vertical enclosures in the presence of a strong horizontal magnetic field. Part 1. Fully-established flow," Eur. J. Mech. B/Fluids, vol. 21, no. 4, pp. 383–398, 2002.

Benchmark: case configuration

To reproduce the benchmark, we are going to configure a case that can be easily run for any pair of input parameters (Ha and Gr).

- The case is generalized using tags (see next slide).
- A python script automates
  - pre-processing
  - meshing and running. **The mesh is customized according to Ha** (mainly the magnitude of the magnetic field).
  - post-processing
- The geometry consists on an enclosure long in proportion with its width (200:1).
- The benchmark is done at the center of the domain, where the influence of both ends is lower .

Benchmark: case configuration

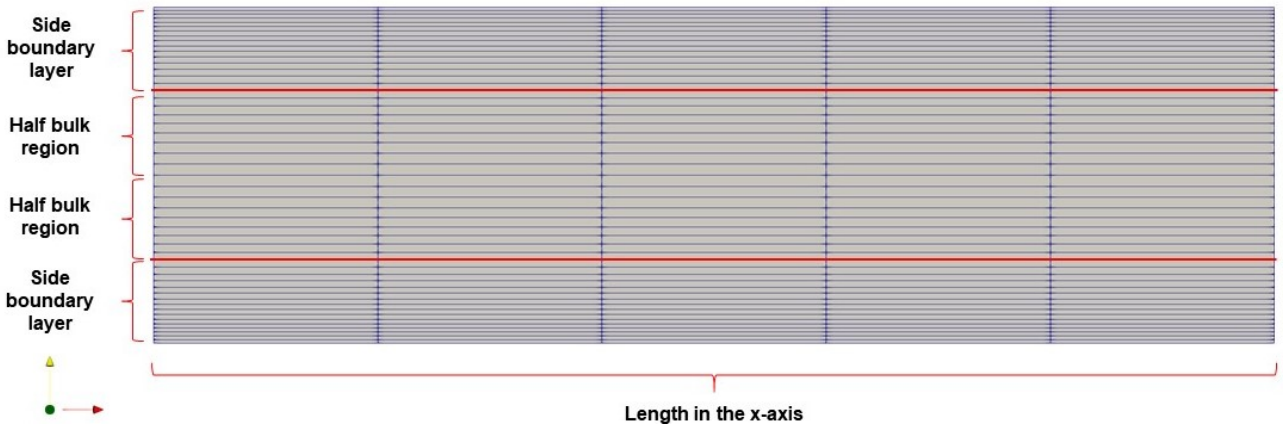
Accompanying files: The directory `baseCaseTagawa` contains all the necessary tagged files. `constant/transportProperties` is shown as an example:

```
rho0        rho0 [1 -3 0 0 0 0 0] /rho0/;
nu          nu [0 2 -1 0 0 0 0] /nu/;
T0          T0 [0 0 0 1 0 0 0] /T0/;
Cp          Cp [0 2 -2 -1 0 0 0] /Cp/;
k           k [1 1 -3 -1 0 0 0] /k/;
beta        beta [0 0 0 -1 0 0 0] /beta/;
a           a [0 1 0 0 0 0 0] /a/;
b           b [0 1 0 0 0 0 0] /b/;
sigma       sigma [-1 -3 3 0 0 2 0] /sigma/;
q0          q0 [1 -1 -3 0 0 0 0] /q0/;
m           m [0 0 0 0 0 0 0 ] /m/;
```

Benchmark: case configuration

Mesh procedure: The mesh is separated in the bulk region and the boundary layer region. The boundary layer region is set up so that there is always enough cells to represent the gradients of the MHD flow. The higher the Ha, the thinner the boundary layer.

```
x          /Lx/;
y          /LyBulk/;
yNeg       /LyNegBulk/;
yBL        /LyBL/;
yNegBL     /LyNegBL/;
Nx         /Nx/;
Ny         /NyBulk/;
NyBL       /NyBL/;
Gy         /GyBulk/;
GyInv      /GyBulkInv/;
GyBL       /GyBL/;
GyBLinv    /GyBLinv/;
vertices
(
    (0     $yNeg     0)
    ($x    $yNeg     0)
    ($x    $yNegBL   0)
    (0     $yNegBL   0)
    (0     $yNeg     0.1)
    ($x    $yNeg     0.1)
    ($x    $yNegBL   0.1)
    (0     $yNegBL   0.1)
    ...
)
```



Benchmark: Results

For running the case, type:

```
./meshAndGo.py
```

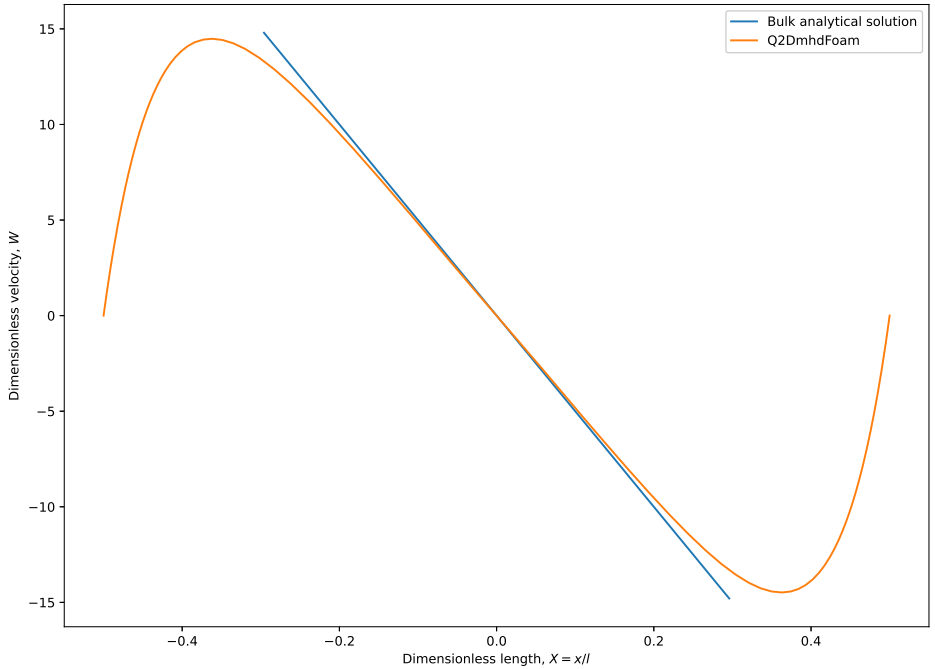
(meshAndGo.py is set up as an executable file)

Or:

```
python meshAndGo.py
```

The output plots the benchmark of the dimensionless velocity W for the **bulk region**.

$$W = v\frac{l}{\nu}$$



Conclusion

In this presentation:

- We have learnt the effects of magnetic fields in electrically conducting fluids and how to analyze them through MHD.
- We have reviewed an MHD solver, `mhdFoam`, implemented in the latest `OpenFOAM` and `foam-extend` distributions.
- We have reviewed the quasi-2D approximation for analyzing MHD flows, and applied it in a brand new solver, called `Q2DmhdFoam`, implemented step by step, and checked how to use it.

Check out the examples and updates in `GitHub`:
https://github.com/iraola/Q2DmhdFoam.

Thank you for your attention!