Research project: Why is the aurora only visible at the earth's magnetic poles?

DD2365 Advanced Computation in Fluid Mechanics

Markus Renoldner

January 19, 2023

Abstract

In this report, we discuss the dynamics of the solar wind - a plasma - under the influence of the geomagnetic field. This field is approximated by a dipole model.
We simulate a diffusion process as well as a more complex Navier-Stokes problem. Starting in 2D and extending to 3D, we can observe that the Lorentz force induced by a locally "flat" (i.e. parallel vectors in the whole domain) magnetic field creates a circular/cylindrical vortex flow pattern.

As a last step, to make the simulation more realistic, a curved magnetic field is modeled, approximating the dipole assumption. This leads to a plasma deflection, that is stronger closer to the equator and weaker close to the poles.
The observations allow the following hypothesis: this feature of the curved field allows more particles that are trapped in the flow vortices to get transported to the poles.

Contents

References

[CT16] Francis F Chen and Humberto Torreblanca. *Introduction to Plasma Physics and controlled fusion, third edition*. 2016 (cit. on pp. 2, 11).

[Hof22] Johan Hoffman. *Lecture notes Advanced Computations in Fluid Mechanics*. Kungliga Tekniska Högskolan, 2022 (cit. on p. 5).

[Wal94] Martin Walt. *Introduction to Geomagnetically Trapped Radiation*. Cambridge Atmospheric and Space Science Series. Cambridge University Press, 1994. DOI: 10.1017/CBO9780511524981 (cit. on p. 4).

1 Introduction

The aurora borealis and aurora australis is a visual phenomenon over the earth's magnetic poles, that is caused by solar wind interacting with the earths atmosphere. Solar wind consists of ionized, i.e. electrically charged particles that come from the corona, which is the top layer of the sun's atmosphere. This particle stream is in the so-called plasma state, see [CT16].

The charged particle stream experiences a force when it interacts with the earth's electric and magnetic fields, the Lorentz force

$$\mathbf{F}_L = q(\mathbf{E} + \mathbf{u} \times \mathbf{B}).$$

As one can see, the force acts parallel to the **E**-field but perpendicular to the **B**-field as well as the velocity vector **u**. The force on one particle is proportional to its electric charge q.

The earth's electric field **E** is approximately a central force field and does not explain the movement of particles to the magnetic poles of the planet. In this report, the behaviour of **u** in the **E**-field are being neglected. Furthermore, the gravitational field, that also acts on the particles is not being taken into account, as it is also just a central force field.

As the force generated by the term $q\mathbf{u} \times \mathbf{B}$ is perpendicular to **u**, it is not immediately clear, why a substantial amount of particles is being transported to the magnetic poles at all.

The perpendicular force on a particle will lead to a more or less circular path in the **B**-field. Depending on the initial velocity, the electric charge and the direction and magnitude of the **B** field, the radius and change of the particle speed will vary. Circling particles in the **B**-field are called "magnetically trapped".

The field lines, i.e. the lines tangential to the local field vector get denser at the magnetic poles. This might be the reason, why more trapped particles end up at the polar region.
But again, the Lorentz force on each particle is perpendicular to its velocity vector, which is why it is not self explanatory what causes the particle transport in the direction of the **B**-field.

2 Method

2.1 Conservation of momentum and mass

Assuming the plasma consists only one type of particles with a constant charge as well as modelling the particle stream as a continuous fluid, the equation of motion for one of these charged particles is

$$m\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = \sum_i \mathbf{F}_i = \mathbf{F}_E + \mathbf{F}_B.$$

The forces due to the electric and magnetic field are $\mathbf{F}_E = q\mathbf{E}$ and $\mathbf{F}_B = q\mathbf{u} \times \mathbf{B}$. With that one gets

$$m\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = q(\mathbf{E} + \mathbf{u} \times \mathbf{B})$$

Taking into account internal stresses and applying the material derivative, one obtains

$$m\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = q(\mathbf{E} + \mathbf{u} \times \mathbf{B}) - \nabla \cdot \sigma$$

where σ denotes the stress tensor.

As discussed, earths **E**-field will be neglected. Furthermore we assume that the **B**-field is constant and not influenced by **u**. All constants are being set to 1. The momentum equation (describing conservation of momentum) becomes

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = \mathbf{u} \times \mathbf{B} - \nabla \cdot \sigma \tag{1}$$

In addition to that, one has to solve the continuity equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{2}$$

which describes the conservation of mass. For an incompressible flow, one yields the divergence free condition

$$\implies \nabla \cdot \mathbf{u} = 0 \tag{3}$$

To summarize, the following equations describe the flow of charged particles in a known constant **B**-field:

$$\begin{cases} \dot{\mathbf{u}} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \mathbf{u} \times \mathbf{B} + \nabla \cdot \sigma = \mathbf{0} \\ \dot{\rho} + \nabla \cdot (\rho \mathbf{u}) = 0 \end{cases} \tag{4}$$

with $\rho = \rho : \mathbb{R}^3 \times \mathbb{R}^+ \to \mathbb{R}$, $\mathbf{u} = \mathbf{u} : \mathbb{R}^3 \times \mathbb{R}^+ \to \mathbb{R}^3$ and $\mathbf{B} = \mathbf{B} : \mathbb{R}^3 \times \mathbb{R}^+ \to \mathbb{R}^3$.
The stress tensor field is a multi-linear mapping $\sigma = \sigma : \mathbb{R}^{3\times 3} \to \mathbb{R}$ for each point in time. In case of imcompressible flows, we have

$$\sigma = -p\mathbf{I} + 2\mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}} \right)$$

For vanishing viscosity, also called inviscid flow (i.e. for high Reynolds numbers), the momentum equation becomes

$$\dot{\mathbf{u}} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \mathbf{u} \times \mathbf{B} + \nabla p = \mathbf{0} \tag{5}$$

also called the Euler equation.

