A Numerical Study of Partially Ionised Plasma Using a 2D Two-Fluid Magnetohydrodynamic Code

L Le Phuong

PhD

2020

A Numerical Study of Partially Ionised Plasma Using a 2D Two-Fluid Magnetohydrodynamic Code

Linh Le Phuong

A thesis submitted in partial fulfilment of the requirements of the University of Northumbria at Newcastle for the degree of Doctor of Philosophy

Research undertaken in the Faculty of Engineering and Environment in the Department of Mathematics, Physics, and Electrical Engineering

February 2020

Abstract

In a magnetised, partially ionised plasma, such as the lower solar atmosphere, the co-existence of both charged and neutral particles and their interaction lead to effects that do not occur in fully ionised plasma. Multi-fluid magnetohydrodynamic (MHD) models of a plasma account for the degree of ionisation and the resulting effects more accurately than common single-fluid MHD models. A 2D two-fluid MHD code has been developed (written in C++) to study partially ionised plasma, based on the Kurganov-Tadmor scheme. The advantage of this scheme is that it is Riemann-solver free, which makes computation faster, and it exhibits a small, time step independent numerical viscosity, which makes the code stable and accurate for much smaller time steps than usual schemes allow for. The explicit Euler and fourth-order Runge-Kutta scheme are implemented to integrate the solution in time. Furthermore, the implementation of a spatial domain decomposition scheme, which is based on Message Passing Interface (MPI) standard, allows for parallel computing. The code has been verified successfully and a number of tests were performed, such as the Sod-shock tube test and the Brio-Wu shock test. Moreover, the effect of ionisation and recombination on a magnetised, partially ionised plasma was investigated by studying a 1.5D slow-mode shock simulation and the 2D Orszag-Tang vortex simulation. The effect on the properties of both fluids, the ionised and the neutral fluid, are compared to simulations where collisions are the only coupling mechanism between the fluids. In the initialisation, the two simulations are fundamentally different; whereas the driver of the slow-mode shock formation is the discontinuity in the magnetic field, it is the velocity field that predominantly drives the vortex formation in the Orszag-Tang vortex simulation. However, for both, it was found that the movement of the ionised fluid decreases as well as the shock speeds when ionisation and recombination are included. The partial ionisation state of the plasma adds complexity to the numerical modelling of the solar atmosphere and so multi-fluid codes have long been ignored. Therefore, the code developed here provides a tool to unlock new investigations into the lower solar atmosphere and will allow new physics on the Sun to be explored.

Acknowledgement

"The only life worth living is one that you're really passionate about."

I would like to express my gratitude to Dr Sergiy Shelyag, without whom I would not have found my passion for numerical mathematics and computational fluid dynamics. Thank you very much for choosing me as your PhD student and always believing in me. Thank you for your support, encouragement, and your academic guidance, while letting me go my own way as well. Your own passion for your research and your curiosity were always a source of inspiration for me too. It was very hard for me to see you leaving the UK and our solar physics group, you are missed here.

On that note, I would like to thank Dr Gert Botha and Prof James McLaughlin for taking me under their wing when Sergiy had to leave. I appreciate the time and effort it takes to take over a PhD student half way through.
Gert, thank you so much for always taking your time for me, no matter what day or time or how busy you were yourself. Thank you for seeing progress where I couldn't see it and for your effort taking up partially ionised plasma research and always looking out for literature. Thank you also for always keeping up to date with the current stage of my thesis, your reliability and making me priority when it was important. Thank you also for digging out some interview questions and for taking the time to prepare me for my future - even though this meant that I will be leaving academic research.
James, you really are a great head of the group. I know it involves many meetings and a lot of administration, therefore, I am grateful for the time you were able to make for me and also for your advice as a second supervisor. I hope this album isn't quite like Little Mix's.
Without all of you, this thesis and my academic journey would not have been the same, and I do appreciate every step of it.

I would also like to thank Northumbria university for giving me the opportunity to work in such a great group and providing us PhD students with an office that has been my second home for the past 3 years. Especially, I want to acknowledge my fellow PhD students and office family Andre Chicrala, Malcolm Druett, Thomas Rees-Crockford and Ajay Tiwari.

I will always think back to the days in the office with you and to a great chapter in my life. And another very special office family member I want to thank is Mehrnoush. You know what you mean to me and I am very glad they let you move to the maths office - destiny. My dear friends, Annabel, Laure, Francesco, Sam, you also made those three years very special for me and something I will remember forever. I would also express my gratitude to Prof Dr Carsten Denker, my master thesis supervisor, who has always been very kind and encouraged me to do a PhD. During my first steps into research, I was guided by Dr Yvonne Unruh during my research internship, whom I would like to thank for the opportunity and her kindness.

Most of all, I would like to thank my family, because without their support and love, I would not be where I am. Everything I am and will be, I owe to you. Thank you so much for your endless and unconditional love, every day and minute of your life, which I can always feel and which gives me strength.
Mum, your kindness, respect and generosity towards everyone you meet, is something I always have and always will look up to. Thank you for your trust and believe in me and supporting me with anything I want or not want to do. All you always want is for me to be happy. Because of this, because I could always make my own decisions and always being loved, I am the happy and content person that I am today. Of course, thank you also for providing me with amazing food, throughout my whole life and even during my PhD. Dad, thanks for always talking to me on my way to work and back, for always being on my side, and your open mind, so I feel like I can always tell you anything. My sisters, Thao and Milly, I thank for their incredible wisdom and ability to think very differently and make me think in a way to make my own life easier, better and happier, too. Thank you also for being there for me always, at any time of the day and with any problem that I have, or just for a chat.

Last, I want to thank my fencing coach, Laszlo Jakab, for always being there for me, supporting and believing in me and for sharing his wisdom regarding fencing and life in general. Your care and excellent coaching have always been something that warmed my heart and lifted me up; and on that note I also want to thank the Durham University fencing club and Laszlo's fencing club for great and fun training, which really was a perfect balance to PhD life.

Gửi mẹ của con, không lời nào có thể diễn tả được con yêu mẹ nhiều như thế nào và tình yêu vô điều kiện của mẹ dành cho con có ý nghĩa vô cùng đối với con. Tình yêu của mẹ dành cho con mỗi ngày và theo nhiều cách khác nhau, là một nguồn năng lượng mạnh mẽ và mang lại sự yên tâm thoải mái cho con. Không có mẹ, con sẽ không biết mình đang ở đâu và con là ai nữa. Cảm ơn mẹ đã hỗ trợ con trong suốt quá trình học (và làm những việc trong nhà cho con mà đáng nhẽ ra con phải làm, haha), vì đã luôn ngủ trên giường của con khi con đến thăm, vì những món ăn tuyệt vời của mẹ và luôn trò chuyện với con trên điện thoại trên đường con đi làm về. Lòng tốt, sự tôn trọng và rộng lượng của mẹ luôn thể hiện với tất cả mọi người mà mẹ gặp là điều con luôn luôn hướng tới và cố gắng làm theo. Trái tim của mẹ, sự thông minh, luôn sẵn sàng lắng nghe và thay đổi là điều làm con kính trọng mẹ rất nhiều. (Thanks to my cousin Gia for this translation.)

Declaration

I declare that the work contained in this thesis has not been submitted for any other award and that it is all my own work. I also confirm that this work fully acknowledges opinions, ideas and contributions from the work of others.

Any ethical clearance for the research presented in this thesis has been approved. Approval has been sought and granted by the University Ethics Committee on 26.01.2018.
I declare that the word count of this thesis is 30953 words.

Name: Linh Le Phuong

Signature:

Date: 27.02.2020

Contents

1 Introduction **1**

1.1 Plasma Description 1

1.2 Partially Ionised Plasma (PIP) 3

1.3 Time and Length Scales 11

1.4 Motivation and Structure of the Thesis 15

2 Derivation of Governing Equations **17**

2.1 Derivation of the Fluid Approach and the Two-Fluid Model 17

2.1.1 From kinetic theory to fluid theory 19

2.1.2 Single-fluid MHD 22

2.1.3 Two-Fluid (ion-neutral) MHD 25

2.2 Derivation of the Hydrodynamic and MHD equations 26

2.2.1 Derivation of Hydrodynamic Equations 28

2.2.2 Derivation of Magnetohydrodynamic (MHD) Equations 35

2.3 Two-Fluid Source Terms 45

3 Numerical Methods – an Overview **55**

3.1 Partial Differential Equations (PDEs) 55

3.2 Finite Volume (FV) Method 59

3.3 The Kurganov-Tadmor Scheme 64

4 The Code and Its Verification **69**

4.1 Code Setup 69

4.2 Code Structure and Functions 73

4.3 Code Tests 78

4.3.1 Hydrodynamic Simulations 79
4.3.2 Magnetohydrodynamic (MHD) Simulations 85
4.3.3 Divergence Control 93

5 Two-Fluid MHD Code and Simulations 102
5.1 Two-Fluid Simulations 103
5.1.1 1.5D Slow-Mode Shock 105
5.1.2 2D Orszag-Tang Vortex 115

6 Conclusions 129
6.1 Conclusion With Regards to Two-Fluid Simulations 129
6.2 Summary 132

7 Future Avenues of Investigation 134
7.1 Rayleigh-Taylor Instability (RTI) 135
7.2 Kelvin-Helmholtz Instability (KHI) 137

A Running the Code 141

B The Code 144
Bibliography 199

Nomenclature

C	Courant number
P	Gas pressure
R_g	Gas constant
R_m	Magnetic Reynolds number
T	Temperature
V	Volume
Ω	Cyclotron frequency
Φ	Total magnetic flux
Φ_E	Total electric flux
α_c	Collision rate
ϵ	Total energy
ϵ_0	Electric permeativity of free space
η	Plasma diffusivity
η_{Hall}	Hall diffusivity coefficient
η_{Ohm}	Ohm diffusivity coefficient

η_{amb}	Ambipolar diffusivity coefficient
γ_{ion}	Ionisation rate
γ_{rec}	Recombination rate
λ_D	Debye length
$\mathbf{B}$	Magnetic field vector
$\mathbf{E}$	Electric field
$\mathbf{F}$	Force
$\mathbf{j}$	Current density
$\mathbf{p}$	Total pressure tensor
$\mathbf{v_A}$	Alfvén velocity
$\mathbf{v}$	Velocity vector
$\mathbf{v}_{pa}$	Particle velocity
$\mathbf{w}_\alpha$	Diffusion velocity of particle species α
μ_0	Magnetic permeability of free space
$\nu_{\alpha\beta}$	Collision frequency between particle α and particle β
ω_{pe}	Electron plasma frequency
ρ	Total mass density
ρ_α	Mass density of particles α
σ	Electrical conductivity
σ_c	Cross-section
τ	Magnetic damping time
τ_c	Characteristic time

τ_{xx}	Stress tensor component
ζ	Fluid fraction
c	Speed of light
c_s	Sound speed
e	Internal energy
k_B	Boltzmann constant
m_α	Mass of particle α
n_α	Number density of particle α
q	Heat
q_α	Charge of the particle α
t	Time
x, y, z	Spatial components of the Cartesian coordinate system

Mathematical Operators

$\mathbf{xy}$	= tensor product of two vectors = $\begin{pmatrix} x_1y_1 & x_1y_2 & x_1y_3 \\ x_2y_1 & x_2y_2 & x_2y_3 \\ x_3y_1 & x_3y_2 & x_3y_3 \end{pmatrix}$
$\mathbf{x}$	$= \mathbf{i}x_1 + \mathbf{j}x_2 + \mathbf{k}x_3$
$\mathbf{x} \cdot \mathbf{y}$	= dot product of two vectors = $x_1y_1 + x_2y_2 + x_3y_3$
$\mathbf{x} \times \mathbf{y}$	= cross product of two vectors = $\begin{pmatrix} x_2y_3 - x_3y_2 \\ x_3y_1 - x_1y_3 \\ x_1y_2 - x_2y_1 \end{pmatrix}$
∇	$= \mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}$

Chapter 1

Introduction

The visible matter in our universe is made of plasma by more than 90% (Goedbloed and Poedts, 2004, p. 3). Plasma is ionised gas and often contains enough free charges for electromagnetic forces to dominate the dynamics (Boyd and Sanderson, 2003).

1.1 Plasma Description

Theoretical models of plasma describe it either microscopically with (Goedbloed and Poedts, 2004, p. 34)

1. the theory of the motion of individual charged particles in given magnetic and electrical fields,

2. the kinetic theory of a collection of such particles (by means of particle distribution functions)

or macroscopically with

3. the fluid theory (magnetohydrodynamics (MHD)), describing plasmas in terms of averaged macroscopic functions

The single particle orbit theory is applied when the density is so low that interactions of particles can be neglected and no collective effects are to be studied. However, in a plasma, the interactions between particles are numerous and the kinetic theory makes use of statistical mechanics approaches to describe the collective behaviour of particles that

make up the plasma (Goedbloed and Poedts, 2004, p. 48). Here, physical information of the electrons and ions are expressed in terms of time-dependent distribution functions $f_\alpha(\mathbf{r}, \mathbf{v}, t)$ in a six-dimensional phase space and the total number of particles is assumed to be constant. The phase space is formed by three position coordinates ($\mathbf{r}$) and by three velocity coordinates ($\mathbf{v}$). Collisions of the particles change the distribution function. The evolution of this distribution function is described by the kinetic equation, also known as the Boltzmann equation. This combined with the Maxwell equations to determine the electric and magnetic field closes the microscopic equations for the plasma kinetic theory (Goedbloed and Poedts, 2004, p. 50).

The fluid theory averages out microscopic fluctuations, but considers microscopical aspects by requiring frequent enough collisions between ions and electrons to establish fluid behaviour. Additionally, besides the microscopic conditions of length and time scales of density and temperature, the macroscopic approach has to allow macroscopic length and time scale conditions of the magnetic field (Goedbloed and Poedts, 2004, p. 28). In this thesis, the fluid theory is applied, i.e. the macroscopic dynamics of magnetised plasma is studied. The fluid description can be applied when there is sufficient number of particles present so that local fluid properties such as pressure, density and velocity can be defined and, furthermore, when those particles, such as ions and electrons, interact or collide frequently enough (Spruit, 2017). The quantitative criterion can be determined with the transport theory, which provides the time scale, where the hydrodynamic time scale has to be much larger than the collisional relaxation times of the electrons, ions, and neutrals (Goedbloed and Poedts, 2004, p. 65). If this is true, a hydrodynamic description is valid. The relaxation time is the time related to the restoration of local equilibrium through collisions (Bittencourt, 2004, p. 135). A more detailed discussion of length and time scales can be found in Sec. 1.3.

In summary, "the dynamical configuration, size, duration, density and magnetic field strength have to be large enough to establish fluid behaviour and to average out the microscopic phenomena like collective plasma oscillations and cyclotron motions of the electrons and ions" (Goedbloed and Poedts, 2004, p. 28). Moreover, charge neutrality is assumed. The assumption of electrical conductivity of a plasma neutralises charge densities quickly and they only appear at the boundaries of the volume (Spruit, 2017).

Furthermore, the magnetic field $\mathbf{B}$ behaves differently depending on the conductivity of the fluid. Three important parameters in MHD are the magnetic Reynolds number

$R_m = \mu_0 \sigma u l$, which is related to the conductivity but actually the more important parameter to look at, the Alfvén velocity $v_A = B/\sqrt{\rho\mu_0}$, and the magnetic damping time $\tau = [\sigma B^2/\rho]$ with u as the velocity component, B the magnetic field component, μ_0 being the permeability of free space, σ the electrical conductivity, ρ the density of the conducting medium, and l the characteristic length scale (Davidson, 2001, p. 8). If the magnetic Reynolds number is high, the magnetic field lines are frozen into the conducting fluid and act like elastic bands. When the medium is disturbed, this would result in almost elastic oscillations (where the magnetic field provides the restoring force), which leads to the formation of Alfvén waves (Davidson, 2001, p. 8). If the magnetic Reynolds number is small, the induced field is negligible compared to the imposed magnetic field, and the velocity $\mathbf{v}$ has little influence on the magnetic field, which in this case is more dissipative, and the time scales considered are not related to the Alfvén velocity, but the damping time τ (Davidson, 2001, p. 9). Therefore, the Reynolds number gives us the strength of coupling between the flow and the magnetic field (Davidson, 2001, p. 71).

1.2 Partially Ionised Plasma (PIP)

Plasma can be partially or fully ionised; in a fully ionised plasma the particles are all charged, whereas a partially ionised plasma includes neutrals to a high portion.

In a magnetised environment like the solar atmosphere, the particles interact with each other, but have different forces acting on them, e.g. charged particles feel the magnetic field, whereas neutrals do not. They are also connected through collisions between them, and in the lower solar atmosphere the plasma is dense and collisions play a significant role. Hence, the existence of both charged particles and neutrals embedded in a magnetic field and colliding with each other results in effects or properties that fully ionised plasmas do not show: e.g. ambipolar diffusion, Hall effect, heating due to ion-neutral friction, charge exchange, or ionisation energy (Ballester *et al.*, 2017). These have direct effects on the plasma dynamics and the energy exchange (González-Morales *et al.*, 2018).

For example, the plasma can drift with respect to the neutrals and this plasma-neutral drift is called ambipolar diffusion (Zweibel, 2015). More precisely, ambipolar diffusion in astrophysics refers to the diffusion of the magnetic field through the neutral fluid, because, although neutral and charged components are decoupled (and move with different velocities), the ions and the neutrals collide and now the magnetic flux, which is frozen

into and moving with the charges, diffuses with respect to the neutrals (Khomenko and Collados, 2012) and there is a redistribution of the magnetic flux (Zweibel, 2015). In the solar chromosphere for instance, the collisions between ions and neutrals couple the otherwise not affected neutrals to the magnetic field but also allow some of the ions to move across the magnetic field, which then can diffuse and the magnetic energy dissipates into thermal energy or heat (Martínez Sykora *et al.*, 2015). Ambipolar diffusion helps converting magnetic energy to thermal energy faster and, because of its larger magntitude, stronger than Ohmic diffusion does (González-Morales *et al.*, 2018). Ambipolar diffusion - a term used in Astrophysics, similarly also referred to as Pedersen resistivity, whereas diffusivity and resistivity are used interchangeably - is caused by the collision of neutrals with ions, act perpendicularly to the magnetic field and can lead to thin current structures in weakly ionised plasma (Leake *et al.*, 2012). The extent to which the magnetic field is coupled to the neutrals and, therefore, the magnitude of ambipolar diffusion depends on the collision frequency – the more collisions, the higher the coupling. This can also be seen in Eqn. (1.4). It can also be inferred from that equation that ambipolar diffusion effects are proportional to the neutral-ion density ratio of the fluid: the bigger the ionisation fraction, the smaller the ambipolar diffusion effects.

The Hall effect in a fully ionised plasma is similar to the one appearing in a partially ionised plasma. In both cases, it results from the decoupling of the ions from the magnetic field, whereas the electrons are still coupled to it. Unlike in a fully ionised plasma, where this decoupling happens because of a difference of inertia of ions and electrons at high frequencies, in a partially ionised plasma, the decoupling is due to the collisions of the ions with the neutrals (González-Morales *et al.*, 2018). The Hall effect does no work on the plasma and, therefore, produces no direct heating (Raboonik and Cally, 2019). Although being dispersive and, therefore, non-dissipative and not contributing to the heating of the plasma, it leads to redistribution of the magnetic energy in the system (González-Morales *et al.*, 2018).

In solar physics, in the quest of solving the coronal heating problem, Alfvén waves play a crucial role. Alfvén waves are assumed to propagate through the chromosphere and corona where those layers are heated by the dissipation of the wave energy (Vranjes *et al.*, 2008). In their paper, Vranjes *et al.* (2008) study the energy flux of Alfven waves under solar-photosphere conditions, which means under weakly ionised plasma conditions and found that this energy flux is orders of magnitudes smaller than under ideal MHD con-

ditions, which would make the generation of waves less efficient there and, therefore, they questioned the photospheric Alfvén waves as a source for chromospheric and coronal heating. Tsap, Stepanov, and Kopylova (2011), however, found that the amplitude of Alfvén waves does not depend on the ionisation fraction and, therefore, is the same in both cases, the ideal MHD case and the weakly ionised plasma case. In 2013, Soler *et al.* (2013) found that the energy flux depends on initial velocities of both (ion and neutral) fluids and pointed out the different initial conditions Vranjes *et al.* (2008) and Tsap, Stepanov, and Kopylova (2011) started off with and which is why they came to different conclusions (Ballester *et al.*, 2017).