2.2 The B-field

In order to compute the **B**-field, one has to solve the maxwell equations

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0} \tag{6}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{7}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{8}$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} \tag{9}$$

or by assuming $\mathbf{E} = 0$ rather only equations (7) and (9). Here, ρ is not the density of the medium, but the electric charge density of the volume in which one analyses the $\mathbf{E}$-field. ε_0, μ_0 and c are constants.

Instead of solving these equations, we approximate the field by applying the dipole-assumption, yielding the following expressions in polar coordinates, see [Wal94]:

$$B_r = \frac{2B_0}{r^3}\cos(\theta) \tag{10}$$

$$B_\theta = \frac{B_0}{r^3}\sin(\theta) \tag{11}$$

where $\mathbf{B} \equiv (B_r, B_\theta, 0)^\mathsf{T} \in \mathbb{R}^3$.

The following plots visualize these equations approximately.

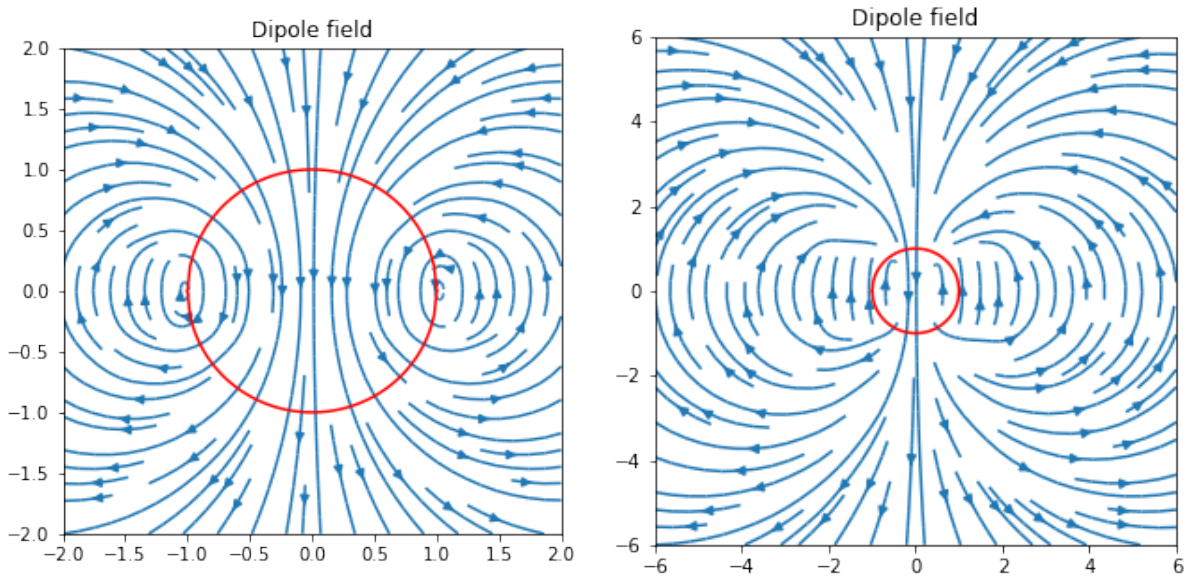

Figure 1: Earth's $\mathbf{B}$-field

The red circle represents the surface of the planet. Now one can clearly see, the field lines get denser at the poles. and are roughly parallel to the planet at the equator.
Even though the axis through earth's magnet poles is tilted with respect to the normal vector of its orbital plane, the Aurora is visible at both poles, so this feature of the field is not being modelled.

One effect that is neglected when assuming a stationary $\mathbf{B}$-field is the influence from the fluid on the field. The plasma consists of electrically charged particles, that inherently create their own $\mathbf{E}$-field, as the charge density $\rho \neq 0$. Of course this field is not constant in time, as the particles are moving. From equation (9) follows, that if $\frac{\partial \mathbf{E}}{\partial t}$ is not constant, the $\mathbf{B}$-field is also changing in time.

2.3 Boundary conditions

The system (4) will satisfy the following boundary conditions

$$\begin{cases} \mathbf{u} = \mathbf{u}_{in} & \mathbf{u} \in \Gamma_{in} \\ p = 0 & p \in \Gamma_{out} \end{cases}$$

with $\Gamma_{in} \cup \Gamma_{out} \equiv \Gamma = \partial\Omega$ and where $\Omega \subset \mathbb{R}^3 \times \mathbb{R}^+$ denotes the domain with boundary $\partial\Omega$.
The inflow Dirichlet condition corresponds to the solar wind coming from one side and the pressure 0 condition models adjacent free space, where the fluid can escape.

2.4 Variational formulation

In order to find a finite element approximation to the system derived above, we formulate the variational formulation, i.e. we want to find $(\mathbf{u}, p) \in V \times Q$, the respective finite dimensional approximation spaces to the Sobolev spaces of the original problem formulation .
We get

$$(\dot{\mathbf{u}} + (\mathbf{u} \cdot \nabla)\mathbf{u}, v) - (p, \nabla \cdot v) + (\nu \nabla \mathbf{u}, \nabla v) + (\nabla \cdot \mathbf{u}, q) + SD(\mathbf{u}, p; v, q) = (\mathbf{F}_B, v),$$

for all test functions $(v, q) \in \hat{V} \times \hat{Q}$.

Here $SD(\mathbf{u}, p; v, q)$ is a residual based stabilization term and

$$(x, y) := \int_\Omega xy \mathrm{d}x$$

denotes the inner product, see [Hof22].
To find the variational formulation of the Lorentz term, one multiplies with a testfunction and integrates over the domain Ω and obtains

$$(\mathbf{F}_B, \mathbf{v}) = (q(\mathbf{u} \times \mathbf{B}), \mathbf{v}) = \int_\Omega q(\mathbf{u} \times \mathbf{B}) \cdot \mathbf{v} \ \mathrm{d}x$$

2.5 Simulation cases

Just by considering the magnetically induced part of the Lorentz-force

$$\mathbf{F}_B = q(\mathbf{u} \times \mathbf{B})$$

one can see, that the vector valued variables $\mathbf{F}, \mathbf{u}, \mathbf{B}$ and therefore the derived system of equations can not be computed in $\mathbb{R}^2$, which increases the computational cost significantly.

One idea to overcome this problem and to avoid doing a 3D simulation is to simulate a 2D cross-section with $\mathbf{F}$ and $\mathbf{u}$ being in the plane of the domain. $\mathbf{B}$ would then be normal to the plane. In order to realize that, one could implement the force as a function of $\mathbf{u}$ - again assuming a constant $\mathbf{B}$.

To visualize the effect of the $\mathbf{B}$-field, the equations (diffusion and Navier-Stokes) were solved for a field with gaussian magnitude, so that the difference between areas with higher and lower Lorentz force are visible.

$$\begin{aligned}
\mathbf{B_x}(x, y) &= 0 \\
\mathbf{B_y}(x, y) &= 0 \\
\mathbf{B_z}(x, y) &= C_1 \left(e^{-C_2(x-0.5)^2 - C_2(y-0.5)^2} \right)
\end{aligned}$$

The z component is the direction perpendicular to the plane, while $\mathbf{u}$ and $\mathbf{F}_B$ are vector fields in the x-y-plane. The $\mathbf{B}$-field is visualized in the following plots.

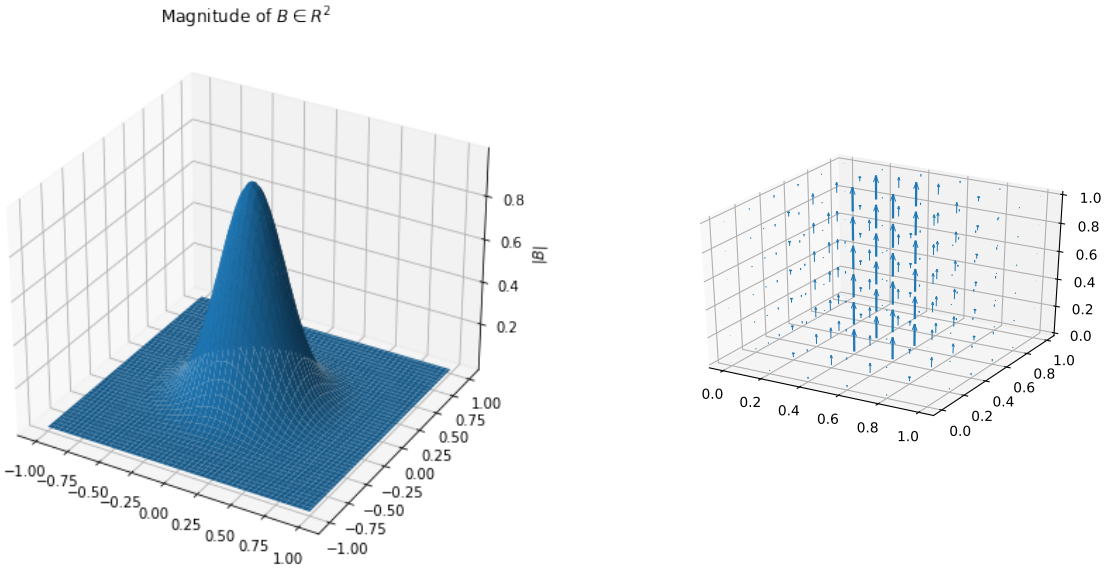

Figure 2: Left: magnitude of **B**-field, right: **B**-field in 3D

After trying to simulate the effect of a **B**-field on a fluid in 2D, we can try to expand this into a 3D simulation. Again we expect circular stream-lines/particle paths. In case of a constant magnetic field, the simulation should develop cylindrical vortices in the plasma. The **B**-field looks exactly like in the 2D case.

Last but not least, we will try to implement a more realistic **B**-field and hope to observe fluid motion along the field lines (perpendicular to the Lorentz force). In the following plot, this is visualized. The inflow boundary is on the left.

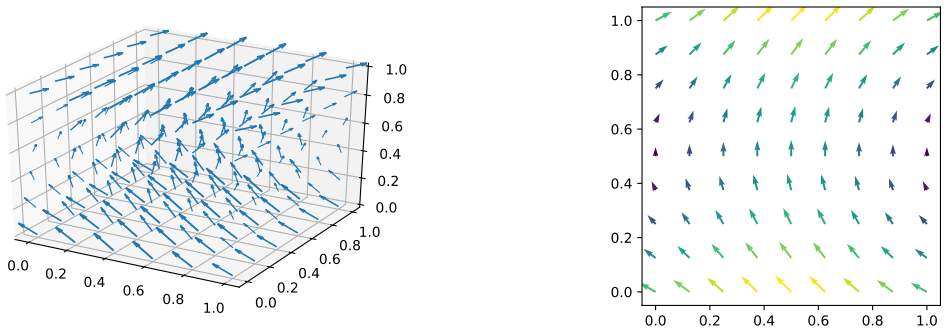

Figure 3: Curved **B**-field, left: cross-section

For simplicity, the first simulation in 2D and 3D will just be a modified diffusion equation

$$\dot{\mathbf{u}} - \Delta \mathbf{u} - \mathbf{u} \times \mathbf{B} = 0 \tag{12}$$

then we look into the full Navier-Stokes problem (4).