However, because the existence of those waves and their generation in the weakly ionised, ion-neutral collisions dominated photosphere has not been observationally confirmed, there thought to be mode conversion mechanisms based on other MHD waves in the lower solar atmosphere that are related to the production of Alfvén waves. Raboonik and Cally (2019) show how a partially or weakly ionised plasma provides a mechanism, namely the Hall effect, to convert fast waves into Alfvén waves. The Hall effect, as well as the ambipolar diffusion, depends on the magnetic field strength, unlike the Ohmic diffusion, which is diffusion due to collisions. These abovementioned effects are related to the difference in velocity of the neutral and ionised fluid, or the drift velocity. A drift in the velocity occurs because of the different forces that act on the particles. But, as the particles collide and exchange momentum, this difference in the velocity reduces over time and they both get the same velocity and flow together (Hillier, 2019). But before that, this drift velocity can also lead to frictional heating and the coupling process can be important for the energy dissipation (Khomenko and Collados, 2012).

The co-existence of ions and neutrals also leads to three shocks that are characteristic for a partially ionised plasma: the continuous shock, or C-type shock, the J-type shock, or the C*-type shock. They are a result of the coupling of ions and neutrals too and their classification is related to the neutral fluid's flow variables. If a shock moves with a speed lower than the Alfvén speed, the ions can build a so called magnetic precursor ahead of the shock, where the coupling to the neutral fluid makes them accelerate with the ionised fluid before the shock arrives. Therefore, post-shock, the neutral fluid's flow variables show a continuous (C-type) variation. If the neutral fluid is subsonic before the shock arrives, the shock will either contain a jump in the neutral flow variables (J-type) or it will have two continuous sonic points (C*-type), as post-shock the neutral fluid will be

super-sonic again (Toth, 1994).

In simulations, the co-existence of neutrals and ions and its effects have been implemented in different ways. In solar physics, there are single-fluid MHD codes that account for partial ionisation by adding extra terms to the equations which approximate the existence of neutrals, for example, described in Leake and Arber (2006). Arber, Haynes, and Leake (2007), for instance, do so in their two-dimensional (2D) code by taking averages of the densities and temperatures of the ions and neutrals and assume ionisation balance and the centre of mass velocity used in Ohm's law is an average over ions and neutrals, too. Vögler *et al.* (2005) created a 3D MHD code to study the solar convection zone and photosphere. They consider partial ionisation in their single-fluid code by adjusting the equation of state according to the ionisation fraction, as partial ionisation influences the adiabatic gradient and, therefore, the specific heat and the radiative damping of temperature fluctuations. Also Martínez-Sykora *et al.* (2017) consider a partially ionised plasma in their model to simulate spicules, which occur when the magnetic tension is amplified through the interaction of ions and neutrals and the resulting ambipolar diffusion. For their single-fluid simulations obtained with the BiFrost code (Gudiksen *et al.*, 2011), they extend the induction equation by an ambipolar diffusion and Hall term. Alvarez Laguna *et al.* (2018) give an informative overview of the differences and advantages of a two-fluid model over a single-fluid MHD model and the regime where two-fluid models are necessary, before introducing their fully-implicit finite volume code to model ideal two-fluid plasmas. Therefore, in the single fluid description of partially-ionized plasma, taking into account partial ionisation can be achieved by adding additional terms to Ohm's law that account for the effects of ion-neutral interaction (Maneva *et al.*, 2017). The generalised Ohm's law then has extra terms such as the Hall term or the ambipolar diffusion term (Cap, 1994, p.126). The induction equation would then look as follows:

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times \left[\mathbf{v} \times \mathbf{B} - \eta_{ohm} \mathbf{j} - \frac{\eta_{hall}}{|B|} \mathbf{j} \times \mathbf{B} + \frac{\eta_{amb}}{B^2} (\mathbf{j} \times \mathbf{B}) \times B \right], \tag{1.1}$$

where **B**, **j**, and **v** are the magnetic field, current density and velocity field, respectively. The terms η_{ohm}, η_{Hall} and η_{amb} are related to the ohmic diffusion, the Hall diffusion term, and the ambipolar diffusion term, respectively, and are defined as:

$$\eta_{ohm} = \frac{m_e(\nu_{ei} + \nu_{en})}{q_e^2 n_e}, \tag{1.2}$$

$$\eta_{hall} = \frac{|B|}{q_e n_e}, \tag{1.3}$$

$$\eta_{amb} = \frac{(|B|\rho_n/\rho)^2}{\rho_i \nu_{in}} = \frac{(|B|\rho_n/\rho)^2}{\rho_n \nu_{ni}}. \tag{1.4}$$

As can be seen, these terms depend heavily on the interaction of the species (ion-neutral and neutral-ion collision rates ν_{in} and ν_{ni}, the total density ρ) and on each of the species' individual characteristics like ion and neutral densities ρ_i and ρ_n, electron number density n_e and electron charge q_e (Martínez Sykora *et al.*, 2015).

In a partially ionised plasma, the Hall term leads to an electric field which is parallel to $\mathbf{j} \times \mathbf{B}$. Ambipolar diffusion occurs in a partially ionised plasma because neutrals are not affected by the Lorentz force and, therefore, decouple from the magnetic field. It also dissipates electric currents perpendicular to the magnetic field and, consequently, magnetic energy is converted to thermal energy. Ambipolar diffusion is dependent on the thermal structure of the plasma and affects instabilities like the Rayleigh-Taylor instability by either inhibiting the instability or increasing small-scale structures (velocities), depending on its spatial distribution (Martínez Sykora *et al.*, 2015).

Furthermore, those ion-neutral collisions can lead to the dissipation of Alfvénic waves and their damping (de Pontieu and Haerendel, 1998; Soler *et al.*, 2017) and, according to Soler, Oliver, and Ballester (2009), who studied the effect of neutrals on wave propagation in a filament thread, actually are the most efficient damping mechanism for short wavelengths. Hence, damping of waves results from collisions between the ionised and neutral fluid and their momentum exchange and the inertia of the neutrals also leads to a decrease of their phase speed (Martínez-Gómez, Soler, and Terradas, 2017). Two energy dissipation mechanisms, which convert the energy of damped MHD waves into thermal energy, are collisional dissipation (resistivity) and viscosity and the presence of neutrals enhances the efficiency of both of these mechanisms (Khodachenko *et al.*, 2004). Collisional dissipation is related to the collisional friction forces that occur due to the relative motion of the fluids and the physical nature of viscosity which makes it a dissipation mechanism is related to the kinetic energy and the momentum transfer between the fluids.

The single-fluid MHD models, however, have limitations and are less accurate. For example, the short spatial and temporal scales related to the collisions of the particles are not sufficiently well treated when using the generalised Ohm's law (Maneva *et al.*, 2017). Therefore, a generalised Ohm's law, where ambipolar diffusion terms are added, can only be applied where the dynamic frequency is much smaller than the collision frequencies (Hillier, 2019). Zaqarashvili, Khodachenko, and Rucker (2011) derive MHD waves in two-fluid partially ionised plasmas and find that, for slow waves, the single-fluid and the two-fluid descriptions give similar results, whereas for waves, whose frequency becomes comparable with or are higher than the ion-neutral collision frequency, the dynamics of MHD waves changes significantly in a two-fluid approach. For example, they found that Alfvén and fast magneto-acoustic waves are damped most at a certain ionisation fraction.

In multi-fluid codes, the aforementioned effects do not have to be accounted for by adding extra terms to the induction equation, as they are naturally implemented and occur just because of the mere co-existence and interaction of the fluids. Multi-fluid theory, for example, accounts for the short length and time scales that are related to the collisions between the fluids that cannot be captured with the generalised Ohm's law added to the single fluid approach (Maneva *et al.*, 2017). An illustrated overview of approaches to mathematically describe a plasma, depending on the regime, can be found in Alvarez-Laguna *et al.* (2018).

In solar physics, two-fluid codes are still not an established means to simulate the solar atmosphere and we are only at the beginning of investigating the physics of the Sun when partial ionisation in a two- or multi-fluid setting is accounted for. Zaqarashvili, Khodachenko, and Rucker (2011) show that there is a significant difference in the damping of fast waves, yet, it needs more investigation in terms of the damping of high-frequency waves with a realistic height profile of the ionisation degree for the solar chromosphere. Leake *et al.* (2012) presented the first multi-fluid simulation of reconnection in the solar atmosphere and found that the advantage of a two-fluid approach is that it allows high non-LTE ion densities, which naturally form in reconnection regions (as their simulations show). Their resulting fast recombination influences the reconnection process.

Hillier, Takasao, and Nakamura (2016) simulate a 1.5D slow mode shock with two fluids, where the two fluids are coupled via a collision term. In their initial conditions everything is in equilibrium, but the magnetic field. They found that the problem set for partially-ionised plasma is dynamically different from ideal MHD and the two-fluid sim-

ulation develops more complex and numerous shocks. Due to changes in the ionisation fraction and density with height propagating slow-mode shocks are constantly evolving on their way and ideal MHD assumptions are not valid any longer. Maneva *et al.* (2017) studied the wave propagation through the chromosphere with their 3D CoolFluid code and find that it significantly differs from single-fluid results. Popescu Braileanu *et al.* (2019) further developed the Mancha3D code (Felipe, Khomenko, and Collados, 2010) and turned it into a two-fluid code to simulate waves and shocks in the solar chromosphere. They show the importance of multi-fluid modelling by revealing the difference of wave propagation when the coupling is strong and when the coupling is weak and, moreover, what consequences the strength of collision has on the wave frequencies and damping. It is clear that more investigation is crucial, as there is a significant difference in the physical consequences and effects when a more accurate approach of the partially or weakly ionised plasma is applied.

Up to here, it has been established that in a partially ionised plasma, the co-existence and interaction of charged particles and neutrals leads to additional effects that change the structure of and the dynamics in a magnetised plasma. Now, if, additionally, the proportion of the charged particles and neutrals changes, the magnitude of these effects and the dynamics in the plasma change again. The proportion can change due to chemical reactions such as ionisation and recombination. In the solar chromosphere, for instance, the complex structure is due to the combination of it being partially ionised, a decreasing density and pressure with height, which leads to a change from a strongly collisional behaviour to a weakly collisional behaviour and a change from being dominated by the gas pressure to being dominated by the magnetic pressure. On top of that, the variation of the ionisation fraction adds to this complexity.

In our model, which will be described in more detail in Chapter 4, we do not consider the solar atmospheric chemical composition, but a pure hydrogen plasma made up of two fluids, one consisting of singly charged ions of charge $+e$ and one fluid consisting of neutral atoms, where charge neutrality ($n_e = n_i = n$) and a negligible electron mass is assumed. The temperature of the electrons is equal to the temperature of the ions, and the ionised fluid moves with the velocity of the ions. Elastic collisions cause momentum exchange of the fluids. Momentum can also be gained and lost due to ionisation and recombination, mathematically manifested in the source terms, discussed in Sec. 2.3. Neutrals and ions can be heated due to ionisation and recombination (inelastic collisions)

(Popescu Braileanu *et al.*, 2019). In this model, simple ionisation and recombination take place: the ions can be recombined and neutrals can be ionised.

Essentially, for recombination, an electron is merged with an ion and together form a neutral atom, whose mass, momentum, and energy is then considered in the neutral fluid in our model. In reality, ionisation and recombination are much more complex and related to the multi-level structure of atoms (Biberman, Vorob'ev, and Yakubov, 1969). Recombination can be spontaneous or stimulated or a three-body collisional recombination, but to account for this, a very elaborated model and code would be needed, which is beyond the scope of this project. But, as an example, to produce and atom in the ground state by recombination, the electron has to pass through a set of excitation states and recombination is, therefore and amongst others, related to the distribution of atoms over the energy levels (Biberman, Vorob'ev, and Yakubov, 1969). In this work, ionisation is also considered in a simple way, namely as the loss of an electron of a neutral hydrogen atom and results in a positively charged ion. It can result from thermal processes or non-thermal processes like collisions with an electron or atoms or electromagnetic radiation, i.e. photons.

Ionisation and recombination affect the dynamics and properties of a plasma in the way that it changes the ionisation fraction and affects the magnitude of effects like ambipolar diffusion, Hall effect or Ohmic dissipation, which were discussed above. Without ionisation and recombination terms included in the simulations, the ionisation fraction stays the same, or in other words, the amount of each neutrals and ions does not change and, therefore, in these simulations the effects resulting from the coupling of ions and neutrals, are not affected by the ion-neutral proportion, when in fact they are. Ionisation means a stronger coupling of the fluids due to more interactions, which can also be seen in the source terms, e.g. Eqn. (2.141), where extra terms are added when including ionisation and recombination. Through ionisation the ionised fluid is heated and through recombination, the neutral fluid is heated (Popescu Braileanu *et al.*, 2019).

Mathers and Cramer (1978) investigate the effect of ionisation and recombination on resistivity. Usually, in a partially ionised plasma, the resistivity is enhanced due to ion slip for currents perpendicular to the magnetic field. However, with ionisation and recombination, this enhancement is reduced. This is thought to be due to the decrease of speed of the ionised fluid relative to the neutral fluid caused by ionisation, as the ion slip is reduced and the dissipation due to plasma-neutral friction decreases. The aforemen-

tioned complex structure of the chromosphere affects the wave propagation through it and causes damping of Alfvén waves (Soler *et al.*, 2017). This damping of waves depends on the ionisation fraction as well and the dispersion of the Alfvén waves is due to the different time scales the Alfvén speed has, depending on the strength of coupling of the fluids (Kumar and Roberts, 2003) . Maneva *et al.* (2017) investigate the wave propagation when ionisation and recombination is included in their code and find, for example, that the results significantly differ from single fluid results in terms of wave propagation through the solar chromosphere. Furthermore, (Pandey and Wardle, 2008) show that both ambipolar diffusion and the Hall diffusion depend on the ionisation fraction and that Hall diffusion is especially important when the ionisation fraction is high. They show that, even though the dynamics of the Hall effect is similar in a highly ionised and weakly ionised plasma, the frequency ranges and spatial scales on which they occur can be different. This means that the ionisation fraction plays an important role in the Hall diffusion too.

Later in this thesis, simulations are studied, where ionisation and recombination rates are activated and compared to simulations without ionisation and recombination, i.e. where collisions are the only interaction between the fluids.

1.3 Time and Length Scales

An overview of the length and time scales that play a role in plasma physics on various scales, gives us an idea about when it is important to apply which plasma theory to the problem under study. Furthermore, it will be briefly discussed what time and length scales have to be considered when dealing with the single-fluid MHD approach or the two-fluid MHD approach.

If we consider one particle only, the motion of a charged particle in a magnetised plasma is influenced by the electromagnetic field. The circular motion of electrons and ions is then given by the gyro or cyclotron radius R and the gyro or cyclotron frequency Ω (Goedbloed and Poedts, 2004, p.28):

$$\Omega \equiv \frac{|q|B}{m} \qquad \text{and} \qquad R \equiv \frac{v_\perp}{\Omega}. \tag{1.5}$$

Both, the cyclotron radius R and cyclotron frequency Ω depend on the particle's charge

q and mass m and it follows that for electrons the radius is smaller and the cyclotron frequency higher compared to the cyclotron radius and frequency of the ions. $v_\perp$ is the perpendicular velocity. The length scales for which the fluid theory can be applied have to be larger than the ion cyclotron radius (Goedbloed and Poedts, 2004, p.21). As for the length scales, there is also a distinction between scales parallel or perpendicular to the magnetic field. Perpendicular to the magnetic field, the length scale is related to and has to be larger than the scales related to the cyclotron motion. However, parallel to the magnetic field, the length or spatial scales have to be much bigger than scales related to the collision frequency and the thermal and plasma velocities (i.e. the thermal free path scale) (Khomenko *et al.*, 2014). Furthermore, the conditions for collective plasma behaviour are also that the length scales have to be bigger than the Debye length for a quasi-neutral plasma. The Debye length λ_D is given by

$$\lambda_D \equiv \sqrt{\frac{\epsilon_0 k_B T}{q_e^2 n}}, \tag{1.6}$$

where ϵ_0 is the electric permeativity of free space, $k_B T$ the thermal energy with k_B as the Boltzmann constant and T as the temperature, q_e is the elementary charge and n the particle density. The sphere with the radius of the Debye length, or also called the Debye shielding length, is the space beyond which the plasma remains effectively neutral (Boyd and Sanderson, 2003, p.8). In the Corona, for instance, the Debye length would be $\lambda_D = 0.07$m (Goedbloed and Poedts, 2004, p.21).

As for the time scales, those have to be larger than the microscopic particle motion time scales such as the above mentioned cyclotron frequency Ω, but also the electron plasma frequency ω_{pe} and the collision frequency between the charged particles ν_{ch}. The electron plasma frequency is related to the oscillations that occur when electrons in a plasma are pulled back by the strong electrostatic forces or the Coulomb forces, when there is a charge imbalance. This mechanism secures charge neutrality. The electron plasma frequency ω_{pe} is given by:

$$\omega_{pe} = \frac{\sqrt{k_B T / m_e}}{\lambda_D} = \left(\frac{n_e q_e^2}{m_e \epsilon_0}\right)^{1/2}, \tag{1.7}$$

and reduces to $\omega_{pe} = 56.4 n_e^{1/2} s^{-1}$ (Boyd and Sanderson, 2003, p. 8). The ion and electron collisions have to be frequent, i.e. the collision time scales short, so that the particles

can be considered as one fluid. Collision times reflect the times for significant particle deflection, i.e. momentum change and exchange (Boyd and Sanderson, 2003, p.62). This also means that the time scales occurring in the problem under study have to be sufficiently larger than the time scales of collisions between the charged particles. However, if there are neutrals as well, these charged-particle-collision time scales have to be sufficiently short compared to the collision times with the neutrals, so that the long-range Coulomb interaction between the charged particles dominate over the short-range collisions with the neutrals (Goedbloed and Poedts, 2004, p.20). Therefore, the time scale τ, has to be much smaller than the time between ion-neutral collisions τ_n, which is expressed through:

$$\tau \ll \tau_n \equiv \frac{1}{n_n \sigma v_{th}} \approx \frac{\lambda_{mfp}}{v_{th}}, \tag{1.8}$$

where λ_{mfp} is the mean free path and v_{th} is the thermal velocity of the particles. The mean free path is, therefore, $\lambda_{mfp} = (n_n \sigma_c)^- 1$, where n_n is the number density of neutrals and σ_c the cross-section. $\sigma_c = \pi a^2$, where a is chosen to be the radius of a neutral hydrogen (H) atom and $v_{th} = (k_B T/m_p)^{1/2}$. Therefore, the mean free path essentially gives us the collision frequency, which then allows the determination of the according time scales and the comparison to other time scales. Moreover, in terms of length scales, the mean free path has to be much smaller than the hydrodynamic length scales. (Boyd and Sanderson, 2003, p.63) For a plasma, the neutral number density can be converted into an ion number density with the Saha-equation. In the solar Corona, where the temperature and ion number density is high ($T = 10^6$K and $n_i = 10^{12}$ m^{-3}, $n_n = 4 \times 10^{-7}$ m^{-3}), the time scale would be $\tau \ll \tau_n \approx 2 \times 10^{20}$s.

In the lower solar atmosphere, there are a high number of neutrals and the temperature is low. To determine the applicability of a certain model for a problem (here, either single-fluid MHD or two-fluid MHD), the microscopic length and time scales mentioned above have to be compared to collisions between the charged particles only and also collisions between the charged particles with the neutrals. Usually, single fluid MHD (where required with some modifications) is a good approximation for the processes on the Sun and the solar photosphere, as collisions between ions and neutrals are frequent enough and the plasma is strongly coupled. The application of single-fluid MHD to simulate solar processes such as convection, magneto-convection, formation of magnetic structures or wave propagation are good examples for the success of single-fluid MHD application

for the Sun (Khomenko *et al.*, 2014). However, when the magnetic field is large, for example in sunspot regions, the cyclotron frequency can become very large too, and can overcome collisional frequencies and ideal MHD assumptions are not any longer valid (Khomenko *et al.*, 2014).

In the chromosphere, the collisional coupling is not as strong as in the photosphere, and, therefore, single-fluid MHD assumptions break as well (Khomenko *et al.*, 2014). This means, if the collisional coupling between charged particles and neutrals is not sufficient so that they can be considered as one fluid or if the time scale of the problem under study is similar or smaller than the collision time scale between the two fluids, a better approach is the two-fluid model.

Therefore, two-fluid modelling is needed where there is a considerate amount of neutral particles that do not show a strong collisional coupling to the charged particles, with collision time scales longer than typical MHD time scales. However, the two-fluid model involves the single-fluid MHD model (which represents the ionised fluid), which expresses all the occurring charged particles as one fluid. Therefore, assumptions and neglections are made, for instance, stationary currents are assumed and charge separation is neglected. Two-fluid approximations can only be applied, where the coupling between the charged particles is stronger than the coupling between the neutral particles (Khomenko *et al.*, 2014). This is further discussed in Chapter 2. In the simulations performed in this thesis, the time and length scales are greater than the ones related to the plasma frequency and the Debye length, respectively. They, therefore, fall into the applicability range of the fluid approach.

For these simulations, the estimations of the time scales are non-dimensionalised (the non-dimensionalisation is further discussed in Chapter 5). The characteristic time scale, τ_c, is calculated with $\tau_c = L/c$, where the characteristic length $L = 1$ and where c is a characteristic speed. For the neutral fluid, the characteristic speed is the maximum sound speed c_S. For a plasma, or the ionised fluid, the characteristic speed is the maximum of c_S or the Alfvén speed v_A. For the hydrodynamic Sod shock tube simulation (discussed in Sec. 4.3.1) $\tau_c = 0.9$. For the ionised fluid in the Brio-Wu shock simulation (discussed in Sec.4.3.2) $\tau_c = 0.3$. In the 1.5D slow-mode shock simulation (discussed in Sec. 5.1.1), a neutral and ionised fluid are present and τ_c for the neutral fluid is 0.5 and τ_c for the ionised fluid is 0.3. The two fluids interact through collisions with a collision rate α_c, given by Eqn.(2.142), which leads to the characteristic collision time scale of $1/\alpha_c$, which is 0.4 for

the 1.5D slow-mode shock. For the two-fluid Orszag-Tang vortex simulation (discussed in Sec. 5.1.2), $\tau_c = 0.6$ for both fluids and the characteristic collision time scale is 1.

1.4 Motivation and Structure of the Thesis

It is a challenging and costly task to observe plasma in space, because the object or event is very far away or can be hidden. Another means of studying astrophysical plasma and help understand the physics in space are simulations.

A two-fluid magnetohydrodynamic (MHD) code has been developed here to account for the partial ionisation state of a plasma more accurately than common single-fluid simulations can. This code is meant to provide the basis of a tool and simulations achieved and studied with this code provides an insight into the two-fluid plasma behaviour and the effect of ionisation and recombination on this magnetised plasma.

The intention is to study the physics of the Sun under the multi-fluid model in the future, applying this code. The Sun is the most important star to us, as it is our main energy source and its activity and behaviour can have a severe effect on our lives and technology on Earth. Moreover, being the closest star to us makes it a great prototype to understand and study other stars and the universe. Yet, there are still open questions to be answered in the study of the Sun. One of them is the coronal heating problem and refers to the temperature of the outermost layer of the Sun, which is by a few orders of magnitude larger than the temperature of the lower layers of the solar atmosphere, the photosphere and chromosphere. The energy that is generated in the interior of the Sun has to go through the photosphere, chromosphere, and transition region. Those layers and regions influence the mass, energy and momentum transfer from the underlying photosphere and convection zone into the corona. Therefore, understanding the energy transport and conversion in those layers is the key to solve the coronal heating problem. The complexity of the chemical structure of the chromosphere, however, makes studying it thoroughly a challenging task, and it is well known that this complexity is related to its partial ionisation state (Maneva *et al.*, 2017). To model and understand the physics in those layers, additional effects resulting from the partial ionisation or the co-existence and interaction of neutrals and charged particles, have to be considered in the modelling and analysis of the solar atmosphere. These simulations allow us to understand the physics that leads to what we can observe and enables us to make predictions. Therefore, a great

advantage of simulations is they enable us to see and analyse what otherwise is hidden in observations and need to be inferred (Arregui, 2015). The question of how and to what extent do partially ionised plasma effects influence the physics and dynamics on the Sun, is still to be answered.

In Chapter 2, the equations that lay the basis for the MHD model of plasma are derived. An overview of the derivation of the fluid approach from the kinetic approach is given and it is shown how the fluid equations are obtained from kinetic equations. It is, furthermore, explained which physical laws lead to the Euler and Navier-Stokes equations and how Maxwell's equations add to the system of hydrodynamic equations to form the MHD equations. In Chapter 3, an overview of the mathematics and numerical methods is given, which are the underlying theories for the code developed here. In Chapter 4, the code is presented. First, the overall code set-up is described, before the code structure is revealed, where the main functions in the code are explained. As simulations are a complex interplay of the physical setting, the underlying mathematics, and appropriate numerical schemes, we first want to ensure that the code does what it is supposed to do, with high accuracy and stability. Therefore, the code is verified with simulations and tests in the hydrodynamic and MHD regime. In Chapter 5, the two-fluid MHD source terms are introduced, and the results of our 1.5D and 2D two-fluid MHD simulations and the effects of ionisation and recombination on a partially ionised plasma are presented. In Chapter 6, a conclusion regarding the two-fluid results is given, before a summary of the thesis closes this chapter. The thesis is completed by giving an insight into the future avenues in Chapter 7.

Chapter 2

Derivation of Governing Equations

First, the mathematical derivation of the fluid theory from kinetic theory is shown. Following this, a more intuitive, physical derivation of the macroscopic hydrodynamic and the MHD equations is presented.

2.1 Derivation of the Fluid Approach and the Two-Fluid Model

Amongst the theoretical descriptions of plasma, the kinetic theory is the most comprehensive model. To simulate a plasma of a certain density with the kinetic theory, collision terms are introduced that change the distribution function and reveal the evolution of the plasma. The derivation of the collision term is a very complex matter, as plasma collisions are many-body interactions, affected also by the long-ranged Coulomb force and forces of the electromagnetic field.
In the hydrodynamic case, the frequent collisions between the neutral particles is the reason why these particles move as one fluid (Chen, 1984, p.53). In a high-temperature plasma, however, collisions can be rare and, as a result, deviations from thermal equilibrium can last considerably longer too. Nevertheless, most of the time, plasma can be described with the fluid approach very well (Chen, 1984, p.225).

As it is not possible to track every particle in a fluid or determine their equation of motion, statistical methods have to be employed to describe a collection of particles. The

starting point for a statistical description of a many-body system, is given by the Liouville function (Cap, 1994, p. 62):

$$F = \{x_\alpha, y_\alpha, z_\alpha, x_\beta, y_\beta, z_\beta, ..., x_N, y_N, z_N, v_{\alpha x}, v_{\alpha y}, v_{\alpha z}, v_{Nx}, v_{Ny}, v_{Nz}, t\}, \tag{2.1}$$

which gives for N number of particles the probability of the particles' (α,β and up to N) position in space x, y, z and their velocity $\mathbf{v}$ at time t. By integrating over all space and velocity variables but the i-th particle, the distribution function is reduced to a one-particle function based on only seven variables (the six space and velocity variables and time), describing the probability of a particle to be found at the position x, y, z with the velocity v_x, v_y, v_z at time t. This six-dimensional (or $2f$ dimensional, where f is the degree of freedom) phase space x, y, z, v_x, v_y, v_z is also called the μ-phase space. This one-particle distribution function, with the according normalisation, equals the Boltzmann distribution function, which is written as:

$$f(x, y, z, v_x, v_y, v_z, t)dxdydzdv_xdv_ydv_z. \tag{2.2}$$

This is the number of particles that are contained in the spacial interval of $x + dx$, $y + dy$, and $z + dz$ with a velocity between v_x and $v_x + dv_x$, v_y and $v_y + dv_y$, and v_x and $v_x + dv_x$. From the distribution function, differential equations (with collisions) can be derived, where there are different methods to obtain the so-called collision equations from the Liouville function, which are described in any plasma physics text book, such as Boyd and Sanderson (2003).

When collisions are to be involved, there are two kind of interactions to be considered, the binary collisions and the many-body collisions. Binary collisions, the collisions of two particles, occur in neutral fluids with low density and short-ranged interaction forces. The collision term for that has been derived by Boltzmann and extended by Landau for plasmas; because for plasmas, long-ranged Coulomb forces lead to many body interactions and have to be taken into account (Cap, 1994, p. 67). Often, interactions in a plasma lead to small scattering angles or so-called weak interactions, where the potential energy of the interaction $\sim q_e^2/\lambda_D$ is much less than the mean thermal energy$\sim k_B T$ (Boyd and Sanderson, 2003, p. 307). In these circumstances, the Fokker-Planck equation can be applied. In general, to reduce the complexity of the theoretical description, approximations and as-

sumptions have to be made, which lead from the Liouville equation to the Fokker-Planck equation, the Boltzmann equation or the Lenard-Balescu equation.

In this thesis, the focus is on the fluid approach and the plasma is described with the MHD equations, which are obtained by starting at the Liouville equations and then applying the BBGKY (for Bogolyubov, Born, Green, Kirkwood and Yvon) approach (which handles the evolution of multiple particles) to get to the Boltzmann equation. In principle, for the BBGKY approach the Liouville equation is integrated step by step over the space and momentum of each particle. This and the derivation of the Boltzmann collision terms are also well explained in Cap (1994, p. 71), however, are beyond the scope of this thesis. The Boltzmann equation for a plasma reads:

$$\frac{df}{dt} \equiv \frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f + \frac{q_\alpha}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \frac{\partial f}{\partial \mathbf{v}} = \left(\frac{\partial f}{\partial t}\right)_{coll}. \tag{2.3}$$

The collision term on the RHS can be determined using different approaches with different underlying assumptions. For the Boltzmann collision term, only binary collisions are considered, whereas Landau, for instance, extended the collision integral to account for interactions in plasmas.

2.1.1 From kinetic theory to fluid theory

In the following, it is shown how to get from the Boltzmann equation to the fluid equations of a plasma, in the case when microscopic behaviour can be neglected. Firstly, it is assumed that the plasma is dense enough for the mean free path to be much shorter than the characteristic length scales over which the distribution function changes (Cap, 1994, p. 99). Being the first to establish this assumption, Hilbert also came up with an equation to describe exactly that and obtain the hydrodynamic description of plasma. To solve the equation Hilbert came up with, one needs conditions for the integrability of the function, where these conditions are exactly the hydrodynamic equations. However, depending on how the equation is solved, one gets to different hydrodynamic equations. Another, nowadays more common way to obtain the fluid equations is the methods of moments, which yields the distribution functions themselves. Here, for physical averages, i.e. moments, a differential equation is derived and solved. To derive the moment of r-th order, the Boltzmann equation (Eqn.(2.3)) is multiplied with a physical quantity Θ, which satisfies the

conservation laws (here $\Theta = mv^r$) and then integrated over all possible velocities (velocity space) to yield a function independent of details of the velocity space (Cap, 1994):

$$\int \Theta \left[\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f + \frac{q_\alpha}{m} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \frac{\partial f}{\partial \mathbf{v}} \right] d\mathbf{v} = \int \Theta \left(\frac{\partial f}{\partial t} \right)_{coll} d\mathbf{v}. \tag{2.4}$$

Since $f(\mathbf{r}, \mathbf{v}, t) d\mathbf{r} d\mathbf{v}$ is the probability of finding particles of type α at time t within a small volume element $d\mathbf{r}d\mathbf{v}$, its integral over the velocity space is the probability of finding the particles within the volume $d\mathbf{r}$, irrespective of velocity (Boyd and Sanderson, 2003, p. 480).
With $\mathbf{F}$ being the forces, we yield:

$$\frac{\partial}{\partial t} \int f m \mathbf{v}^r d\mathbf{v} + \nabla \cdot \int m \mathbf{v}^r \mathbf{v} f + \frac{1}{m} \int \mathbf{F} m \mathbf{v}^r \cdot \frac{\partial f}{\partial \mathbf{v}} d\mathbf{v} = \left(\frac{\partial}{\partial t} \right)_{coll} \int f m \mathbf{v}^r d\mathbf{v}. \tag{2.5}$$

The RHS describes for $r = 0$ (moment of zeroth order) the change of the total mass density, for $r = 1$ (moment of first order) the change of the total momentum and for $r = 2$ (moment of second order) the change of the total (kinetic) energy, due to the interactions of particles through collisions. In the case of elastic collisions or like-particle collisions, the quantities are conserved and the RHS is zero. With

$$\int f d\mathbf{v} = n, \quad n\langle m\mathbf{v}^r \rangle = \int m \mathbf{v}^r f d\mathbf{v}, \quad n\langle \Theta \rangle = \int \Theta f d\mathbf{v}, \tag{2.6}$$

we can define the r-th moment as the average and write:

$$\langle m\mathbf{v}^r \rangle = \frac{\int m \mathbf{v}^r \mathbf{v} f d\mathbf{v}}{\int f d\mathbf{v}}. \tag{2.7}$$

For any r, we yield the Maxwell-Boltzmann transport equation

$$\frac{\partial}{\partial t} (n\langle m\mathbf{v}^r \rangle) + \nabla \cdot (n\langle m\mathbf{v}^{r+1} \rangle - \frac{n}{m} \langle m\mathbf{F} \frac{\partial \mathbf{v}^r}{\partial \mathbf{v}} \rangle = \int m \mathbf{v}^r \left(\frac{\partial f}{\partial t} \right)_{coll} d\mathbf{v}. \tag{2.8}$$

More general, this would read as:

$$\frac{\partial}{\partial t} (n\langle \Theta \rangle) + \nabla \cdot (n\langle \Theta \mathbf{v} \rangle - \frac{n}{m} \langle m\mathbf{F} \frac{\partial \Theta}{\partial \mathbf{v}} \rangle = \int \Theta \left(\frac{\partial f}{\partial t} \right)_{coll} d\mathbf{v}. \tag{2.9}$$

(Cap, 1994). This means, a macroscopic variable $\langle g \rangle(\mathbf{r}, t)$ is the average of a phase space function $g(\mathbf{r}, \mathbf{v}, t)$.

First, a few words about the general meaning of the moments of different orders, which have the following meaning in the plasma physics sense:

1. $r = 0$
   particle density (if $m = 1$):

$$n(x,t) = \int f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v}. \tag{2.10}$$

   mass density:

$$n\langle m \rangle = \rho(\mathbf{x}, t) = mn(\mathbf{x}, t) = m \int f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v}. \tag{2.11}$$

2. $r = 1$
   particle flux (if $m = 1$):

$$n\mathbf{v} = \int \mathbf{v} f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v}. \tag{2.12}$$

   mass flux:

$$n\langle m\mathbf{v} \rangle = \rho\mathbf{v} = m \int \mathbf{v} f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v}. \tag{2.13}$$

3. $r = 2$
   momentum flux:

$$n\langle m\mathbf{v}\mathbf{v} \rangle = \rho\langle \mathbf{v}, \mathbf{v} \rangle = \int m\mathbf{v}\mathbf{v} f d\mathbf{v}. \tag{2.14}$$

The flux of the energy density $\Theta = \frac{1}{2}mv^2$, yields:

$$n\langle\frac{1}{2}mv^2\mathbf{v}\rangle = \frac{m}{2}\int v^2\mathbf{v}f d\mathbf{v}. \tag{2.15}$$

With those velocity integrals, which are called moment integrals of a function f, we have now yielded fluid variables. Now, we want to obtain the evolution equations for these macroscopic variables by taking appropriate moments of the kinetic equation (Boyd and Sanderson, 2003, p. 482). Applying the moment method to the Maxwell-Boltzmann equation in Eqn. (2.8), for $r = 0$ the continuity equation is derived (Cap, 1994, p. 103):

$$\frac{\partial\rho}{\partial t} + \nabla(\rho\mathbf{v}) = 0 \tag{2.16}$$

and for $r = 1$ the momentum equation is obtained:

$$\frac{\partial}{\partial t}(\rho\mathbf{v}) + \nabla\cdot(\rho\langle\mathbf{vv}\rangle) - n\mathbf{F} = 0 \tag{2.17}$$

and for $r = 2$ and by multiplying with energy density, we yield the energy equation:

$$\frac{\partial}{\partial t}\left(n\frac{1}{2}m\langle v^2\rangle\right) + \nabla\cdot\left(n\frac{1}{2}m\langle v^2\mathbf{v}\rangle\right) - qn\mathbf{E}\cdot\mathbf{v} = 0. \tag{2.18}$$

The RHS is treated with according collision terms. As mentioned before, in fluid theory, the variables are a function of four variables, the three space components x, y, z and time t. This independence of the velocity is possible because the velocity distribution is assumed to be Maxwellian everywhere and, therefore, can be determined with the temperature T only (Chen, 1984, p.225).

2.1.2 Single-fluid MHD

A plasma consisting of different species, where each species can be described by its own set of fluid equations, can be simplified to a single, conducting fluid, described by just one set of equations. To describe this macroscopic behaviour of the plasma as a whole,

the contributions of the various particle species are added up, and we end up with the total macroscopic parameters (Bittencourt, 2004). In this process, information gets lost, but time is gained when solving only one set of equations. The species we combine here, however, are only electrons and ions, as the neutrals are considered as a separate fluid in this work, and therefore, less information is lost compared to combining all three.

Following Bittencourt (2004), the mass density would be:

$$\rho_m = \sum_\alpha \rho_{m\alpha} = \sum_\alpha n_\alpha m_\alpha \tag{2.19}$$

and the mean fluid velocity, $\mathbf{v}$ is defined through the total momentum density:

$$\rho_m \mathbf{v} = \sum_\alpha \rho_{m\alpha} \mathbf{v}_\alpha, \tag{2.20}$$

where $\mathbf{v}$ is a weighted mean value, where the velocity of each species is weighted proportionally to its mass density. Here, the mean velocity is, therefore, essentially the velocity of the ions. The mean velocity of each particle species, is called the diffusion velocity $\mathbf{w}_\alpha$:

$$\mathbf{w}_\alpha = \mathbf{v}_\alpha - \mathbf{v} = \mathbf{v}_\alpha - \frac{1}{\rho_m} \sum_\alpha \rho_{m\alpha} \mathbf{v}_\alpha. \tag{2.21}$$

The mass flux (also mass current density) is:

$$\mathbf{j}_m = \sum_\alpha n_\alpha m_\alpha \mathbf{v}_\alpha = \rho_m \mathbf{v}. \tag{2.22}$$

The charge flux (or electric current density) is given by:

$$\mathbf{j} = \sum_\alpha n_\alpha q_\alpha \mathbf{v}_\alpha = \rho \mathbf{v} + \sum_\alpha n_\alpha q_\alpha \mathbf{w}_\alpha, \tag{2.23}$$

where q is the particle charge. As the diffusion velocity of the electrons is greater than the ion diffusion velocity, the electric current density is determined by the electrons. A random velocity is introduced as $c_\alpha = \mathbf{v}_{pa} - \mathbf{v}_\alpha$, where $\mathbf{v}_{pa}$ is the particle velocity and $\mathbf{v}_\alpha$ the fluid velocity. The pressure is defined as the time in which momentum is transported by the particles through a surface element moving with the mean particle velocity. The

total kinetic pressure tensor is:

$$\mathbf{p} = \sum_{\alpha} \mathbf{p}_{\alpha} + \sum_{\alpha} \rho_{m\alpha} \mathbf{w}_{\alpha} \mathbf{w}_{\alpha}, \tag{2.24}$$

where $\mathbf{p}_{\alpha}$ is the pressure relative to a particles mean velocity and $\mathbf{p}$ is the pressure relative to the global mean velocity. For random velocities, the tensor can be expressed by the pressure scalar. The total scalar pressure P is defined as one-third the trace of $\mathbf{p}$, $P = \frac{1}{3}\sum_i \mathbf{p}_{ii}$, and the scalar pressure tensor is written as:

$$P = \sum_{\alpha} P_{\alpha} + \frac{1}{3} \sum_{\alpha} \rho_{m\alpha} w_{\alpha}^2. \tag{2.25}$$

The single-fluid equations derived by adding up the contributions of all species, lead to the vanishing of the collision term, as the density, momentum and energy are conserved.