Summary of the different simulation cases:

1. Diffusion in 2D
2. Navier Stokes in 2D
3. Diffusion in 3D

4. Navier Stokes in 3D
5. Navier Stokes in 3D with realistic **B**-field

3 Results

2D diffusion

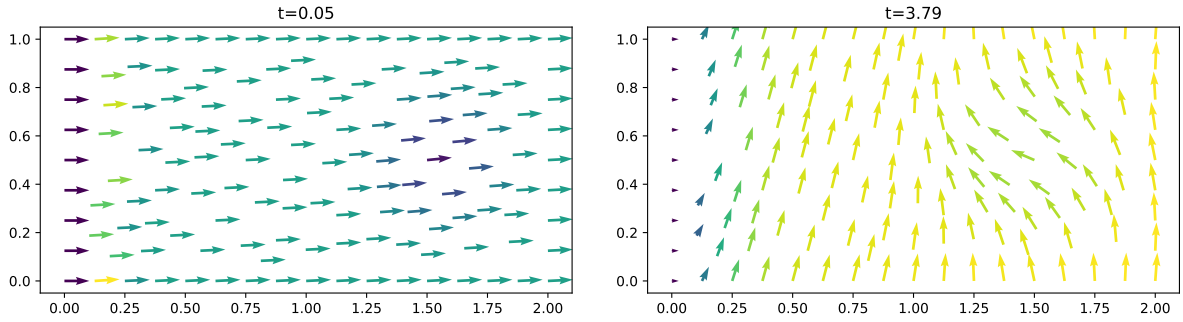

Figure 4: 2D diffusion

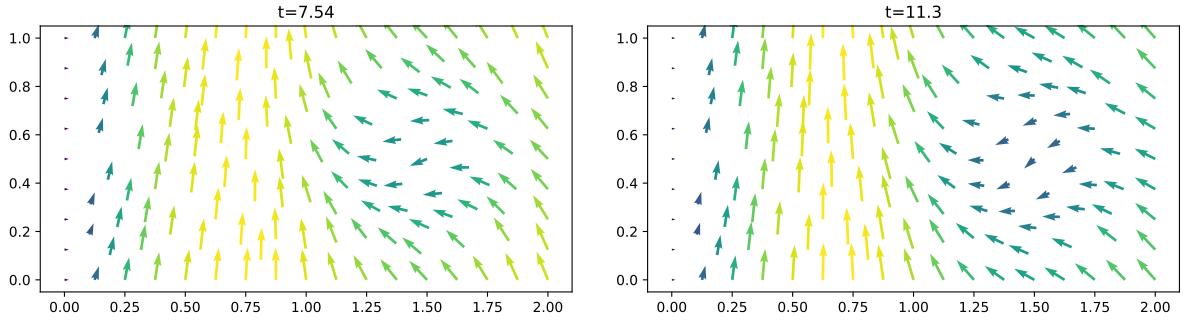

Figure 5: 2D diffusion

2D Navier-Stokes

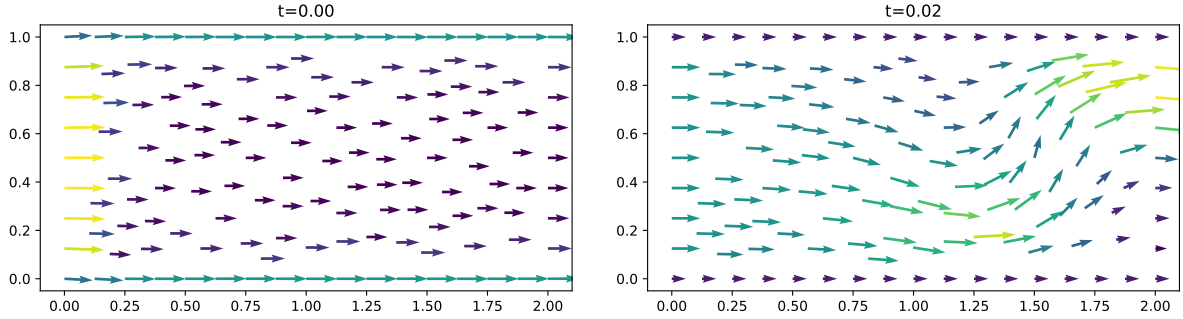

Figure 6: 2D Navier-Stokes

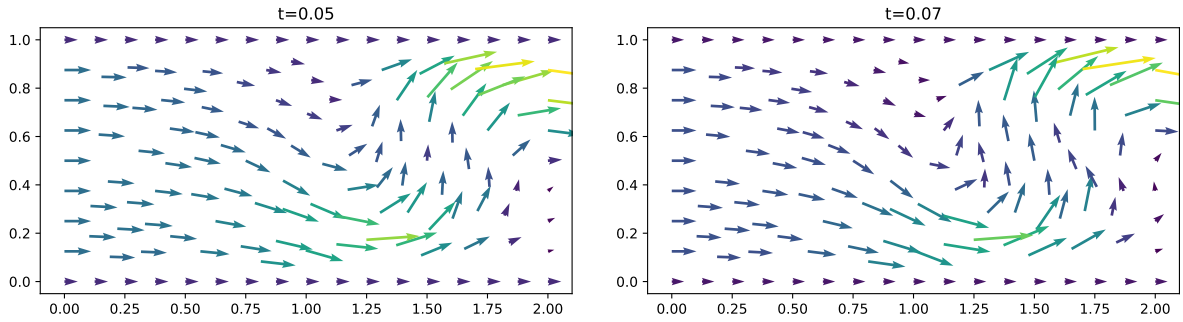

Figure 7: 2D Navier-Stokes

3D diffusion

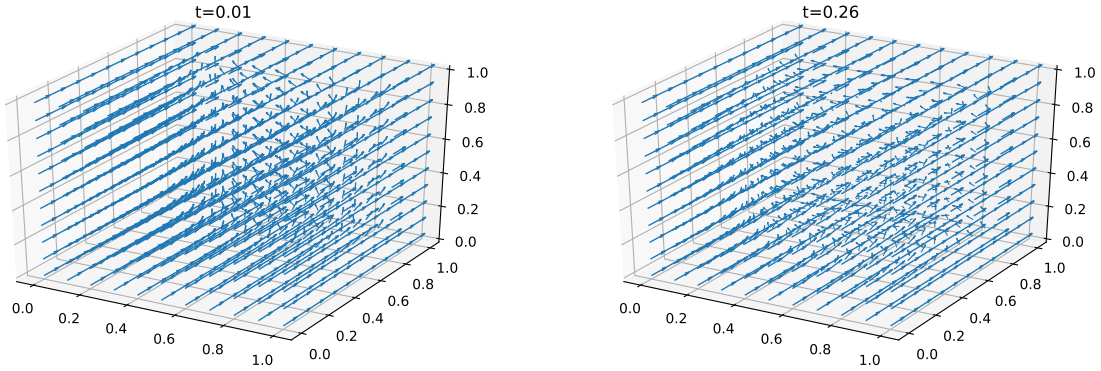

Figure 8: 3D diffusion

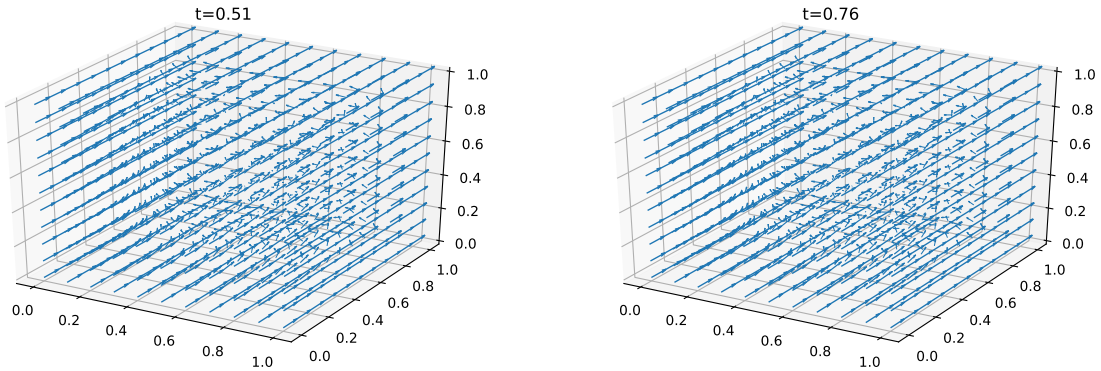

Figure 9: 3D diffusion

3D Navier-Stokes

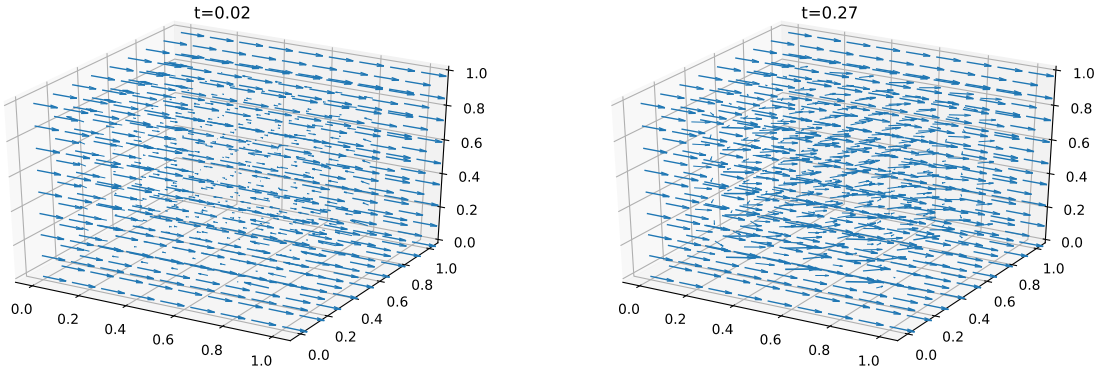

Figure 10: 3D Navier-Stokes

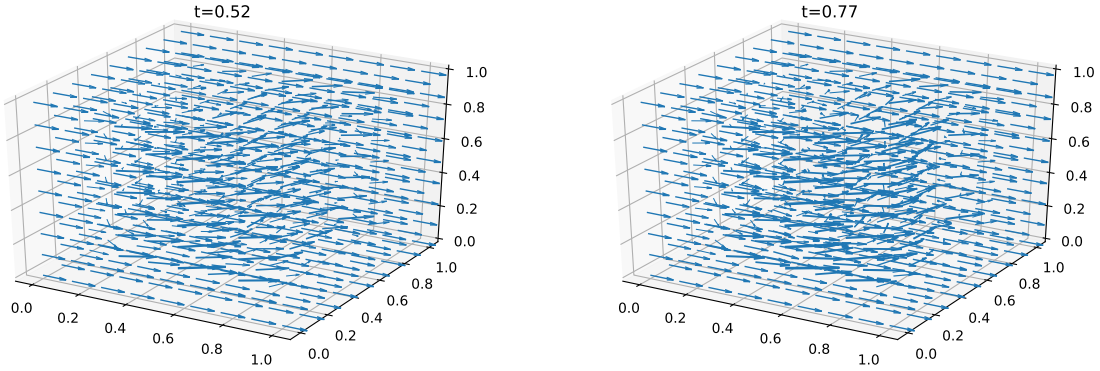

Figure 11: 3D Navier-Stokes

3D Navier-Stokes with curved B-field

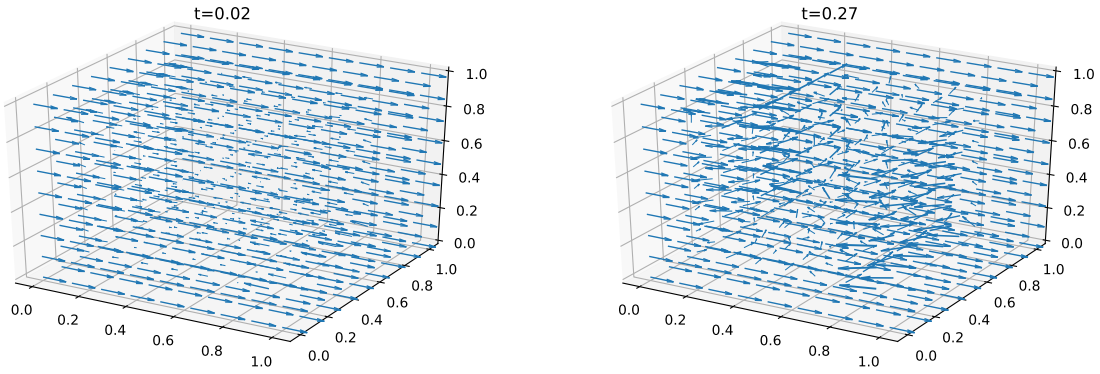

Figure 12: 3D Navier-Stokes with curved **B**-field

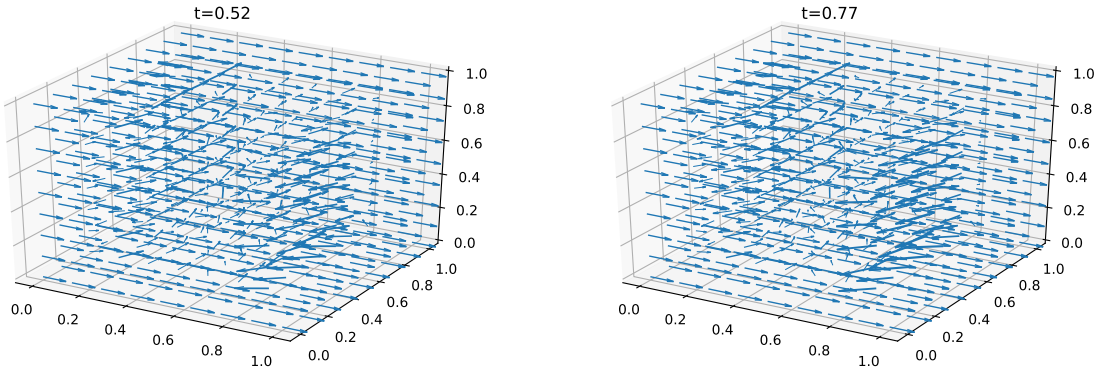

Figure 13: 3D Navier-Stokes with curved **B**-field

4 Discussion

4.1 Observations

1. In the first two 2D simulation cases, we could verify the expected behaviour of a plasma in a **B**-field. I.e. we see circular particle motion, where the radius and speed is proportional to the magnitude of the magnetic field.

2. In the first 3D simulations with the simple **B**-field, we observe the same vortices - only now as 3-dimensional cylindrical patterns.