The correct relationship between currents and field is given by the generalised Ohm's law (Boyd and Sanderson, 2003, p.64). Although this would need a two-fluid approach to be derived correctly, the inequality of the masses of ions and electrons and the condition of charge neutrality lead to an Ohm's law that is applicable in the MHD approximation. The generalised Ohm's law is derived by combining the equations of motion with which we obtain a charge-weighted average and which can be used to extend the ideal MHD or resistive MHD model (Goedbloed and Poedts, 2004, p. 562). Adding the equations of motion for the ions and electrons, which can be expressed as:

$$m_i n \frac{\partial \mathbf{v_i}}{\partial t} = q_e n(\mathbf{E} + \mathbf{v}_i \times \mathbf{B}) - \nabla P_i + m_i n \mathbf{g} + \mathbf{C}_{ie} \tag{2.26}$$

$$m_e n \frac{\partial \mathbf{v_e}}{\partial t} = q_e n(\mathbf{E} + \mathbf{v}_e \times \mathbf{B}) - \nabla P_e + m_e n \mathbf{g} + \mathbf{C}_{ei}, \tag{2.27}$$

the collision term, here $\mathbf{C}_{ie} = -\mathbf{C}_{ei}$, as well as the electric field cancel out and we yield:

$$\rho \frac{\partial \mathbf{v}}{\partial t} = (\mathbf{j} \times \mathbf{B}) - \nabla P + \rho \mathbf{g}. \tag{2.28}$$

As the fluid is neutral, the electric field does not appear explicitly (Chen, 1984, p. 173). If we take the momentum equations, Eqns. (2.26) and (2.27), and now multiply one with

the mass of the other and subtract the electron equation from the ion equation, we obtain:

$$m_i m_e n \frac{\partial}{\partial t}(\mathbf{v}_i - \mathbf{v}_e) = q_e n(m_i + m_e)\mathbf{E} + q_e n(m_e \mathbf{v}_i + m_i \mathbf{v}_e) \times \mathbf{B} - m\nabla P_i + m_i \nabla P_e - (m_i + m_e)P_{ei}, \tag{2.29}$$

which, following Chen (1984, p. 173), becomes

$$\mathbf{E} + \mathbf{v} \times \mathbf{B} = \eta \mathbf{j} + \frac{1}{q_e \rho}\left[\frac{m_i m_e n}{q_e}\frac{\partial}{\partial t}\frac{\mathbf{j}}{n} + (m_i - m_e)\mathbf{j} \times \mathbf{B} + m_e \nabla P_i - m_i \nabla P_e)\right], \tag{2.30}$$

which is the generalised Ohm's law and describes the electric properties of a conducting fluid. Because the $\frac{\partial}{\partial t}$ term can be neglected where inertial effects (like the cyclotron frequency) are not important, i.e. in slow motions, this can be further simplified, and in the limit $m_e/m_i \to 0$ this yields:

$$\mathbf{E} + \mathbf{v} \times \mathbf{B} - \eta \mathbf{j} = \frac{1}{q_e n}(\mathbf{j} \times \mathbf{B} - \nabla P_e) \tag{2.31}$$

Often the Hall term ($\mathbf{j} \times \mathbf{B}$) and the ∇P_e term are small enough and can be neglected, and the equation reduces to:

$$\mathbf{E} + \mathbf{v} \times \mathbf{B} = \eta \mathbf{j}, \tag{2.32}$$

which is called the resistive MHD Ohm's law. If the resistivity (or diffusivity) η is neglected too, the ideal MHD Ohm's law is obtained. In this work, Ohm's law is integrated in the induction equation of the set of equations for the charged fluid. There, Ohm's law and Faraday's law are used to express the electric field and electric current density in terms of the magnetic field and the velocity.

2.1.3 Two-Fluid (ion-neutral) MHD

The code developed here covers the two-fluid ion neutral model for partially ionised plasmas. This means that there are two sets of equations, one for the ionised fluid and one for the neutrals. From the above sections one can sum up the approximations and assumptions made for the two-fluid (neutral-ion) MHD model applied here. Apart from fundamental plasma such as charge-neutrality, for an ideal gas (like our MHD set of equations), there

are no forces between the particles and, therefore, no Coulomb forces (Cap, 1994, p. 115). Moreover, pure hydrogen plasma is assumed and, therefore, the ions are of only one kind and ions and electrons have the same temperature. Furthermore, it is assumed that the electron inertia can be neglected, as the electron mass is much lower than the ions' mass. However, in fact, a centre-of-mass velocity, the velocity of the ionised fluid equals the ions' velocity, whereas the electron's velocity could be calculated from the current density **j**, which is expressed through **B** with the Maxwell equations (Cap, 1994, p. 106). However, in the derivation of the single-fluid MHD equations from an electron-ion two-fluid equation, the collision terms containing **j** that are proportional to the electron density, are neglected.

In the following, the macroscopic derivation of the hydrodynamic and MHD equations are given. The chapter ends with a derivation and explanation of the source terms, including the collision terms, which are the terms connecting the fluids.

2.2 Derivation of the Hydrodynamic and MHD equations

The theoretical models of plasma lead to non-linear PDEs (Goedbloed and Poedts, 2004, p. 3). The PDEs solved for the two fluid model are equations describing neutral fluid flow in terms of hydrodynamics and plasma dynamics in terms of MHD. When considering fluid flow, there are two ways of looking at the fluid and modelling it. We can either consider a control volume V (which is a finite region of the flow with a control surface S) or an infinitesimal fluid element with volume dV. Both can be either fixed in space (and we obtain the conservation form) or moving with the flow, which is referred to as the Eulerian and Lagrangian frames of reference, respectively (Wendt *et al.*, 2009). In our code, we consider a finite control volume V (which directly leads to the integral form of the equations) in the Eulerian framework. Fluid flow is described with equations based on conservation laws. Conservation laws are a natural consequence of a central postulate which states that there is balance in a physical system and, therefore, in a closed volume, the production of mass, energy, or charge for example, is balanced by the flux of the same across the boundaries of the volume (Hesthaven, 2018, p.1). Following Hesthaven (2018), general conservation laws can be mathematically expressed and derived as follows.

We consider a mass per unit length, or the density $\rho(x,t)$, of a fluid that is distributed in the spatial domain $x \in [x_1, x_2] = \Omega_x$. Hence the total mass is given as:

$$M(t) = \int_{\Omega_x} \rho(x,t)dx. \tag{2.33}$$

If $v(x,t)$ is the local velocity of the fluid at position x and time t, mass conservation is expressed as

$$\frac{d}{dt}\int_{\Omega_x} \rho(x,t)dx = \rho(x_1,t)v(x_1,t) - \rho(x_2,t)v(x_2,t). \tag{2.34}$$

This means that, assuming $v > 0$, mass can only be gained by a higher inward mass flux at x_1 than outward mass flux at x_2. Integrating over a temporal domain $\Omega_t = [t_1, t_2]$, we recover

$$\int_{\Omega_x} \rho(x,t_2)dx - \int_{\Omega_x} \rho(x,t_1)dx = \int_{\Omega_t} \rho(x_1,t)v(x_1,t) - \rho(x_2,t)v(x_2,t)dt, \tag{2.35}$$

which is a direct consequence of the basic principle that mass cannot be created or disappear by itself. Assuming that the density and the velocity are both differentiable functions at (x,t), we obtain with the Fundamental Theorem of Calculus:

$$\rho(x,t_2) - \rho(x,t_1) = \frac{d}{dt}\int_{\Omega_t} \rho(x,t)dt = \int_{\Omega_t} \frac{\partial}{\partial t}\rho(x,t)dt, \tag{2.36}$$

or, equivalently:

$$\rho(x_2,t)v(x_2,t) - \rho(x_1,t)v(x_1,t) = \frac{d}{dx}\int_{\Omega_x} \rho(x,t)v(x,t)dx = \int_{\Omega_x} \frac{\partial}{\partial x}\rho(x,t)v(x,t)dx. \tag{2.37}$$

Combining the theorem with Eqn. (2.35), we obtain

$$\int_{\Omega_x}\int_{\Omega_t} \left[\frac{\partial \rho(x,t)}{\partial t} + \frac{\partial}{\partial x}\rho(x,t)v(x,t)\right] dtdx = 0. \tag{2.38}$$

Since this must hold for any volume $\Omega_x \times \Omega_t$, it must hold in a pointwise sense and we recover the conservation law in differential form:

$$\frac{\partial \rho(x,t)}{\partial t} + \frac{\partial}{\partial x}\rho(x,t)v(x,t) = 0, \qquad (x,t) \in \Omega_x \times \Omega_t. \tag{2.39}$$

As this applies to any conserved variable, say $u(x,t)$ and the associated flux $f(u,t)$, we

yield for the conservation law in integral form:

$$\int_{\Omega_x}\int_{\Omega_t}\left[\frac{\partial u(x,t)}{\partial t}+\frac{\partial}{\partial x}f(u,t)\right]dtdx=0, \tag{2.40}$$

or, in differential form:

$$\frac{\partial u(x,t)}{\partial t}+\frac{\partial}{\partial x}f(u,t)=0, \qquad (x,t)\in\Omega_x\times\Omega_t. \tag{2.41}$$

In multi dimensions, a conservation law is expressed as

$$\frac{d}{dt}\int_{\Omega_x}\mathbf{u}(\mathbf{x},t)dV=\oint_{\partial\Omega_x}\hat{\mathbf{n}}\cdot\mathbf{f}(\mathbf{u},\mathbf{x},t)dS, \qquad \mathbf{x}\in\Omega_x\subset R^d, \quad t>0, \tag{2.42}$$

now, $\mathbf{u}(\mathbf{x},t):\Omega_x\times\Omega_t\rightarrow R^m$ is the dependent variable, $\hat{\mathbf{n}}$ is the outward pointing unit vector along the boundary $\partial\Omega_x$ of Ω_x and $\mathbf{f}(\mathbf{u},\mathbf{x},t):R^m\times\Omega_x\times\Omega_t\rightarrow R^m$ is the flux through the boundary. Assuming that the solution and the flux are differentiable and applying Gauss' theorem over a control volume, we yield

$$\int_{\Omega_t}\int_{\Omega_x}\left[\frac{\partial\mathbf{u}(\mathbf{x},t)}{\partial t}+\nabla\cdot\mathbf{f}(\mathbf{u},\mathbf{x},t)\right]dxdt=0. \tag{2.43}$$

As this, again, holds for any control volume, we recover the general conservation law in differential form

$$\frac{\partial\mathbf{u}(\mathbf{x},t)}{\partial t}+\nabla\cdot\mathbf{f}(\mathbf{u},\mathbf{x},t)=0, \qquad \mathbf{x}\in\Omega_x, \quad t>0. \tag{2.44}$$

The above is a system of m conservation laws in a d-dimensional space. Again, our two fluid model requires two systems of equations: one governing the dynamics of the neutral fluid (hydrodynamic equations) and one governing the dynamics of the ionised fluid (MHD equations).

2.2.1 Derivation of Hydrodynamic Equations

Hydrodynamic or fluid dynamic equations are based on the fundamental principles of mass, momentum and energy conservation. The following derivation will be based on Wendt *et al.* (2009). For the continuity equation or the conservation of mass, a control volume V is considered; the net mass flow out of this volume through the surface S (LHS

of Eqn. (2.45)) equals the rate of decrease of mass inside the volume (RHS of Eqn. (2.45)). This leads to the following mathematical expression:

$$\oiint_S \rho \mathbf{v} \cdot \mathbf{d}S = -\frac{\partial}{\partial t} \iiint_V \rho dV \tag{2.45}$$

or

$$\frac{\partial}{\partial t} \iiint_V \rho dV + \oiint_S \rho \mathbf{v} \cdot \mathbf{d}S = 0, \tag{2.46}$$

which is the conservation and integral form of the continuity equation. The conservation and partial differential form is written as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{v} = 0. \tag{2.47}$$

The momentum equation is derived by applying Newton's second law, which is:

$$\mathbf{F} = m\mathbf{a}. \tag{2.48}$$

In the fluid dynamics case, the force **F** described by Newton's law is the net force on the fluid element. There are two sources of forces: firstly, there are the body forces which "act at a distance" and directly on the volumetric mass of the fluid element, like gravitational and electromagnetic forces. Secondly, there are surface or contact forces, which act directly on the surface of the fluid element and are basically due to the pressure distribution and the shear and normal stress distributions that are imposed by the surrounding fluid. We denote the body force acting in x-direction with $\rho f_x(dxdydz)$, the shear stress with τ_{xy} and the normal stress with τ_{xx}. Normal stresses are usually much smaller than shear stresses and only become important when the gradient of the velocity are very large like, for example inside shock waves. With the pressure force, $-p$, acting inwards with respect to the fluid element, we get to the equation for the total force in x-direction:

$$F_x = \left(-\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} \right) dxdydz + \rho f_x dxdydz. \tag{2.49}$$

Furthermore, as the mass of the fluid element is fixed, and with:

$$\mathbf{F} = m\mathbf{a} \qquad \text{where} \qquad m = \rho dxdydz \qquad \text{and} \qquad a_x = \frac{Dv_x}{Dt}, \tag{2.50}$$

where $\frac{Dv_x}{Dt}$ is the substantial derivative of the velocity in x-direction u, we obtain the x-component of the momentum equation

$$\rho\frac{Dv_x}{Dt} = \left(-\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}\right) + \rho f_x, \tag{2.51}$$

and, similarly, the y- and z-components:

$$\rho\frac{Dv_y}{Dt} = \left(-\frac{\partial p}{\partial y} + \frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}\right) + \rho f_y, \tag{2.52}$$

$$\rho\frac{Dv_z}{Dt} = \left(-\frac{\partial p}{\partial z} + \frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}\right) + \rho f_z. \tag{2.53}$$

These equations are also called Navier-Stokes equations and are partial differential equations in non-conservation form. Those scalar equations can be formed into equations of conservation form by applying the definition of the substantial derivative, namely:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{v}\cdot\nabla \tag{2.54}$$

or in the case of Eqn. (2.51):

$$\rho\frac{Dv_x}{Dt} = \rho\frac{\partial v_x}{\partial t} + \rho\mathbf{v}\cdot\nabla v_x. \tag{2.55}$$

In addition, we expand the derivative and yield

$$\frac{\partial(\rho v_x)}{\partial t} = \rho\frac{\partial v_x}{\partial t} + v_x\frac{\partial \rho}{\partial t} \qquad \text{or} \qquad \rho\frac{\partial v_x}{\partial t} = \frac{\partial(\rho v_x)}{\partial t} - v_x\frac{\partial \rho}{\partial t}. \tag{2.56}$$

With the vector identity for the divergence of a scalar and a vector, namely:

$$\nabla\cdot(\rho v_x\mathbf{v}) = v_x\nabla\cdot(\rho\mathbf{v}) + (\rho\mathbf{v})\cdot\nabla v_x \qquad \text{or} \qquad \rho\mathbf{v}\cdot\nabla v_x = \nabla\cdot(\rho v_x\mathbf{v}) - v_x\nabla\cdot(\rho\mathbf{v}) \tag{2.57}$$

and by substituting the above equations into Eqn. (2.55), we obtain:

$$\rho\frac{Dv_x}{Dt} = \frac{\partial(\rho v_x)}{\partial t} - v_x\frac{\partial \rho}{\partial t} - v_x\nabla\cdot(\rho\mathbf{v}) + \nabla\cdot(\rho v_x\mathbf{v}) \tag{2.58}$$

or

$$\rho\frac{Dv_x}{Dt} = \frac{\partial(\rho v_x)}{\partial t} - v_x\left[\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v})\right] + \nabla\cdot(\rho v_x\mathbf{v}). \tag{2.59}$$

As the term in the square brackets is just LHS of the continuity equation (Eqn. (2.47)) and, therefore, zero we get for the conversion of our momentum equation from non-conservation form to conservation form:

$$\rho\frac{Dv_x}{Dt} = \frac{\partial(\rho v_x)}{\partial t} + \nabla\cdot(\rho v_x\mathbf{v}), \tag{2.60}$$

which leads to the following components of the momentum equation in x, y and z direction in partial differential and conservation form:

$$\begin{aligned}
\frac{\partial(\rho v_x)}{\partial t} + \nabla\cdot(\rho v_x\mathbf{v}) &= \left(-\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}\right) + \rho f_x \\
\frac{\partial(\rho v_y)}{\partial t} + \nabla\cdot(\rho v_y\mathbf{v}) &= \left(-\frac{\partial p}{\partial y} + \frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}\right) + \rho f_y \\
\frac{\partial(\rho v_z)}{\partial t} + \nabla\cdot(\rho v_z\mathbf{v}) &= \left(-\frac{\partial p}{\partial z} + \frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}\right) + \rho f_z.
\end{aligned} \tag{2.61}$$

The energy equation is based on the first law of thermodynamics which states that the energy of an isolated system is constant, hence energy can neither be destroyed nor created. Considering a fluid element, this means that the rate of change of energy inside it equals the net flux of heat into it plus the rate of work done on the fluid element due to body and contact forces. The work done by a force is the product of the force times the component of the velocity in the direction of the force. Considering the derivation of the body and contact force terms for the momentum equation (RHS of Eqn. (2.61)), we get for the work done by these forces, arbitrarily denoted as C:

$$C = \Bigg[-\left(\frac{\partial v_x p}{\partial x} + \frac{\partial v_y p}{\partial y} + \frac{\partial v_z p}{\partial z}\right) + \frac{\partial v_x \tau_{xx}}{\partial x} + \frac{\partial v_y \tau_{xy}}{\partial y} + \frac{\partial v_z \tau_{xz}}{\partial z} + \frac{\partial v_x \tau_{yx}}{\partial x} + \frac{\partial v_y \tau_{yy}}{\partial y} + \frac{\partial v_z \tau_{yz}}{\partial z} + \frac{\partial v_x \tau_{zx}}{\partial x} + \frac{\partial v_y \tau_{zy}}{\partial y} + \frac{\partial v_z \tau_{zz}}{\partial z} \Bigg] dxdydz + \rho \mathbf{f} \cdot \mathbf{v} dxdydz. \tag{2.62}$$

Here, C is the combined terms for the work done on the fluid element, and we now consider the change of energy due to heat flux, which is a result of volumetric heating, e.g. absorption or emission of radiation and the heat transfer across the surface due to temperature gradients, also known as thermal conduction.

With the mass of the fluid element being $\rho dxdydz$ and the denotation of q as the rate of the volumetric heat addition per unit mass, our volumetric heating term equals $\rho q dxdydz$. The thermal conduction term is obtained by considering the heat transferred into the fluid element (in x-direction this would be $q_x dydz$) and the heat transferred out of the fluid element ($[q_x + (\frac{\partial q_x}{\partial x})dx]dydz$). The net heating in x-direction, q_{netx}, then is

$$q_{netx} = \left[q_x - \left(q_x + \frac{\partial q_x}{\partial x} \right) dx \right] dydz = -\frac{\partial q_x}{\partial x} dxdydz, \tag{2.63}$$

or for all directions:

$$q_{net} = -\left(\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} \right) dxdydz. \tag{2.64}$$

The volumetric heating together with the heating by thermal conduction then equals:

$$Q = \left[\rho q - \left(\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} \right) \right] dxdydz. \tag{2.65}$$

Because the heat transfer by thermal conduction is proportional to the local temperature gradient (Braginskii, 1965, p. 221),

$$q_x = -k\frac{\partial T}{\partial x}; \qquad q_y = -k\frac{\partial T}{\partial y}; \qquad q_z = -k\frac{\partial T}{\partial z}, \tag{2.66}$$

where k is a proportionality constant, we can replace q_x, q_y, q_z in Eqn. (2.65) and arrive at

$$Q = \left[\rho q + \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)\right] dxdydz, \tag{2.67}$$

for the net flux of heat into the fluid element. The total energy of our moving fluid element is the sum of its internal energy e and its kinetic energy $\mathbf{v}^2/2$ per unit mass, and hence the time rate of change of energy is as follows:

$$E = \frac{D}{Dt}\left(e + \frac{\mathbf{v}^2}{2}\right)\rho dxdydz. \tag{2.68}$$

The energy equation then is:

$$\begin{aligned}\rho\frac{D}{Dt}\left(e + \frac{\mathbf{v}^2}{2}\right) \quad = \quad & \rho q + \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right) \\ & - \left(\frac{\partial v_x p}{\partial x} + \frac{\partial v_y p}{\partial y} + \frac{\partial v_z p}{\partial z}\right) + \frac{\partial v_x \tau_{xx}}{\partial x} + \frac{\partial v_y \tau_{xy}}{\partial y} + \frac{\partial v_z \tau_{xz}}{\partial z} \\ & + \frac{\partial v_x \tau_{yx}}{\partial x} + \frac{\partial v_y \tau_{yy}}{\partial y} + \frac{\partial v_z \tau_{yz}}{\partial z} + \frac{\partial v_x \tau_{zx}}{\partial x} + \frac{\partial v_y \tau_{zy}}{\partial y} + \frac{\partial v_z \tau_{zz}}{\partial z} + \rho\mathbf{f}\cdot\mathbf{v}\end{aligned} \tag{2.69}$$

which is the total energy equation in non-conservation form. To write the energy equation in terms of the internal energy only, we take the momentum equations in x, y and z directions (Eqn. (2.61)) and multiply them on the LHS by $v_x^2/2, v_y^2/2$ and $v_z^2/2$, respectively. Adding those three altered momentum equations together, we get

$$\begin{aligned}\rho\frac{D}{Dt}\left(\frac{\mathbf{v}^2}{2}\right) \quad = \quad & -v_x\frac{\partial p}{\partial x} - v_y\frac{\partial p}{\partial y} - v_z\frac{\partial p}{\partial z} + v_x\left(\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z}\right) \\ & + v_y\left(\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z}\right) + v_z\left(\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}\right) + \rho(v_x f_x + v_y f_y + v_z f_z).\end{aligned} \tag{2.70}$$

Subtracting (2.70) from (2.69) and noting that $\rho\mathbf{f}\cdot\mathbf{v} = \rho(v_x f_x + v_y f_y + v_z f_z)$ the energy equation in terms of the internal energy only becomes:

$$\begin{aligned}\rho\frac{De}{Dt} &= \rho q+\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)\\ &\quad -p\left(\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}+\frac{\partial v_z}{\partial z}\right)+\tau_{xx}\frac{\partial v_x}{\partial x}+\tau_{xy}\frac{\partial v_y}{\partial y}+\tau_{xz}\frac{\partial v_z}{\partial z}\\ &\quad +\tau_{yx}\frac{\partial v_x}{\partial x}+\tau_{yy}\frac{\partial v_y}{\partial y}+\tau_{yz}\frac{\partial v_z}{\partial z}+\tau_{zx}\frac{\partial v_x}{\partial x}+\tau_{zy}\frac{\partial v_y}{\partial y}+\tau_{zz}\frac{\partial v_z}{\partial z}\end{aligned} \tag{2.71}$$

and as done with the momentum equation, we can get the energy equation in conservation form by applying the definition for the substantial derivative (Eqn. (2.54)) and the vector identity (Eqn. (2.57)) and obtain:

$$\begin{aligned}\frac{\partial(\rho e)}{\partial t}+\nabla\cdot(\rho e\mathbf{v}) &= \rho q+\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)\\ &\quad -p\left(\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}+\frac{\partial v_z}{\partial z}\right)+\tau_{xx}\frac{\partial v_x}{\partial x}+\tau_{xy}\frac{\partial v_y}{\partial y}+\tau_{xz}\frac{\partial v_z}{\partial z}\\ &\quad +\tau_{yx}\frac{\partial v_x}{\partial x}+\tau_{yy}\frac{\partial v_y}{\partial y}+\tau_{yz}\frac{\partial v_z}{\partial z}+\tau_{zx}\frac{\partial v_x}{\partial x}+\tau_{zy}\frac{\partial v_y}{\partial y}+\tau_{zz}\frac{\partial v_z}{\partial z}\end{aligned} \tag{2.72}$$

which is the energy equation in terms of the internal energy in conservation form. The above equations express fluid behaviour for a viscid flow, and hence, consider the dissipative, transport phenomena of viscosity and thermal conduction. If those phenomena are neglected we talk about inviscid flow and with neglecting shear and normal stress the equations simplify to:

$$\begin{aligned}\frac{\partial\rho}{\partial t}+\nabla\cdot(\rho\mathbf{v}) &= 0.\\ \frac{\partial\rho v_x}{\partial t}+\nabla\cdot(\rho v_x\mathbf{v}) &= -\frac{\partial P}{\partial x}+\rho f_x,\\ \frac{\partial\rho v_y}{\partial t}+\nabla\cdot(\rho v_y\mathbf{v}) &= -\frac{\partial P}{\partial y}+\rho f_y,\\ \frac{\partial\rho v_z}{\partial t}+\nabla\cdot(\rho v_z\mathbf{v}) &= -\frac{\partial P}{\partial z}+\rho f_z,\\ \frac{\partial e}{\partial t}+\nabla\cdot(\rho\mathbf{v}e) &= \rho q-\nabla\cdot\mathbf{v}P+\rho\mathbf{f}\cdot\mathbf{v}.\end{aligned} \tag{2.73}$$

The energy equation here is given in terms of the energy per unit mass ($\rho dxdydz$) and if this is one, the full hydrodynamic equations, without forces, are written as:

$$\begin{aligned} \frac{\partial}{\partial t}\rho + \nabla \cdot (\rho \mathbf{v}) &= 0 \\ \frac{\partial}{\partial t}\rho \mathbf{v} + \nabla \cdot (\rho \mathbf{v}\mathbf{v} + P\mathbf{I}) &= 0 \\ \frac{\partial}{\partial t}e + \nabla \cdot (e + P)\mathbf{v} &= 0 \end{aligned} \tag{2.74}$$

2.2.2 Derivation of Magnetohydrodynamic (MHD) Equations

The equations governing our ionised fluid are, in essence, the same governing our neutral fluid. Additionally, the fluid is ionised (plasma) and, therefore, electrically conducting, and non-magnetic (Davidson, 2001, p. 3). In its core, the fluid theory of plasma and the macroscopic plasma dynamics are about the interaction of plasma motion as a whole within the magnetic field geometry. Plasma motion is influenced by magnetic fields, which are, in turn, generated by the plasma motion itself, and MHD is a way to describe this global interplay of plasma and magnetic field (Goedbloed and Poedts, 2004, p. 28). This requires, apart from the continuity, momentum and energy equations, an equation that describes the evolution of the magnetic field. Therefore, fluid dynamic equations are combined with Maxwell equations. Furthermore, an expression to describe the interaction of the magnetic field with the fluid flow or velocity field of the fluid has to be provided. The velocity of the electrons is what creates currents and therefore influences the magnetic field, which, again, influences the fluid.

Starting from the equations to evaluate the electromagnetic field, the Maxwell equations are derived. Then, in order to combine them and account for the interaction with the velocity field, the induction equation is modified. The electromagnetic field equations are a compound of four equations and together are known as the Maxwell equations. These four Maxwell equations are:

$$\nabla \cdot \mathbf{E} = \frac{\rho^*}{\epsilon_0}, \tag{2.75}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{2.76}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \tag{2.77}$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t}, \tag{2.78}$$

where $\mathbf{E}$ is the electric field, $\mathbf{B}$ is the magnetic field, ϵ_0 is the electric permeability of free space, ρ^* is the charge density, μ_0 is the magnetic permeability of free space, j is the current and c is the speed of light. Maxwell's set of equations to describe the evolution and interaction of electric and magnetic fields with matter is based on research and ideas of great mathematicians and physicists like Gauss, Ampère and Faraday (Mattis, 1965).

Gauss' law, as seen in Eqn. (2.75), states that the divergence of an electric field depends on the electric charge (Priest, 2014, p. 75). We get to this expression by starting from Coulomb's law of electrostatics in 1785 (Eqn. 2.2.2), which gives us the magnitude and direction of the force $\mathbf{F}$ between two charge carrying particles q_1, q_2 at distance r_{21} (Grant and Phillips, 1999, pp. 2).

$$\mathbf{F} = \frac{1}{4\pi\epsilon_0}\frac{q_2 q_1}{r_{21}^3}\mathbf{r}_{21}. \tag{2.79}$$

Considering a point charge q at the centre of a sphere of the radius r, the electric field on its closed surface S with the surface area $A = 4\pi r^2$ is everywhere of magnitude $E = q/4\pi r^2\epsilon_0$, with ϵ_0 being the permittivity of free space constant. The product of area and electric field $A \cdot E$ becomes q/ϵ_0 and shows the independence on the radius (Grant and Phillips, 1999, p. 16). If we, now, do not have a single charge, but a number of charges, the total field can be determined with the principle of superposition:

$$\mathbf{E} = \sum_{i=1}^{n} \mathbf{E}_i, \tag{2.80}$$

which says that the total field is the sum of all the individual fields (Griffiths, 1999, p. 68).

The flux Φ_E coming from all of the charges together and going through the sum of the infinitesimal area $d\mathbf{a}$, i.e. the closed surface S, that encloses the total charge Q_{encl}, then, is :

$$\Phi_E \equiv \oint \mathbf{E} \cdot d\mathbf{a} = \sum_{i=1}^{n} \left(\oint \mathbf{E_i} \cdot d\mathbf{a} \right) = \sum_{i=1}^{n} \left(\frac{q_i}{\epsilon_0} \right). \tag{2.81}$$

As this holds for any surface S, the total electric flux Φ_E or the flux of $\mathbf{E}$ going through a closed surface S is:

$$\Phi_E = \oint_S \mathbf{E} \cdot d\mathbf{a} = \frac{1}{\epsilon_0} Q_{encl}. \tag{2.82}$$

Equation (2.82) is Gauss' law in integral form. We get the differential form by applying the divergence theorem, where dV is an infinitesimal volume element:

$$\oint_S \mathbf{E} \cdot d\mathbf{a} = \int_V (\nabla \cdot \mathbf{E})\, dV. \tag{2.83}$$

The charge enclosed can be rewritten in terms of the charge density ρ^*,

$$Q_{encl} = \int_V \rho^* dV, \tag{2.84}$$

and we get:

$$\int_V \nabla \cdot \mathbf{E} dV = \int_V \frac{\rho^*}{\epsilon_0} dV. \tag{2.85}$$

Again, as this is valid for any volume, the integrands have to be the same and Gauss law simplifies to:

$$\nabla \cdot \mathbf{E} = \frac{\rho^*}{\epsilon_0}, \tag{2.86}$$

which can be seen as Coulomb's law plus the law of superposition (Griffiths, 1999, p. 232).

The second of Maxwell's equations is based on Gauss' law of magnetism, as seen in Eqn. (2.76). It is an expression for the non-existence of magnetic monopoles or free magnetic poles or another analogue to electric charges. The flux integrated over any finite closed surface is zero, because as many lines that enter the volume enclosed by the surface also leave it (Priest, 2014; Grant and Phillips, 1999, p. 75, 191). Hence, the magnetic field lines are continuous, and form either closed loops or extend out to infinity (Griffiths, 1999, p. 232).

Faraday's law, as seen in Eqn. (2.77), states that time-varying magnetic fields give rise to electric fields (Priest, 2014, p.75). Based on a series of experiments Faraday conducted, he found, in 1831, that if there is a loop of wire and a magnetic field, we get a current flowing by holding the loop still and changing the magnetic field. The force acting on a charge at rest, cannot, however, be a magnetic force and Faraday inferred that a changing magnetic field, induces an electric field (Griffiths, 1999, p. 302).

The induced electromotive force (emf) V is related to the induced electric field $\mathbf{E}$, which is caused by the changing magnetic field in the following way (Grant and Phillips, 1999, pp. 218):

$$V = \oint \mathbf{E} \cdot d\mathbf{l}. \tag{2.87}$$

Furthermore, with the resistance of a material R, the current I can be determined with:

$$I = \frac{V}{R} = \frac{1}{R} \oint \mathbf{E} \cdot d\mathbf{l}. \tag{2.88}$$

From experiments, it is known that the current I is proportional to the rate of change with time of the total magnetic flux Φ through the circuit and the magnitude of the emf is therefore proportional to $\frac{d\Phi}{dt}$:

$$\left| \oint \mathbf{E} \cdot d\mathbf{l} \right| \propto \frac{d\Phi}{dt}. \tag{2.89}$$

Lenz' law gives information about the direction of this current induced and the emf. The direction of the current is such that it produces a magnetic flux tending to oppose the original change of flux:

$$\oint \mathbf{E} \cdot d\mathbf{l} \propto -\frac{d\Phi}{dt}. \tag{2.90}$$

The magnetic flux through a coil is given by:

$$\Phi = \int_S \mathbf{B} \cdot d\mathbf{S}. \tag{2.91}$$

In SI units, the proportionality constant is unity and from:

$$\oint \mathbf{E} \cdot d\mathbf{l} = -\frac{d\Phi}{dt}, \tag{2.92}$$

we obtain

$$\oint \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_S \mathbf{B} \cdot d\mathbf{S}. \tag{2.93}$$

According to Eqn. (2.93), the line integral of the induced electric field around a loop can be related to the time-rate-of-change of the magnetic flux through the surface enclosed by this loop. If we now use Stokes' theorem and express the line integral of the electric field by the integral over the surface S of the curl of the vector E,

$$\oint_C \mathbf{E} \cdot d\mathbf{l} = \int_S \nabla \times \mathbf{E} \cdot d\mathbf{S} \tag{2.94}$$

we obtain

$$\int_S \nabla \times \mathbf{E} \cdot d\mathbf{S} = -\frac{d}{dt} \int_S \mathbf{B} \cdot d\mathbf{S}. \tag{2.95}$$

The Leibniz rule for the derivative of an integral with constant integration limits allows us to write (2.95) as

$$\int_S \nabla \times \mathbf{E} \cdot d\mathbf{S} = \int_S -\frac{\partial \mathbf{B}}{\partial t} \cdot d\mathbf{S}, \tag{2.96}$$

and we arrive at the differential form of Faraday's law:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}. \tag{2.97}$$

Maxwell's fourth equation, based on Ampère's law, as seen in Eqn. (2.78), describes how currents and changing electric fields produce magnetic fields (Priest, 2014, p.75). This equation is a generalisation of Ampère's law, which puts the line integral of a magnetic field around a closed loop in relation to the current that passes through the loop (Grant and Phillips, 1999, p. 135):

$$\oint_C \mathbf{B} \cdot d\mathbf{l} = \mu_0 j, \tag{2.98}$$

where μ_0 is the permeability of free space, a constant that helps to define the unit ampère (Griffiths, 1999, p. 216). However, Ampère's law is only valid if currents are stationary and continuous (Tipler and Mosca, 2009, p. 1064). Maxwell found an expression to make Ampère's law valid for currents that are not continuous by adding a displacement current

I_d (Tipler and Mosca, 2009, p. 1166):

$$I_d = \epsilon_0 \frac{d\Phi_E}{dt}. \tag{2.99}$$

We get to the general form of Ampère's law:

$$\oint_C \mathbf{B} \cdot d\mathbf{l} = \mu_0(I + I_d) = \mu_0 I + \mu_0 \epsilon_0 \frac{d\Phi_E}{dt}. \tag{2.100}$$

With the propagation speed for electromagnetic waves of:

$$c = \frac{1}{\sqrt{\mu_0 \epsilon_0}}, \tag{2.101}$$

we get:

$$\oint_C \mathbf{B} \cdot d\mathbf{l} = \mu_0 I + \frac{1}{c^2} \frac{d\Phi_E}{dt}. \tag{2.102}$$

The differential form of Ampère's law can be obtained with an expression for the total current (as a surface integral over the current density j) and Stoke's theorem (Grant and Phillips, 1999, p. 148):

$$I = \int_S \mathbf{j} \cdot d\mathbf{S}. \tag{2.103}$$

Without the displacement current we can then rewrite Ampère's law in the following manner:

$$\oint_C \mathbf{B} \cdot d\mathbf{l} = \mu_0 \int_S \mathbf{j} \cdot d\mathbf{S}. \tag{2.104}$$

According to Stokes' theorem, a line integral can be expressed as a surface integral:

$$\oint_C \mathbf{B} \cdot d\mathbf{l} \equiv \int_S \nabla \times \mathbf{B} \cdot d\mathbf{S}, \tag{2.105}$$

and Ampère's law can be written as:

$$\int_S \nabla \times \mathbf{B} \cdot d\mathbf{S} = \mu_0 \int_S \mathbf{j} \cdot d\mathbf{S}. \tag{2.106}$$

As this must hold for any surfaces, the integrands must be the same. By adding the

displacement current, we get the differential form:

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \frac{1}{c^2} \frac{d\Phi_E}{dt}. \tag{2.107}$$

Ampère's law describes what causes a magnetic field, and the induction equation - how it evolves. The induction equation is derived starting at Faraday's equation:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \quad \text{or} \quad \frac{\partial \mathbf{B}}{\partial t} = -(\nabla \times \mathbf{E}). \tag{2.108}$$

With Ohm's law, we can then get an expression for the electric field. Ohm's law, in general, relates the current density $\mathbf{j}$ to a force $\mathbf{f}$ per unit charge, where the proportionality factor σ is the conductivity of the medium (Griffiths, 1999, p. 285):

$$\mathbf{j} = \sigma \mathbf{f}. \tag{2.109}$$

In our case, the force that drives the charges is electromagnetic force, and we get to the special case of Ohm's law:

$$\mathbf{j} = \sigma(\mathbf{E} + \mathbf{v} \times \mathbf{B}). \tag{2.110}$$

If the velocity of the charges is sufficiently small, it reduces to:

$$\mathbf{j} = \sigma \mathbf{E}. \tag{2.111}$$

For the electric field, we otherwise obtain:

$$\mathbf{E} = \frac{\mathbf{j}}{\sigma} - \mathbf{v} \times \mathbf{B}. \tag{2.112}$$

Substituting $\mathbf{E}$ in Eqn. (2.108), we get:

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times (-\mathbf{v} \times \mathbf{B} + \frac{\mathbf{j}}{\sigma}). \tag{2.113}$$

From Ampère's law we can get the current density $\mathbf{j} = \nabla \times \mathbf{B}/\mu_0$. Therefore, $\mathbf{j}/\sigma = \nabla \times$

$\mathbf{B}/\mu_0\sigma$. With the constant $\eta = 1/\mu_0\sigma$, where η is the diffusivity, we yield

$$\begin{aligned}\frac{\partial \mathbf{B}}{\partial t} &= \nabla \times (\mathbf{v} \times \mathbf{B}) - \nabla \times (\eta \nabla \times \mathbf{B}) \\ &= \nabla \times (\mathbf{v} \times \mathbf{B}) - \eta \nabla \times (\nabla \times \mathbf{B}).\end{aligned} \tag{2.114}$$

In general, η is not constant, which means the simplification in Eqn. (2.114) cannot be made. For η as a function of space, the extra term $-\nabla\eta \times (\nabla \times \mathbf{B})$ has to be added. With the vector calculus identity $\nabla \times (\nabla \times \mathbf{B}) = \nabla(\nabla \cdot \mathbf{B}) - \nabla^2\mathbf{B}$ and knowing that $\mathbf{B}$ is a solenoidal vector field, hence $\nabla \cdot \mathbf{B} = 0$, we get the induction equation:

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{B}) + \eta \nabla^2 \mathbf{B}, \tag{2.115}$$

where the first term on the RHS describes the advection of the magnetic field with the plasma and the second term describes the diffusion of the magnetic field through the plasma (Wilmot-Smith, Priest, and Hornig, 2005). With another vector identity ($\nabla \times (\mathbf{v} \times \mathbf{B}) = \nabla \cdot (\mathbf{B}\mathbf{v} - \mathbf{v}\mathbf{B})$) and for ideal MHD (magnetic diffusivity $\eta = 0$), we obtain the induction equation:

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{v}\mathbf{B} - \mathbf{B}\mathbf{v}) = 0. \tag{2.116}$$

Before merging the hydrodynamic equations with the electromagnetic equations to obtain the MHD equations, we have to make the Maxwell equations Galilean invariant (Schnack, 2009, p. 36). This means, we consider low velocities, $v/c << 1$. With the Ohm's law, which couples the dynamics of the electromagnetic field and the fluid:

$$\mathbf{E}' = \eta \mathbf{j}, \tag{2.117}$$

where E' is the electric field in the moving frame, which, according to the theory of relativity, is

$$\mathbf{E}' = \frac{\mathbf{E} + \mathbf{v} \times \mathbf{B}}{\sqrt{1 - \frac{v^2}{c^2}}}, \tag{2.118}$$

we arrive at an expression for Ohm's law in MHD form:

$$\mathbf{E} + \mathbf{v} \times \mathbf{B} = \eta \mathbf{j}. \tag{2.119}$$

This is often referred to as resistive Ohm's law. If we assume infinite conductivity, η becomes zero and the ideal MHD Ohm's law is obtained (Schnack, 2009, p. 36).