3. In the more elaborate case, where the **B**-field is curved, the deflection of **u** is not only dependent on the magnitude of **B**, but also on the z-coordinate, i.e. distance from the equator. Particles closer to the poles should get deflected less significant.

The third observation should be visible by comparing the plots of the Navier-Stokes simulation with "flat" (i.e. parallel **B**-vectors in the whole domain) and curved **B**-field at the last time step:

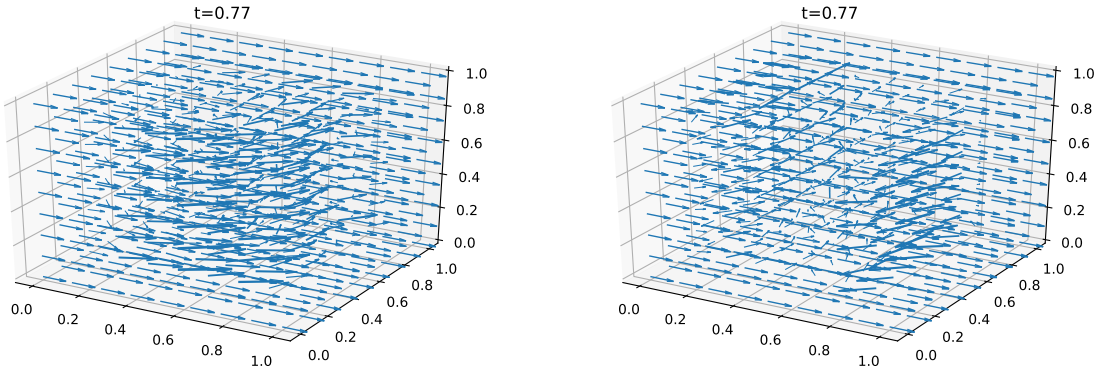

Figure 14: 3D Navier-Stokes, left: straight **B**-field, right: curved **B**-field

Unfortunately, there are some turbulent structures in the simulation with the curved **B**-field, that make it hard to see that.

4.2 Implication/Hypothesis

The fluid transport along the field lines could not be seen clearly, but the difference between a "flat" and a curved **B**-field were clearly visible.

Knowing this, the hypothesis is that the fluid transport along the field lines, i.e. in polar direction happens due to effects not directly linked to the structure of the **B**-field. These could be inertial and viscous forces, or just the planet as a physical obstacle itself that is influencing particle directions.

Still, the fact that the fluid deflection in areas with parallel **B**-field-lines (in the polar region) is smaller, could at least positively reinforce inertial particle transport as the **B**-field is less of an obstacle itself.

5 Conclusion

This project covered the 2D and 3D simulation of a simpler diffusive model as well as the more complex Navier-Stokes model of an electrically charged fluid (also called plasma) under the influence of a differently shaped **B**-fields.

Among others, it was assumed, that no central force field is present, the **B**-field was static, the planet was not included as a physical obstacle. Further, a term describing particle collision of the plasma with neutral (and therefore not influenced by the **B**-field) particles of the surrounding medium suggested by [CT16] was not included in the momentum equation. The full equation would read:

$$\dot{\mathbf{u}} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \mathbf{u} \times \mathbf{B} + \nabla \cdot \sigma + \frac{\mathbf{u} - \mathbf{u}_0}{\tau} = 0 \tag{13}$$

where τ is the mean free time between two collisions.

It is only natural to suggest extending the scope of this project in future work by modelling a more complex and more realistic environment.
Concerning the physics of the project one could

- include the planet as a physical obstacle
- model particle collisions with a neutral medium
- implement a more realistic **B**-field, that exactly reproduces the dipole equations
- include a central force, that models gravity as well as earth's **E**-field
- compute the pressure field
- simulate the full Maxwell model, with time dependent **B**- and **E**-fields.

The last point is obviously the most time consuming one.