In Ampère's law (Eqn. (2.78)) the displacement current can be neglected for low frequencies, where the electric field is negligibly small, and Eqn. (2.78) then becomes (Schnack, 2009, p. 36):

$$\mu_0 \mathbf{j} = \nabla \times \mathbf{B}. \tag{2.120}$$

The momentum equation of our full MHD equations represents balance between acceleration, pressure gradient and the Lorentz force. It can take on different forms and in the following we want to briefly outline how they are related. In the momentum equation, we have body forces acting on the fluid element. These body forces can be gravitational or electromagnetic forces, like the Lorentz force, which can be expressed by

$$F_L = \mathbf{j} \times \mathbf{B}. \tag{2.121}$$

The Lorentz force acts perpendicular to the magnetic field, therefore, any acceleration along the magnetic field lines is due to a pressure gradient or gravity (Goedbloed and Poedts, 2004, p. 70). We use Ampère's law

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j} \quad \text{or} \quad \mathbf{j} = \frac{\nabla \times \mathbf{B}}{\mu_0} \tag{2.122}$$

in the following vector identity (where μ_0 is a constant):

$$\begin{aligned}
\frac{1}{2}\nabla(\mathbf{B}\cdot\mathbf{B}) &= (\mathbf{B}\cdot\nabla)\mathbf{B} + \mathbf{B}\times(\nabla\times\mathbf{B}) \qquad (2.123)\\
\frac{\nabla B^2}{2} &= (\mathbf{B}\cdot\nabla)\mathbf{B} + \mathbf{B}\times(\mu_0\mathbf{j})\\
\frac{\nabla B^2}{2} &= (\mathbf{B}\cdot\nabla)\mathbf{B} + \mu_0(\mathbf{B}\times\mathbf{j})\\
\frac{\nabla B^2}{2\mu_0} &= \frac{(\mathbf{B}\cdot\nabla)\mathbf{B}}{\mu_0} + (\mathbf{B}\times\mathbf{j}).
\end{aligned}$$

Now, by rearranging, we obtain:

$$\mathbf{j}\times\mathbf{B} = \frac{(\mathbf{B}\cdot\nabla)\mathbf{B}}{\mu_0} - \frac{\nabla\mathbf{B}^2}{2\mu_0}, \tag{2.124}$$

where the first term in the RHS is the magnetic tension force and the second term is the magnetic pressure force.

The terms of the energy equation are derived from the net flux of heat into a fluid element as well as the rate of work done on the fluid element due to body and contact forces. In an electromagnetic field this can be thermal conduction, viscous heating as well as Joule dissipation. The total energy is the sum of kinetic, thermal, and magnetic energy. The flux term of the energy equation in Hillier, Takasao, and Nakamura (2016) is $\nabla[\mathbf{v}(e+P)+\frac{c}{4\pi}\mathbf{E}\times\mathbf{B}]$ and relates to our energy equation as follows (applying Ohm's law and the triple vector product):

$$\begin{aligned}
\mathbf{E} &= -\frac{1}{c}(\mathbf{v}\times\mathbf{B}) \qquad (2.125)\\
\frac{c}{4\pi}\mathbf{E}\times\mathbf{B} &= \frac{c}{4\pi}\left[-\frac{1}{c}(\mathbf{v}\times\mathbf{B})\times\mathbf{B}\right]\\
&= -\frac{1}{4\pi}\left[-(\mathbf{B}\cdot\mathbf{B})\mathbf{v}+(\mathbf{B}\cdot\mathbf{v})\mathbf{B}\right]\\
&= \frac{1}{4\pi}\left[(\mathbf{B}\cdot\mathbf{B})\mathbf{v}-\mathbf{B}(\mathbf{B}\cdot\mathbf{v})\right]
\end{aligned}$$

The full MHD equations write as follows:

$$\frac{\partial}{\partial t}\rho + \nabla\cdot(\rho\mathbf{v}) = 0 \tag{2.126}$$

$$\frac{\partial}{\partial t}\rho\mathbf{v} + \nabla\cdot(\rho\mathbf{v}\mathbf{v} + p\mathbf{I} - \mathbf{B}\mathbf{B}) = 0 \tag{2.127}$$

$$\frac{\partial}{\partial t}e + \nabla\cdot\left[(e + P + B^2/8\pi)\mathbf{v} - \mathbf{B}/4\pi(\mathbf{v}\cdot\mathbf{B})\right] = 0 \tag{2.128}$$

$$\frac{\partial}{\partial t}\mathbf{B} + \nabla\cdot(\mathbf{v}\mathbf{B} - \mathbf{B}\mathbf{v}) = 0. \tag{2.129}$$

2.3 Two-Fluid Source Terms

The crucial aspect of a two-fluid code is the coupling of the fluids due to collisions, expressed mathematically in the source terms of the system of equations. If they were not coupled, they would simply behave as separate fluids sharing the same space. However, the fluids' interaction leads to significant effects, as described in Sec. 1.2. Those effects are captured with our two-fluid MHD code and the source term is presented on the following.

For two-fluid MHD simulations, the full solution vector for our two-fluid code has the following form, where the subscript p denotes the ionised fluid and n the neutral hydrogen fluid:

$$\mathbf{U} = [\rho_p, \rho_p v_{xp}, \rho_p v_{yp}, \epsilon_p, B_x, B_y, \rho_n, \rho_n v_{xn}, \rho_n v_{yn}, \epsilon_n]^T, \tag{2.130}$$

and the flux vectors are:

$$\mathbf{F} = \begin{bmatrix} \rho_p v_{xp} \\ \rho_p v_{xp}^2 + P_p + B^2/8\pi - B_x^2/4\pi \\ \rho_p v_{xp} v_{yp} - B_x B_y/4\pi \\ (\epsilon_p + P_p + B^2/8\pi) v_{xp} - B_x(\mathbf{v}_p \cdot \mathbf{B})/4\pi \\ 0 \\ v_{xp} B_y - v_{yp} B_x \\ \rho_n v_{xn} \\ \rho_n v_{xn}^2 + P_n \\ \rho_n v_{xn} v_{yn} \\ v_{xn}(\epsilon_n + P_n) \end{bmatrix},$$

$$\mathbf{G} = \begin{bmatrix} \rho_p v_{yp} \\ \rho_p v_{xp} v_{yp} - B_x B_y/4\pi \\ \rho_p v_{yp}^2 + P_p + B^2/8\pi - B_y^2/4\pi \\ (\epsilon_p + P_p + B^2/8\pi) v_{yp} - B_y(\mathbf{v}_p \cdot \mathbf{B})/4\pi \\ v_{xp} B_y - v_{yp} B_x \\ 0 \\ \rho_n v_{yn} \\ \rho_n v_{xn} v_{yn} \\ \rho_n v_{yn}^2 + P_n \\ v_{yn}(\epsilon_n + P_n) \end{bmatrix}. \tag{2.131}$$

The system of equations is closed with the following equations for the total energy:

$$\epsilon_p = \frac{\rho v_p^2}{2} + \frac{P_p}{(\gamma - 1)} + \frac{B^2}{8\pi} \tag{2.132}$$

for the ionised fluid, and

$$\epsilon_n = \frac{\rho v_n^2}{2} + \frac{P_n}{(\gamma - 1)} \tag{2.133}$$

for the neutral fluid. Each of these represents an ideal gas equation. The first term in these total energy equations expresses the kinetic energy, the second term the thermal energy and the third term in Eqn. (2.132) is the magnetic energy term.

As has been mentioned before, the crucial aspect of multi-fluid simulations is the coupling of the fluids. The two sets of equations for each, the neutral and the ionised fluid, are coupled through collision, ionisation, and recombination terms, mathematically implemented in the source terms, expressed for the entire system of equations in Eqn. (2.134).

$$\mathbf{S} = \begin{bmatrix} -\gamma_{rec}\rho_p + \gamma_{ion}\rho_n \\ \rho_p g_x + \alpha_c \rho_n \rho_p (v_{xn} - v_{xp}) - \gamma_{rec}\rho_p v_{xp} + \gamma_{ion}\rho_n v_{xn} \\ \rho_p g_y + \alpha_c \rho_n \rho_p (v_{yn} - v_{yp}) - \gamma_{rec}\rho_p v_{yp} + \gamma_{ion}\rho_n v_{yn} \\ \rho_p g_x v_{x_p} + \alpha_c \rho_n \rho_p [(v_n^2 - v_p^2)/2 + 3R_g(T_n - T_p)] - (\gamma_{rec}\rho_p v_p^2)/2 + (\gamma_{ion}\rho_n v_n^2)/2 \\ 0 \\ 0 \\ \gamma_{rec}\rho_p - \gamma_{ion}\rho_n \\ \rho_n g_x - \alpha_c \rho_n \rho_p (v_{xn} - v_{xp}) + \gamma_{rec}\rho_p v_{xp} - \gamma_{ion}\rho_n v_{xn} \\ \rho_n g_y - \alpha_c \rho_n \rho_p (v_{yn} - v_{yp}) + \gamma_{rec}\rho_p v_{yp} - \gamma_{ion}\rho_n v_{yn} \\ \rho_n g_y v_{y_n} - \alpha_c \rho_n \rho_p [(v_n^2 - v_p^2)/2 + 3R_g(T_n - T_p)] + (\gamma_{rec}\rho_p v_p^2)/2 - (\gamma_{ion}\rho_n v_n^2)/2 \end{bmatrix}. \tag{2.134}$$

γ_{ion} and γ_{rec} are the ionisation and recombination rates, respectively. The temperature is calculated with the ideal gas law and in the non-dimensionalised form it is given by $T_n = P_n\gamma/\rho_n$ for the neutral fluid and by $T_p = P_p\gamma/2\rho_p$ for the ionised fluid. γ is the adiabatic index and R_g the gas constant.

The interaction between particles with each other or existing force fields is often also referred to as collisions and, hence, the terms expressing the particle interactions are called collision terms, also known as collision integrals. These collisions can involve transfer of mass, momentum and energy and their study is part of transport theory (Boyd and Sanderson, 2003, p. 296). Collisions are important for the establishment of thermodynamic equilibrium and can be divided into elastic and inelastic collisions. Inelastic collisions change the internal state of a fluid by various processes such as ionisation, recombination, excitation and variuos other plasmachemical reactions (Rozhansky and Tsendin, 2001, p. 19). The collision terms are a very complex matter and one of the most challenging topics in plasma physics, as they require, for example, the evaluation of the kind of collision,

the scattering angle, and various cross sections. The plasma books by Bittencourt (2004) or Boyd and Sanderson (2003) have been proven to be comprehensive and well written sources of literature on that topic. Therefore, only a brief insight into collision terms and their form is given and afterwards each collision term used in our equations will be briefly described in their meaning.

The collision integral for a species α, is the sum over all collisions between the particles of α with the the particles of the other species, say β (Rozhansky and Tsendin, 2001, p. 18). For elastic, two-body collisions, for instance, Boltzmann derived a collision integral, which is written as (Cap, 1994):

$$\left(\frac{\partial f_\alpha}{\partial t}\right)_{coll} = \int (\mathbf{v}_\alpha - \mathbf{v}_\beta)[f(\mathbf{x}_\alpha, \mathbf{v}^*{}_\alpha, t) f(\mathbf{x}_\beta, \mathbf{v}^*{}_\beta, t) - f(\mathbf{x}_\alpha, \mathbf{v}_\alpha, t) f(\mathbf{x}_\beta, \mathbf{v}_\beta, t)]\sigma_c d\mathbf{v}_\beta, \quad (2.135)$$

where the subscript α and β denote different kind of species and σ_c is the cross section. $\mathbf{v}^*$ denote the velocities before the collision, which are scattered to the velocity $\mathbf{v}$. At equilibrium each collision term is equal to zero, which is satisfied with the Maxwell distribution. Landau included Coulomb collisions and extended the Boltzmann collision integral (Cap, 1994).

As can be seen in Eqn. (2.4), for the evolution of the macroscopic variables (i.e., for the analysis of relatively slow, large-scale transport processes), the moment for each collision term has to be taken. The collision terms of the macroscopic quantities (mass density, momentum and energy) are:

$$\int \left(\frac{\partial f_\alpha}{\partial t}\right)_{coll} d\mathbf{v} = S_\alpha \quad (2.136)$$

$$m_\alpha \int \left(\mathbf{v}\frac{\partial f_\alpha}{\partial t}\right)_{coll} d\mathbf{v} = R_\alpha \quad (2.137)$$

$$\frac{m_\alpha}{2} \int \left(v^2\frac{\partial f_\alpha}{\partial t}\right)_{coll} d\mathbf{v} = Q_\alpha \quad (2.138)$$

where S_α, R_α, and Q_α stand for the source terms of the continuity, momentum, and energy equation, respectively. It follows that for inelastic collisions, the collision terms have to be the inverse of each other, so that conservation laws apply. In our neutral-ion plasma,

this would mean:

$$S_p = -S_n, \quad R_p = -R_n, \quad Q_p = -Q_n. \tag{2.139}$$

This implies that every collision term in the source term of the ionised fluid has the opposite sign of the same collision term in the neutral fluid equation. The shape of Eqn. (2.139) are determined by some appropriate physical approximations, which can be obtained from classical transport theory (Boyd and Sanderson, 2003, p. 487). Their derivation can be found in Braginskii (1965), Boyd and Sanderson (2003), and Rozhansky and Tsendin (2001).

Source terms represent either forces acting on a fluid element or the transfer of a quantity onto the other fluid. In our continuity equation, the source term includes terms expressing the transfer of mass (per volume) due to ionisation and recombination (Draine, Roberge, and Dalgarno, 1983). If no particles are produced or annihilated, the source term is zero (Braginskii, 1965). In the following, the source term of the ionised fluid (top row) and the neutral fluid (bottom row) shows that the mass density that is added to one fluid, will be subtracted from the other fluid.

$$\mathbf{S}_{density} = \begin{bmatrix} -\gamma_{rec}\rho_p + \gamma_{ion}\rho_n \\ \gamma_{rec}\rho_p - \gamma_{ion}\rho_n \end{bmatrix}. \tag{2.140}$$

As can be seen, if γ_{ion} and γ_{rec} are set to zero, no mass density would be exchanged at all. By taking the zeroth moment of a Maxwell distribution function, Meier and Shumlak (2012) derive concisely that the impact of ionisation on the mass density of the ionised fluid equals the inverse of the same for the neutral fluid, i.e. $\gamma^p_{ion} = -\gamma^n_{ion}$, where the superscript here indicates the fluid which is affected by the collision term. The same applies for the recombination rate, where $-\gamma^p_{rec} = \gamma^n_{rec}$. A good qualitative consideration of the collision terms or transport coefficients is also given by Rozhansky and Tsendin (2001). In the momentum and energy equations, body and contact forces, like gravity $\mathbf{g}$,

come into play in the source term.

$$\mathbf{S}_{momentum} = \begin{bmatrix} \rho_p g_x + \alpha_c \rho_n \rho_p (v_{xn} - v_{xp}) - \gamma_{rec} \rho_p v_{xp} + \gamma_{ion} \rho_n v_{xn} \\ \rho_p g_y + \alpha_c \rho_n \rho_p (v_{yn} - v_{yp}) - \gamma_{rec} \rho_p v_{yp} + \gamma_{ion} \rho_n v_{yn} \\ \rho_n g_x - \alpha_c \rho_n \rho_p (v_{xn} - v_{xp}) + \gamma_{rec} \rho_p v_{xp} - \gamma_{ion} \rho_n v_{xn} \\ \rho_n g_y - \alpha_c \rho_n \rho_p (v_{yn} - v_{yp}) + \gamma_{rec} \rho_p v_{yp} - \gamma_{ion} \rho_n v_{yn} \end{bmatrix}. \tag{2.141}$$

The first two rows of Eqn. (2.141) are the ionised fluid's momentum source terms and the third and fourth row are the neutral fluid's momentum source terms, in $x-$ and $y-$direction. As gravity can act at an angle, depending on the orientation of the computational domain, it has an x and a y component. However, in this work, gravity is neglected and set to zero. The source term for the momentum equation describes how much momentum is added to each fluid; and the source term of the energy equations describes how much energy (kinetic, thermal or magnetic) is added to each of the fluids (Draine, Roberge, and Dalgarno, 1983). Furthermore, the numerical setup is such that the two fluids basically share the same position in space and move through thermal motions, i.e. due to their thermal energy. Momentum transfer are the result of the collisions between the two fluids, described by the collision rate α_c:

$$\alpha_c = \alpha_c(T_0) \sqrt{\frac{T_n + T_p}{2T_0}}. \tag{2.142}$$

This collision rate, like in Hillier, Takasao, and Nakamura (2016), covers the elastic collisions between the ionised fluid and the neutral fluid and shows a dependence on temperatures of both fluids. $\alpha_c(T_0)$ expresses the collision rate at a reference temperature, the ambient temperature T_0, which is the surrounding temperature at the beginning of the simulation. The friction force is the second term in Eqn. (2.141), which due to the loss of the particles' relative velocity after a collision leads to a redirected motion (Rozhansky and Tsendin, 2001, p. 31). The conversion of the energy of direct motion into heat by friction force is called Joule heating (Rozhansky and Tsendin, 2001, p. 37). The friction force term includes the velocities of both fluids in relation to each other, the drift velocity, and is proportional to it. The resulting velocity is not isotropic (as for thermal motions), but has a certain direction. This direction is the direction $\mathbf{v}_n - \mathbf{v}_p$, i.e. the direction of the drift velocity, and implies the impact of one fluid on the other. On the one hand, the

friction force is determined by and proportional to the drift velocity, but it also causes a drift, which is in the direction of the electric field (Rozhansky and Tsendin, 2001, p. 41). This means, the higher the drift velocity, i.e. the greater the difference in velocity of both fluids, the greater the friction force. It follows that the fluid moves in the direction of the electric field with a velocity which is proportional to the collision frequency. The thermal energy term related to the thermal exchange due to ionisation and recombination ($\frac{1}{\gamma-1}\frac{k_B}{m_n}(\gamma_{rec}\rho_p T_p - \gamma_{ion}\rho_n T_n)$, compare Leake *et al.* (2012); Popescu Braileanu *et al.* (2019)), has not been included to this set of equations, as a direct comparison to Hillier, Takasao, and Nakamura (2016) was the aim, therefore, the exact same equations therein were solved for.

The third and fourth terms are related to ionisation and recombination, the inelastic interactions between the fluids. Those two terms are related to the velocities of each fluid in the respective direction, but not to the drift velocity. As can be seen, recombination adds to the momentum of the neutral fluid (at the same time it takes away momentum from the ionised fluid) and ionisation adds to the momentum of the ionised fluid (and takes away momentum from the neutral fluid). Or in other words, recombination processes, which are expressed mathematically as the γ_{rec}-terms and are added to the source term of the neutral fluid, increase the neutral fluid's momentum and ionisation processes increase the ionised fluid's momentum.

In the energy source terms, Eqn. (2.143), the top row is the ionised fluid's energy source term and second row is the neutral fluid's energy source term.

$$\mathbf{S}_{energy} = \begin{bmatrix} \rho_p g_x v_{x_p} + \alpha_c \rho_n \rho_p [(v_n^2 - v_p^2)/2 + 3R_g(T_n - T_p)] - (\gamma_{rec}\rho_p v_p^2)/2 + (\gamma_{ion}\rho_n v_n^2)/2 \\ \rho_n g_y v_{y_n} - \alpha_c \rho_n \rho_p [(v_n^2 - v_p^2)/2 + 3R_g(T_n - T_p)] + (\gamma_{rec}\rho_p v_p^2)/2 - (\gamma_{ion}\rho_n v_n^2)/2 \end{bmatrix}. \tag{2.143}$$

Here, the second term is related to the energy exchange between the fluids due to elastic collisions, where one part expresses the kinetic energy exchange and the other part (including the temperature), represents the thermal energy exchange. The last two terms are related to inelastic collisions, namely ionisation and recombination processes. As can be seen from the last term of the ionised fluid's energy source term, through ionisation the ionised fluid gains energy and through recombination, the neutral fluid gains energy (third term in the neutral fluid's energy source term). Therefore, each fluid's energy level

depends on the size of each term, γ_{ion} and γ_{rec}.

Ionisation and recombination are very complex processes, as mentioned in the Sec. 1.2. We want to investigate what happens when ionisation and recombination rates are implemented and activated. In order to emphasize the difference with and without ionisation and recombination, the rates used maximise the effect of ionisation and recombination in the two chosen cases that will be presented. Following ionisation and recombination rates are used:

$$\gamma_{ion} = 0.25[\tanh(T_n - T_{max}/2) + 1], \tag{2.144}$$

$$\gamma_{rec} = 0.5[1 - \gamma_{ion}]. \tag{2.145}$$

The form of γ_{ion} and γ_{rec} is not derived from actual physical settings, but, again, chosen to maximise the effects of these rates in the parameter regime studied, i.e. the parameter regime of the slow-mode shock and the Orszag-Tang vortex.
A physical derivation and description can be found in Braginskii (1965), where transport equations for fully ionised plasmas and, to some extent, partially ionised plasmas are derived and explained. Meier and Shumlak (2012) focus more on partially ionised plasma, or the interaction of ionised-neutral fluids and present their model based on kinetic theory. A comparison with formulations derived from physical principles is presented in the following. The ionisation and recombination rates used here, according to Eqns. (2.144) and (2.145), are compared to the ionisation and recombination rates used in Popescu Braileanu *et al.* (2019) (and also used in Leake *et al.* (2012)):

$$\alpha_{ion} = n_e A \frac{1}{X + \Phi_{ion}/T_e} \left(\frac{\Phi_{ion}}{T_e}\right)^K e^{(-\Phi_i on/T_e)}, \tag{2.146}$$

$$\alpha_{rec} = \frac{n_e}{\sqrt{T_e}} 2.6 \times 10^{-19}, \tag{2.147}$$

with $\Phi_{ion} = 13.6eV$, $A = 2.91 \times 10^{-14}$, $K = 0.39$, $X = 0.232$, and the temperature T_e is given in eV. In Fig. 2.1, the rates given by Eqns. (2.146) and (2.147) are compared to the ionisation and recombination rates applied in this work, see Eqns. (2.144) and (2.145). Scaling is applied to compare their behaviour as a function of temperature. This scaling brings our temperature regime to the regime used in Popescu Braileanu *et al.* (2019) (with

$T_{scaled} = 1500T + 13330$) and their ionisation and recombination rates are scaled down to match our regime (with $n_{e\,scaled} = 0.0025 n_e$).

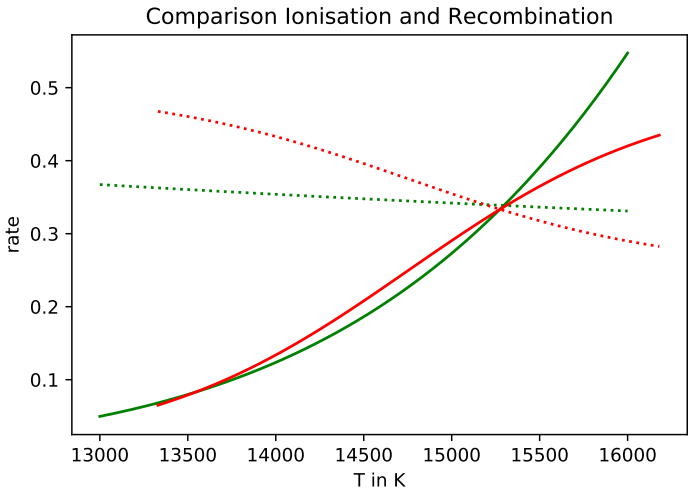

Figure 2.1: The ionisation rate (solid line) and recombination rate (dotted line) in Popescu Braileanu *et al.* (2019) *green* and in our model *red*

.

As can be seen, our ionisation and recombination rate (red) and the physical ionisation and recombination rate (green), behave very similarly in the regime where ionisation and recombination both play a role, i.e. where temperatures are not so low that the fluid is completely neutral, i.e. ionisation is unlikely to happen or where temperatures are so high that everything is ionised.
The full two-fluid equations are:

$$
\begin{aligned}
\frac{\partial}{\partial t}\rho_p + \nabla\cdot(\rho_p \mathbf{v}_p) &= -\gamma_{rec}\rho_p + \gamma_{ion}\rho_n \\
\frac{\partial}{\partial t}\rho_p \mathbf{v}_p + \nabla\cdot(\rho_p \mathbf{v}_p \mathbf{v}_p + p_p \mathbf{I} - \mathbf{B}\mathbf{B}) &= \rho_p \mathbf{g} + \alpha_c \rho_n \rho_p (\mathbf{v}_n - \mathbf{v}_p) - \gamma_{rec}\rho_p \mathbf{v}_p + \gamma_{ion}\rho_n \mathbf{v}_n \\
\frac{\partial}{\partial t} e_p + \nabla\cdot\left[\left(e_p + P_p + B^2/8\pi\right)\mathbf{v}_p - \mathbf{B}/4\pi(\mathbf{v}_p\cdot\mathbf{B})\right] &= \rho_p \mathbf{g}\mathbf{v_p} + \alpha_c \rho_n \rho_p \left[\left(\mathbf{v}_n^2 - \mathbf{v}_p^2\right)/2 + 3R_g(T_n - T_p)\right] \\
&\quad - \left(\gamma_{rec}\rho_p \mathbf{v}_p^2\right)/2 + \left(\gamma_{ion}\rho_n \mathbf{v}_n^2\right)/2 \\
\frac{\partial}{\partial t}\mathbf{B} + \nabla\cdot(\mathbf{v}_p\mathbf{B} - \mathbf{B}\mathbf{v}_p) &= 0 \\
\frac{\partial}{\partial t}\rho_n + \nabla\cdot(\rho_n \mathbf{v}_n) &= \gamma_{rec}\rho_p - \gamma_{ion}\rho_n \\
\frac{\partial}{\partial t}\rho_n \mathbf{v}_n + \nabla\cdot(\rho \mathbf{v}_n \mathbf{v}_n + p_n \mathbf{I}) &= -\rho_p \mathbf{g} - \alpha_c \rho_n \rho_p (\mathbf{v}_n - \mathbf{v}_p) + \gamma_{rec}\rho_p \mathbf{v}_p - \gamma_{ion}\rho_n \mathbf{v}_n \\
\frac{\partial}{\partial t} e_n + \nabla\cdot(e_n + p_n)\mathbf{v}_n &= \rho_p \mathbf{g}\mathbf{v_p} + \alpha_c \rho_n \rho_p \left[\left(\mathbf{v}_n^2 - \mathbf{v}_p^2\right)/2 + 3R_g(T_n - T_p)\right] \\
&\quad - \left(\gamma_{rec}\rho_p \mathbf{v}_p^2\right)/2 + \left(\gamma_{ion}\rho_n \mathbf{v}_n^2\right)/2
\end{aligned}
$$

Chapter 3

Numerical Methods – an Overview

To simulate or computationally model any physical phenomenon or processes, one needs to

1. define the problem
2. model it mathematically and then
3. computationally simulate it.

After the definition of the problem, the governing equations and the initial and boundary conditions have to be found (Peiro and Sherwin, 2005). Often conservation laws and their mathematical representation in the form of PDEs are involved and even though they can be solved numerically, obtaining their solution can be a very complex task.

3.1 Partial Differential Equations (PDEs)

PDEs express a function of several variables and how fast the function changes, or its rate of change. They include partial derivatives, where the function (the dependent variable) depends on at least two independent variables. The coefficient of the partial derivative determines the order of the PDE (Shyy, 2006, p. 27).

There are three operators that are each a typical feature of the three general classes of partial differential equations (Shyy, 2006, p. 27):

1. Laplace operator, $\nabla^2 = \frac{\partial^2}{\partial X_1^2} + ... + \frac{\partial^2}{\partial X_n^2}$, which typifies elliptic equations (e.g. in potential flow problems in fluid mechanics)

2. Diffusion operator, $\frac{\partial}{\partial t} - \nabla^2$, which typifies parabolic equations (e.g. in diffusion dominated situations, as in heat conduction)

3. D'Alembert operator, $\Box = \frac{\partial^2}{\partial t^2} - \nabla^2$, which typifies hyperbolic equations and is applied the most in wave transmissions

Physical behaviour can be divided into the categories of equilibrium and marching problems (Versteeg and Malalasekera, 2007, p. 26). Elliptic equations govern steady state situations. Marching, or propagating problems, are governed by parabolic and hyperbolic equations. Distinctive for an elliptic equation or problem is that if the interior of the solution is disturbed or a perturbation into the boundary condition is introduced, it instantaneously changes the solution everywhere (Peiro and Sherwin, 2005). For example, a local change in temperature would lead to a global change in temperature (Versteeg and Malalasekera, 2007, p. 27). A typical example for an elliptic equation is the Laplace equation, which describes irrotational flow of an incompressible fluid and steady state heat transfer and reads in two dimensions as:

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = 0. \tag{3.1}$$

Parabolic equations describe time-dependent problems with diffusion, like unsteady viscous flows or heat conduction. The heat or diffusion equation is a typical example of a parabolic equation (Versteeg and Malalasekera, 2007, p. 28):

$$\frac{\partial \Phi}{\partial t} = \alpha \frac{\partial^2 \Phi}{\partial x^2}. \tag{3.2}$$

Here, the solutions move forward in time and diffuse in space. That means, if we disturb the interior of the solution region at $t_1 > t_0$, it influences the solution only at a later stage where $t > t_1$ and diffusion ensures smoothness of the solutions (Versteeg and Malalasekera, 2007, p.28). If $t \to \infty$, or $\frac{\partial \Phi}{\partial t} = 0$, the steady state has been reached and the governing equation is now the one governing the steady distribution of Φ and is elliptic (Versteeg and Malalasekera, 2007, p. 28). Hyperbolic equations govern the analysis of time-dependent processes with a negligible energy dissipation (Versteeg and Malalasekera, 2007, p. 28),

such as steady or unsteady inviscid compressible and incompressible flows (Wendt *et al.*, 2009, p.82). The wave equation is a typical hyperbolic equation, where c is the wave speed (Versteeg and Malalasekera, 2007, p. 28):

$$\frac{\partial^2 \Phi}{\partial t^2} = c^2 \frac{\partial^2 \Phi}{\partial x^2} \tag{3.3}$$

Hyperbolic PDEs have special characteristics, which are related to the wave speed: they allow the wave propagation (or information to travel) at finite speed, and the occurrence of discontinuities (shocks) in the solution (Versteeg and Malalasekera, 2007; Shyy, 2006, p. 29, 89). For non-linear PDEs, the classification cannot be made that easily. For hydrodynamic and MHD equations, for instance, convection or diffusion terms can be added, which are hyperbolic and parabolic, respectively. The classification, therefore, depends on the dominating term and can only be made locally, for a region in the domain (Wendt *et al.*, 2009, p. 81). Depending on the mathematical problem or the governing equation, the boundary values have to be chosen. The boundary-value problem requires the solution of a differential equation or system of equations in a region R, at which boundaries additional conditions have to be taken into consideration (Scheid, 1968, p. 382). For our simulation, the solution of the Navier-Stokes and Maxwell equations is approximated with the Kurganov-Tadmor scheme, which is a finite volume scheme and further explained in Sec. 3.3. Common spatial discretisation or integration methods are the forward or upwind Euler, backward or downstream Euler, or the central scheme; common temporal discretisation or integration schemes are the forward or explicit Euler, backward or implicit Euler, or Runge-Kutta. The classical numerical discretisation methods are finite difference (FD), finite volume (FV) and the finite element (FE) methods (Peiro and Sherwin, 2005). There are three main aspects for the analysis of discretisation schemes that are also interrelated: the accuracy, stability, and consistency and convergence of the scheme used to discretise the problem (Shyy, 2006, p.3-5). Consider a differential equation that describes the variation of a function U, with respect to one or more independent continuous variables like x (space coordinate in one dimension) and t (time coordinate). To apply, for example, the finite difference method, we now want to get a discrete function u, which is a function of x and t at discrete points. If we let h and k be positive constants, so that $x_m = mh$ and $t_n = nk$ (with integers m and n) are spatial grid points and time instants, respectively, the corresponding function at (x_m, t_n) will be u_m^n, i.e. $u_m^n = U(x_m, t_n)$ (Shyy, 2006, p. 3). We

replace the differential operators $l(u)$ and apply finite difference operators $L(u)$ like the:

1. forward difference : $\delta u_m = u_{m+1} - u_m$
2. backward difference : $\delta u_m = u_m - u_{m-1}$
3. central difference : $\delta u_m = u_{m+1} - u_{m-1}$.

The *consistency* of a scheme has to do with whether a difference operator $L(u)$ can truly represent the differential operator $l(u)$ as $h, k \to 0$, and *convergence* with whether the finite difference solution approaches the exact differential solution in the limit, i.e. $u_m^n \to U(x_m, t_n)$ as $h, k \to 0$ (Shyy, 2006, p. 3-5). Basically, a numerical scheme is consistent if the discrete numerical equation tends to the exact differential equation as the mesh size tends to zero (Peiro and Sherwin, 2005, p. 25). For example, we consider the linear advection equation $u_t + au_x = 0$ (where the subscripts t and x indicate the derivative with respect to t (time) and x (space)), respectively, which is given by a central in space and forward in time scheme:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} = 0. \tag{3.4}$$

The superscript n indicates the step in time and the subscript i the grid point in space. Here, the Courant number $C = a\Delta t/\Delta x$, where a is the velocity, Δx the grid spacing and Δt the time step, we obtain:

$$u_i^{n+1} = u_i^n - \frac{C}{2}(u_{i+1}^n - u_{i-1}^n). \tag{3.5}$$

With the Taylor expansions for u_i^{n+1}, u_{i+1}^n and u_{i-1}^n:

$$\begin{aligned} u_i^{n+1} &= u_i^n + (\Delta t)u_t|_i^n + \frac{(\Delta t)^2}{2}u_{tt}|_i^n + \ldots \\ u_{i+1}^n &= u_i^n + (\Delta x)u_x|_i^n + \frac{(\Delta x)^2}{2}u_{xx}|_i^n + \frac{(\Delta x)^3}{6}u_{xxx}|_i^n + \ldots \\ u_{i-1}^n &= u_i^n - (\Delta x)u_x|_i^n + \frac{(\Delta x)^2}{2}u_{xx}|_i^n - \frac{(\Delta x)^3}{6}u_{xxx}|_i^n + \ldots \end{aligned} \tag{3.6}$$

substituted in (3.4) and re-arranged, we yield:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} - (u_t + au_x)|_i^n = \epsilon_T, \tag{3.7}$$

where ϵ_T is the truncation error given by:

$$\epsilon_T = \frac{\Delta t}{2} u_{tt}|_i^n + \frac{(\Delta x)^2}{6} a u_{xxx}|_i^n + O((\Delta t)^2, (\Delta x)^4). \tag{3.8}$$

If Δt and Δx go to zero, the LHS (ϵ_T) will tend to zero, and we can say that the approximation is consistent, as the numerical scheme approaches the exact equation at point x_i and time t^n (Peiro and Sherwin, 2005, p. 27).

Stability refers to the computed solution and the exact solution of the discrete equation to be bounded, hence u_m^n is bounded as $n \to \infty$ (Shyy, 2006, p. 3-5). For explicit schemes, at least the Courant-Friedrichs-Lewy (CFL) condition is to be fulfilled for them to be stable and, therefore, for the errors not to grow uncontrolled (Peiro and Sherwin, 2005). The CFL condition is based on the theory of characteristics for hyperbolic systems. It says that the domain of dependence of a PDE is the portion of the domain that influences the solution at a given point. The CFL condition expresses that for each point in the mesh the domain of dependence of the FD approximation has to contain the domain of dependence of the PDE in order for the explicit scheme to solve a hyperbolic PDE to be stable (Peiro and Sherwin, 2005). In other words, the domain of dependence of the discretisation must include the domain of dependence of the PDE. For hyperbolic and parabolic PDEs, the stability can be analysed with the von Neumann stability analysis, which is based on the decomposition of the errors into Fourier series. To sum up, "Stability is the necessary and sufficient condition for a consistent linear FD approximation to a well-posed linear initial-value problem to be convergent." (Peiro and Sherwin, 2005). The accuracy of a scheme relates to how closely u_m^n approximates $U(x_m, t_n)$ as h and k vary, and is often characterised by the order of accuracy.

3.2 Finite Volume (FV) Method

The scheme used for the code is based on the FV method. Unlike the FD method, where the differential or strong form is used, in finite volume methods it is the integral form of the equations that provides the basis (Peiro and Sherwin, 2005). This has the advantage

of being a good scheme for discontinuous (or weak) solutions too (Keppens, 2007) and providing a more natural treatment of Neumann boundary conditions and discontinuous source terms, as it does not require the solution to be as smooth. Furthermore, the integral formulations do not depend on a special mesh structure and deal with curvature more naturally and, therefore, make FV methods more suitable for complex geometries in multi-dimensions (Peiro and Sherwin, 2005). Moreover, it ensures global continuity and, therefore, is consistent with the mathematical structure of Navier-Stokes equations (Shyy, 2006, p. 116).