If one wants to optimize the numerics and implementation, one could

- adjust the boundary conditions
- adjust fluid parameters
- figure out the reason for the turbulence in the case of the curved **B**-field, to do a more thorough comparison of the simulations
- parallelize the code in order to simulate longer time intervals or resolve turbulent flow patterns.
- compute a velocity projection on the local **B** vector, to quantify the velocity transport to the poles

6 Implementation

2D Diffusion case

```
from dolfin import *
from mshr import *
import dolfin.common.plotting as fenicsplot
import numpy as np
from matplotlib import pyplot as plt

# Time-steps
T = 15.0
num_steps = 300
dt = T / num_steps

# mesh and spaces
channel = Rectangle(Point(0, 0), Point(2, 1))
res = 10
mesh = generate_mesh(channel,res)
V = VectorFunctionSpace(mesh, 'Lagrange', 1)
u = TrialFunction(V)
v = TestFunction(V)

# BC
u_D = Constant((0.0001,0))
boundary_D = 'near(x[0], 0)'
bc = DirichletBC(V, u_D, boundary_D)

# equations
u_n = interpolate(u_D, V)
f = Constant((0,0.0001))
B = Expression(("0","0","exp(-10*(pow((x[0]-1.5),2)+pow((x[1]-0.5),2)))"),degree
                                        =1)
uxB = as_vector(( u[1]*B[2], -u[0]*B[2] ))
F = inner(u,v)*dx - inner(u_n + dt*f,v)*dx + dt*dot(uxB,v)*dx +0.01*dt*inner(
                                        grad(u), grad(v))*dx
a, L = lhs(F), rhs(F)

# time loop
u = Function(V)
t = 0
fignr=1
for n in range(num_steps):
    t += dt
    u_D.t = t

    solve(a == L, u, bc)

    if n%(num_steps/4)==0:
        plt.figure()
        plot(u)
        plt.title("t="+str(t)[:4])
        plt.tight_layout()
        fignr+=1

    # Update previous solution
    u_n.assign(u)
```

2D Navier-Stokes case

```
from __future__ import print_function
from fenics import *
from mshr import *
import numpy as np
import matplotlib.pyplot as plt

T = 0.1             # final time
num_steps = 200     # number of time steps
dt = T / num_steps # time step size
mu = 0.001          # dynamic viscosity
rho = 1             # density

# mesh and spaces
channel = Rectangle(Point(0, 0), Point(2, 1))
cylinder = Circle(Point(2, 0.5), 0.4)
domain = channel #- cylinder
mesh = generate_mesh(domain, 10)
V = VectorFunctionSpace(mesh, 'P', 2)
Q = FunctionSpace(mesh, 'P', 1)

# BC
inflow   = 'near(x[0], 0)'
outflow  = 'near(x[0], 2)'
walls    = 'near(x[1], 0) || near(x[1], 1)'
cylinder = 'on_boundary && x[0]>0.1 && x[0]<0.3 && x[1]>0.1 && x[1]<0.3'
bcu_walls = DirichletBC(V, Expression(("0.005","0"), degree=2), walls)
bcu_cylinder = DirichletBC(V, Constant((0, 0)), cylinder)
bcu_inflow = DirichletBC(V, Expression(("0.01","0"), degree=2), inflow)
bcp_outflow = DirichletBC(Q, Constant(0), outflow)
bcu = [bcu_inflow, bcu_walls]
bcp = [bcp_outflow]

# Define trial and test functions
u = TrialFunction(V)
v = TestFunction(V)
p = TrialFunction(Q)
q = TestFunction(Q)

# Define functions for solutions at previous and current time steps
u_n = Function(V)
u_  = Function(V)
p_n = Function(Q)
p_  = Function(Q)

# Define expressions used in variational forms
U  = 0.5*(u_n + u)
n  = FacetNormal(mesh)
f  = Constant((0, 0))
k  = Constant(dt)
mu = Constant(mu)
rho = Constant(rho)

# Define symmetric gradient
def epsilon(u):
    return sym(nabla_grad(u))

# Define stress tensor
def sigma(u, p):
    return 2*mu*epsilon(u) - p*Identity(len(u))

# lorentz term
```

```
B = Expression(("0","0","1*exp(-10*(pow((x[0]-1.5),2)+pow((x[1]-0.5),2)))"),
                                        degree=1)
uxB = as_vector(( u[1]*B[2], -u[0]*B[2] ))

# Define variational problem for step 1
F1 = rho*dot((u - u_n) / k, v)*dx \
   + rho*dot(dot(u_n, nabla_grad(u_n)), v)*dx \
   + inner(sigma(U, p_n), epsilon(v))*dx \
   + dot(p_n*n, v)*ds - dot(mu*nabla_grad(U)*n, v)*ds \
   - dot(f, v)*dx \
   + 100*dot(uxB,v)*dx
a1 = lhs(F1)
L1 = rhs(F1)

# Define variational problem for step 2
a2 = dot(nabla_grad(p), nabla_grad(q))*dx
L2 = dot(nabla_grad(p_n), nabla_grad(q))*dx - (1/k)*div(u_)*q*dx

# Define variational problem for step 3
a3 = dot(u, v)*dx
L3 = dot(u_, v)*dx - k*dot(nabla_grad(p_ - p_n), v)*dx

# Assemble matrices
A1 = assemble(a1)
A2 = assemble(a2)
A3 = assemble(a3)

# Apply boundary conditions to matrices
[bc.apply(A1) for bc in bcu]
[bc.apply(A2) for bc in bcp]

# Time-stepping
t = 0
fignr=1
for n in range(num_steps):

    # Update current time
    t += dt

    # Step 1: Tentative velocity step
    b1 = assemble(L1)
    [bc.apply(b1) for bc in bcu]
    solve(A1, u_.vector(), b1, 'bicgstab', 'hypre_amg')

    # Step 2: Pressure correction step
    b2 = assemble(L2)
    [bc.apply(b2) for bc in bcp]
    solve(A2, p_.vector(), b2, 'bicgstab', 'hypre_amg')

    # Step 3: Velocity correction step
    b3 = assemble(L3)
    solve(A3, u_.vector(), b3, 'cg', 'sor')

    # Plot solution
    if n%(num_steps/4) == 0:
        plt.figure()
        plot(u_, title='t='+str(t)[:4])
        plt.tight_layout()
        fignr+=1

    # Update previous solution
    u_n.assign(u_)
    p_n.assign(p_)
```

3D Diffusion case

```
from dolfin import *
from mshr import *
import dolfin.common.plotting as fenicsplot
import numpy as np
from matplotlib import pyplot as plt

# Time-steps
T = 1
num_steps = 80
dt = T / num_steps

# mesh and spaces
res = 8
mesh = UnitCubeMesh(res,res,res)
V = VectorFunctionSpace(mesh, 'Lagrange', 1)
u = TrialFunction(V)
v = TestFunction(V)

# BC
u_D = Constant((0.5,0,0))
# u_D = Constant((0,0))
def boundary(x, on_boundary):
    return on_boundary
bc = DirichletBC(V, u_D, boundary)

# Lorentz force
B = Expression(("0","0","10*exp(-10*(pow((x[0]-0.5),2)+pow((x[1]-0.9),2)))"),
                                        degree=1)
uxB = cross(u,B)

# rest of equations
u_n = interpolate(u_D, V)
f = Constant((0.1,0,0))
F = inner(u,v)*dx - inner(u_n + dt*f,v)*dx + 10*dt*dot(uxB,v)*dx + 0.1*dt*inner(
                                        grad(u), grad(v))*dx
a, L = lhs(F), rhs(F) #- g*v*ds

# time loop
u = Function(V)
t = 0
fignr=1
for n in range(num_steps):
    t += dt
    u_D.t = t

    solve(a == L, u, bc)

    if n%(num_steps/10)==0:
        print(n)

    if n%(num_steps/4)==0:
        plt.figure()
        plot(u)
        plt.title("t="+str(t)[:4])
        plt.tight_layout()
        fignr+=1

#     # Update previous solution
    u_n.assign(u)
```

3D Navier-Stokes case

Code source: https://fenicsproject.org/qa/5987/navier-stokes-demo-in-3d/

```
from dolfin import *
from mshr import *
import dolfin.common.plotting as fenicsplot
import numpy as np
from matplotlib import pyplot as plt

# Load mesh from file
res = 8
mesh = UnitCubeMesh(res,res,res)

class along_xy(SubDomain):

    def inside(self, x, on_boundary):
        return bool((near(x[0], 0) or near(x[1], 0)) and
        (not ((near(x[0], 1) and near(x[1], 0)) or
        (near(x[0], 0) and near(x[1], 0)))) and on_boundary)

    def map(self, x, y):
       if near(x[0], 1) and near(x[1], 1):
           y[0] = x[0] - 1
           y[1] = x[1] - 1
           y[2] = x[2]
       elif near(x[0], 1):
           y[0] = x[0] - 1
           y[1] = x[1]
           y[2] = x[2]
       elif near(x[1], 1):
           y[0] = x[0]
           y[1] = x[1] - 1
           y[2] = x[2]
       else:
           y[0] = -1000
           y[1] = -1000
           y[2] = -1000

axy = along_xy()

# Define function spaces (P2-P1)
V = VectorFunctionSpace(mesh, "CG", 1, dim=3, constrained_domain=axy)
Q = FunctionSpace(mesh, "CG", 1, constrained_domain=axy)

# Define trial and test functions
u = TrialFunction(V)
p = TrialFunction(Q)
v = TestFunction(V)
q = TestFunction(Q)

# Set parameter values
dt = 0.01
T = 1
nu = 0.01

# Define boundary conditions
noslip  = DirichletBC(V, (1, 0, 0),"on_boundary")
bcu = [noslip]

# Create functions
u0 = Function(V)
u1 = Function(V)
p1 = Function(Q)
```

```
# Define coefficients
k = Constant(dt)
f = Expression(('0.01', '0', '0'),degree=1)

B = Expression(("0","0","10*exp(-10*(pow((x[0]-0.5),2)+pow((x[1]-0.5),2)))"),
                                        degree=1)
B = Expression(("0","x[1]-0.5","10*exp(-1*(pow((x[0]-0.5),2)))-0.5"),degree=1)

# fancy B field
# B = Expression(("x[2]-0.5","0","20*exp(-1*(pow((x[0]-0.5),2)))-0.5"),degree=1)

uxB = cross(u0,B)

# Tentative velocity step
F1 = (1/k)*inner(u - u0, v)*dx + inner(grad(u0)*u0, v)*dx + \
     nu*inner(grad(u), grad(v))*dx - inner(f, v)*dx + inner(uxB,v)*dx
a1 = lhs(F1)
L1 = rhs(F1)

# Pressure update
a2 = inner(grad(p), grad(q))*dx
L2 = -(1/k)*div(u1)*q*dx

# Velocity update
a3 = inner(u, v)*dx
L3 = inner(u1, v)*dx - k*inner(grad(p1), v)*dx

# Assemble matrices
A1 = assemble(a1)
A2 = assemble(a2)
A3 = assemble(a3)

# Use amg preconditioner if available
prec = "amg" if has_krylov_solver_preconditioner("amg") else "default"

# Time-stepping
t = dt
fignr=1
n=0
num_steps = T/dt
while t < T + DOLFIN_EPS:

    # Update pressure boundary condition
    p_in.t = t

    # Compute tentative velocity step
    begin("Computing tentative velocity")
    b1 = assemble(L1)
    [bc.apply(A1, b1) for bc in bcu]
    solve(A1, u1.vector(), b1, "gmres", "default")
    end()

    # Pressure correction
    begin("Computing pressure correction")
    b2 = assemble(L2)
    #[bc.apply(A2, b2) for bc in bcp]
    solve(A2, p1.vector(), b2, "gmres", prec)
    end()

    # Velocity correction
    begin("Computing velocity correction")
    b3 = assemble(L3)
```

```
    [bc.apply(A3, b3) for bc in bcu]
    solve(A3, u1.vector(), b3, "gmres", "default")
    end()

    # Move to next time step
    u0.assign(u1)
    t += dt

    if n%(num_steps/4)==0:
        plt.figure()
        plot(u0)
        plt.title("t="+str(t)[:4])
        plt.tight_layout()
        fignr+=1
    n+=1
```