First of all, the region we integrate over is a finite volume (usually called "control" volume) around the point x, represented by $x_{i-(1/2)} \leq x \leq x_{i+(1/2)}$. The integral form of our conservative variable u and its flux $f(u)$ would be:

$$\int_{x_{i-(1/2)}}^{x_{i+(1/2)}} u_t dx + \int_{x_{i-(1/2)}}^{x_{i+(1/2)}} f_x(u) dx = 0 \tag{3.9}$$

where the subscripts denote the derivation in time t and space x, respectively. The flux term can be obtained with:

$$\int_{x_{i-(1/2)}}^{x_{i+(1/2)}} f_x(u) dx = f(u_{i+(1/2)}) - f(u_{i-(1/2)}). \tag{3.10}$$

This way of approximating the solution produces a conservative scheme if the flux on the boundary of one cell equals the flux on the boundary of the neighbouring cell (Peiro and Sherwin, 2005, p. 22). This means the changes in the cell average value can only happen through losses or gains through the cell boundary. If a scheme does not fulfil this conservative discretisation form to correctly treat discontinuous or weak solutions, incorrect shock speeds can be produced (Keppens, 2007). This control volume approach, therefore, requires the construction of the fluxes at the interfaces or across the control surfaces (Shyy, 2006, p. 116). This needs an extrapolation within the cell i from the volume averaged value U_i. This extrapolation can be achieved in two ways. Either by means of a constant extrapolation, where the values at the interface or edges are equal to the value at the cell centre ($u(x \in [x_{i-1}, x_{i+1}]) = u_i$) or by a linear extrapolation with a slope σ'. The constant extrapolation is consistent with u_i, but leads to a first order of accuracy, where the fluxes at the interface are calculated as an average (e.g. $f_{i+1/2} = (f_i + f_{i+1})/2$) (Keppens, 2007). The linear extrapolation is still also consistent with u_i, but yields a

second order accuracy (LeVeque, 2002, p. 106):

$$u(x \in [x_{i-1}, x_{i+1}]) = u_i + \sigma_i' \frac{x - x_i}{\Delta x_i}, \tag{3.11}$$

where

$$x_i = \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) = x_{i-1/2} + \frac{1}{2}\Delta x. \tag{3.12}$$

Let's say the slope is the difference of the values at the interfaces:

$$\sigma' = u_{i+1/2} - u_{i-1/2} \equiv \Delta u_i. \tag{3.13}$$

This linear reconstruction is used to obtain a *left* and *right* value for one interface, for example for $u_{i+1/2}$, there would be a $u^L_{i+1/2}$ and also a $u^R_{i+1/2}$:

$$u^L_{i+1/2} = u_i + \Delta u_i/2 \tag{3.14}$$

and

$$u^L_{i+1/2} = u_{i+1} - \Delta u_{i+1}/2, \tag{3.15}$$

and the flux at one cell interface takes the average form:

$$f_{i+1/2} = (f(u^L_{i+1/2}) + f(u^L_{i+1/2}))/2. \tag{3.16}$$

When using the linear reconstruction approach to calculate the fluxes at the interfaces, the slopes involved must be limited to avoid the introduction of spurious oscillations (Keppens, 2007). This is done by comparing the slopes of neighbouring cells and taking the least steep slope. Furthermore, if the slopes differ in sign, then the reconstruction of the fluxes at the cell centre is done with the first order accuracy constant reconstruction (Keppens, 2007). If the slope is zero (Godunov's method), first order accuracy is achieved. Second order accuracy slopes can be obtained in three ways:

1. centered slope: $\sigma_i'^n = \frac{u^n_{i+1} - u^n_{i-1}}{2\Delta x}$
2. upwind slope: $\sigma_i'^n = \frac{u^n_i - u^n_{i-1}}{\Delta x}$
3. downwind slope: $\sigma_i'^n = \frac{u^n_{i+1} - u^n_i}{\Delta x}$,

which are also referred to as the Fromm, Beam-Warming and Lax-Wendroff method, respectively (LeVeque, 2002, p. 108). In general, it is desirable for the slope to be second-order accurate in smooth regions, and around discontinuities, a first order slope calculation is applied. If the discontinuity stretches out over more than one cell and care is taken to avoid oscillations, even in discontinuities non-zero slopes can help keep the solution from smearing out too far and, therefore, increase the resolution and keep discontinuities sharp (LeVeque, 2002, p. 108). Various applications of first and second order accuracy slope calculation depending on the conditions are called slope-limiting methods and were first introduced by van Leer in his monotonic upstream-centred scheme for conservation laws (MUSCL). So, ideally, we want to apply a second order method for the slope, but at the same time have to make sure that there are no non-physical oscillations that arise. This can be done by measuring the oscillation in the solution with the *total variation* of a function, defined as:

$$TV \equiv \Sigma_i |u_{i+1} - u_i|. \tag{3.17}$$

In essence, it is the sum of the absolute values of the differences of one cell to a neighbouring cell. A scheme is called *Total Variation Diminishing (TVD)* (Harten, 1982), if it ensures that this total variation diminishes with time, hence:

$$TV^{n+1} \leq TV^n. \tag{3.18}$$

If this is fulfilled, it is ensured that monotone initial data remain monotone and, therefore the TVD property of a scheme guarantees it is monotonocity preserving and the creation of spurious oscillations in the numerical solution is avoided (Keppens, 2007). All in all, to make sure that no spurious oscillations develop with time, or to ensure the TVD property of the solution (Sweby, 1984), the creation and growth of local extrema (Wendt *et al.*, 2009; Versteeg and Malalasekera, 2007) is suppressed with a limiter that is applied to the flux terms of our conservative quantities. Hence, solving hyperbolic PDEs numerically, requires methods that can handle solution discontinuities and do not generate non-physical oscillations.

The Kurganov-Tadmor scheme, which will be described in the next section, deals with discontinuities and controls spurious oscillations by ensuring TVD properties. Another way of oscillation control are explicit artificial diffusion terms, which are added to the equations by means of an artificial viscosity coefficient. When artificial diffusion is

chosen, this viscosity coefficient can be a scalar or it is substituted by a more complex diffusion matrix. For a scalar viscosity term, the consequence it that it often leads to too much diffusion in a large part of the the domain, where it is not needed and wished. By applying limiter functions for each field, as done in the KT scheme, the optimal amount of artificial viscosity is added (LeVeque, 2002). If the explicit artificial viscosity is not a scalar but a matrix, those artificial diffusion schemes are equivalent to FV schemes with TVD, for instance. Those TVD schemes are often considered as part of the artificial viscosity models too (Denner *et al.*, 2017). In fact, could also be translated into an explicit artificial viscosity term (Davis, 1987). However, there are some situations where one approach of dealing with spurious oscillations (scalar artificial viscosity term or FV scheme with TVD / diffusion matrix) could be more advantageous than the other. For transonic steady flows, where shock waves are not very strong, scalar artificial diffusion is the better approach, as it is computationally cheap and easy to code up. However, for flows associated with strong shock waves, the FV scheme and TVD or the matrix diffusion are needed, as they provide the best possible resolution of shock waves and contact discontinuities (Wendt *et al.*, 2009, p. 221). Depending on the simulation, either the Minmod limiter (Roe, 1986) as in Eqn. (3.19) is used or the Optimum Symmetric Polynomial-Ratio Expression (OSPRE) limiter (Waterson and Deconinck, 2007) as in Eqn. (3.20):

$$\Psi(r)_{Minmod} = max[0, min(r, 1)], \tag{3.19}$$

$$\Psi(r)_{OSPRE} = \frac{3r(r+1)}{2(r^2+r+1)}, \tag{3.20}$$

where r is the ratio of the neighbouring gradients (Waterson and Deconinck, 2007), defined as

$$r_i = \frac{u_i - u_{i-1}}{u_{i+1} - u_i}. \tag{3.21}$$

The use of flux limiters is an effective way to construct non-linear discretisation schemes, which adjust themselves according to the local solution to maintain bounded and with that fulfill the TVD condition. Flux limiters are simple functions which take the form of piece-wise linear functions, ratios of equal order polynomials or combinations of the two (Waterson and Deconinck, 2007). We want a non-linear convective scheme, because, according to Godunov's theorem, linear convection schemes with second order accuracy

or higher cannot be monotonic. And as monotonicity, or TVD has to be ensured to avoid the development of non-physical oscillations, a convective scheme has to be non-linear if high accuracy is aimed for.

3.3 The Kurganov-Tadmor Scheme

The Kurganov-Tadmor (KT) scheme was developed and introduced in 1999 with the aim of creating a new central scheme that is independent of the eigenstructure of the problem (which means no calculation of the Jacobian matrix or characteristic decomposition) and also showing a small numerical viscosity. A fully-discrete scheme and a semi-discrete scheme were developed, where the latter can be combined with any time stepping scheme wished. In this work, the semi-discrete version of the KT scheme is used, combined with a forward Euler or fourth order Runge-Kutta scheme. Kurganov and Tadmor (2000) prove that their semi-discrete second order central scheme also satisfies the TVD property and, therefore, prevents the development of spurious oscillations. Furthermore, the scheme also shows an independence of the numerical viscosity on the time step, therefore, high-resolution can be achieved even with very small time steps.

For convection-diffusion equations, achieving the solution with an explicit scheme is computationally cheaper, but also restricts the choice of time step to the parabolic CFL condition $\Delta t \leq C(\Delta x)^2$, with Δx being the grid spacing (Cavalli *et al.*, 2006). In general, when a semi-discrete scheme is coupled with an appropriate ODE solver, a small numerical viscosity is obtained, which is proportional to a vanishing time step, Δt. Most (fully discrete) schemes work best close to a convective CFL $\Delta t \leq C\Delta x$, i.e., $\Delta t \sim \Delta x$. If the time step is much smaller, they accumulate large numerical dissipation of the order $O(\Delta x^{(2r)}/\Delta t)$, with r being the accuracy of the scheme, and if therefore $\Delta t \to 0$, the numerical viscosity becomes very large (Cavalli *et al.*, 2006). When discontinuities are resolved, the computed sub-shocks are smeared due to larger numerical dissipation, which accumulates with every time step. The KT semi-discrete scheme allows for smaller time steps due to a small numerical viscosity of the order of $O(\Delta x^{(2r-1)})$, which is time step independent. This is achieved by treating smooth and non-smooth regions separately (Kurganov and Tadmor, 2000). The non-smooth region, which is at the cell interface (i.e. between u_j^n and u_{j+1}^n) and the resulting Riemann fan is illustrated in Fig. 3.1. This is convenient in solar physics, as time steps for simulations of the solar atmosphere can get very small. This is

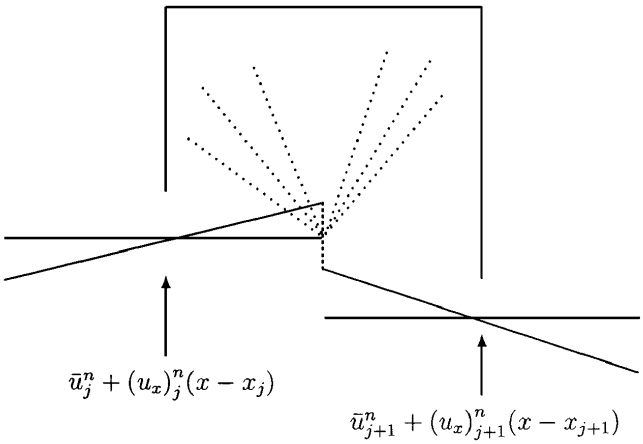

Figure 3.1: Illustration of the central differencing approach and the staggered integration over the local Riemann fan, which results from the discontinuity at the cell interface, is shown (from Kurganov and Tadmor (2000), Fig. 2.1.)

due to frequent collisions in the photosphere and chromosphere, which then lead to stiff source terms (González-Morales *et al.*, 2018).

Compared to upwind schemes, central schemes have the advantage of being independent of the eigenstructure of the problem and are Riemann-solver free, which makes computation cheaper and easier. In the KT scheme, the values of the state variables are calculated as an average over the cell, and the fluxes are computed at the left and right cell interfaces. In this Riemann-solver free approach, the fluxes are calculated in terms of the discrete values of the neighbouring cells, where the approximate flux derivatives are computed in a component-wise manner, instead of being obtained from the eigenvalues (which are the speeds of characteristic waves like the sound wave) of the flux Jacobians. This is illustrated in Fig. 3.2 and discussed in the following. Given is the solution u_j^n, at point x_j in space and point n in time. To yield the solution at the next time step $n+1$, the idea is to average over the non-smooth Riemann fans. To do so, the KT scheme uses local wave propagation speeds. This means that first, the propagation speed at the cell interfaces at $u_{i\pm1/2}$ is calculated, where we are interested in the maximum propagation speed and this results in new cell widths, w in Fig. 3.2. Over these smaller cells the solution is computed. These cells are essentially the width of the Riemann fan, which originates at the interfaces and which can vary in size at each time step. In fact, the calculation of the maximum propagation speeds has often been done already, when calculating the CFL

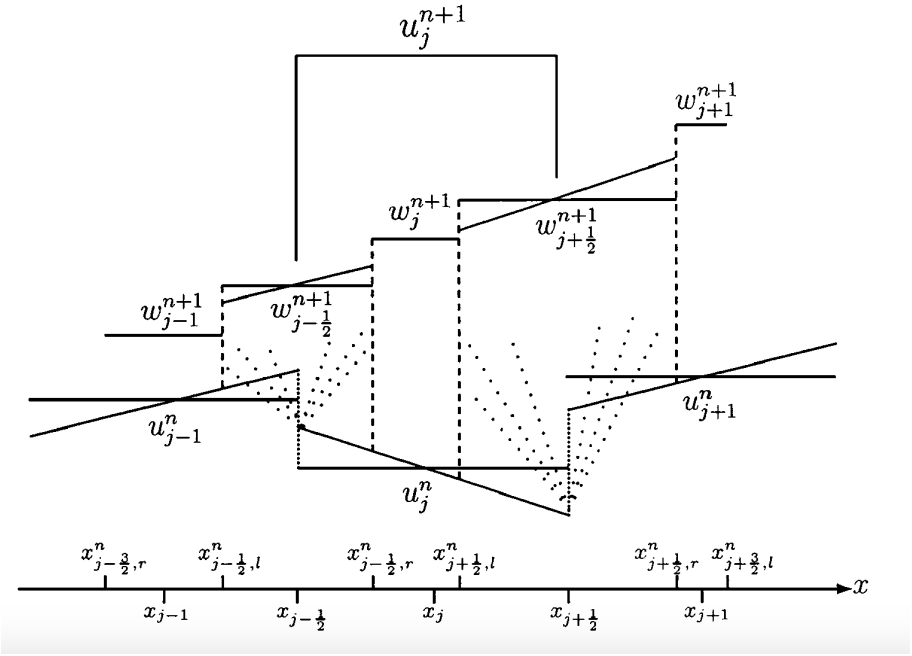

Figure 3.2: This figure illustrates the steps from the solution u_j^n, at point x_j in space and point n in time, to the solution u_j^{n+1}, i.e. at the next time step. It shows the cells w, which are obtained by computing the maximum propagation speed at the cell boundaries and which are used to realise the solution at the next time level (Kurganov and Tadmor (2000), Fig. 3.2).

number. Therefore, if the propagation speed is a, the control volume between $x_{j-1/2} - a$ and $x_{j-1/2} + a$, denoted in Fig. 3.2 by $x_{j-1/2}, l$ and $x_{j-1/2}, r$, respectively and, therefore, the new control volume w has the size $2a$. Due to finite speed of propagation, those points separate between smooth and non-smooth regions. These non-uniform averages on the staggered grid are then converted back into the original grid. Non-staggered grids have the advantage of being simpler to implement if the geometry and boundaries are more complex (Jiang *et al.*, 1997). In their paper, Jiang *et al.* (1997) show how to convert a scheme that is based on a staggered grid to one that is non-staggered, this has been followed by Kurganov and Tadmor as well. In that way, the numerical solution is updated on the edges of the staggered grid, where it is smooth and can be computed via a Taylor expansion, and the computationally expensive Riemann problem does not have to be solved (Cavalli *et al.*, 2006).

Summing up, behind the construction of the KT scheme there are two main ideas: firstly, it uses more precise information of the local propagation speeds and secondly, the scheme constructs the non-smooth part of the approximate solution in terms of its cell averages, which can vary in size, integrated over the non-smooth Riemann fans (Kurganov and Tadmor, 2000).

As mentioned before, the time integration is achieved with the forward Euler or fourth-order Runge-Kutta scheme. The forward Euler scheme is a simple linear scheme, where the solution at the next time step $u^{t+1} = u^t + hf_t$, where h is the discretisation step size (Quarteroni, Sacco, and Saleri, 2006, p.472). The Runge-Kutta methods are actually a family of schemes of different orders, where the first order Runge-Kutta method equals the forward Euler method. For the 4th order Runge-Kutta method (Quarteroni, Sacco, and Saleri, 2006, p.511), the slope is calculated in four steps with different weighing of each step and leads to:

$$\begin{aligned} u(t_0 + 1) &= u(t_0) + mh \\ &= u(t_0) + \left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)h, \end{aligned} \tag{3.22}$$

where $u(t_0)$ is a known initial condition and the solution to $u(t)$ is found with u_0 and a

weighted sum or weighted average, m, of the slope approximated, and with

$$\begin{aligned} k_1 &= f(u(t_0), t_0), \\ k_2 &= f(u(t_0) + k_1 \frac{h}{2}, t_0 + \frac{h}{2}), \\ k_3 &= f(u(t_0) + k_2 \frac{h}{2}, t_0 + \frac{h}{2}), \\ k_4 &= f(u(t_0) + k_3 h, t_0 + h). \end{aligned} \tag{3.23}$$

Chapter 4

The Code and Its Verification

The equations governing hydrodynamics and magnetohydrodynamics are highly non-linear partial differential equations (PDEs). As a result, propagating waves and various discontinuous waves are formed throughout the simulation, which makes computation very difficult, even with modern schemes and computers (Hesthaven, 2018). The scheme implemented in this code, the Kurganov-Tadmor scheme, has been introduced in the previous chapter, Sec. 3.3. This scheme advances the solution on a Cartesian grid, where a variety of boundary conditions are implemented to account for different physical settings. In the following, the code set-up and the simulations and tests, which substantiate the code's stability and accuracy, are presented.

4.1 Code Setup

The system of non-linear PDEs for conservation laws like advection-diffusion equations are solved with the Kurganov-Tadmor scheme (Kurganov and Tadmor, 2000) as described in Sec. 3.3. The time integration is achieved with the forward Euler or fourth-order Runge-Kutta scheme (Hartley and Wynn-Evans, 1979; Atkinson, 1989). This is done on a uniform Cartesian grid, which means that the cells have the same size everywhere in the computational domain. As illustrated in 4.1, the Cartesian grid is made of $n\_x\_global \times n\_y\_global$ cells, where n_x_global and n_y_global represent the number of total grid points in the x and y direction, respectively. The cell size dx, is determined by the physical dimension and the number of grid points, so that $dx = physical\_domain/n\_x\_global$,

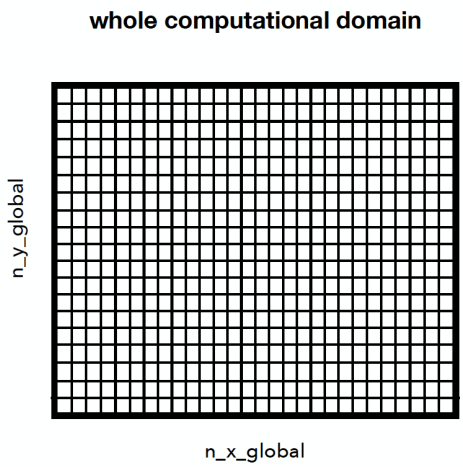

Figure 4.1: Computational domain.

and the same applies for dy.

The time step is controlled through the speed v_{max}, which is the maximum of the Alfvén speed and the sound speed, $v_{max} = max(v_A, c_s)$. The time step is calculated with $dt = CFL(dxy_{min}/v_{max})$, where CFL is a Courant-Friedrichs-Lewy coefficient (smaller than one for this explicit solver) and dxy_{min} is the minimum (if grid cell size varies) spatial grid cell size over all directions. If collisions in the plasma are numerous, then the very small collision time scale and the related elastic collision term in the source term of the equations (Eqns. (2.134)), represent another possible constraint to the time step. Furthermore, perpendicular to the magnetic field there is the time scale of the magnetoacoustic waves that add to that constraint, however in a plasma with a low plasma beta $\beta = \frac{c_s}{v_A} << 1$, the fast magnetosonic speed can be approximated by the Alfv́en speed.

Message Passing Interface (MPI) allows the parallel computation of each subdomain on different central processing units (CPUs), which can make computation faster. Therefore, the computational domain is divided into subdomains and each subdomain is sent to a different CPU to be calculated, which is illustrated in Fig. 4.2. Each subdomain has a size of $n_x = n\_x\_global/n\_cpu\_x + 4 * n\_ghost$ in x-direction and $n_y = n\_y\_global/n\_cpu\_y + 4 * n\_ghost$ in y-direction, where n_cpu_x and n_cpu_y denote the number of CPUs used for each dimension. n_ghost is the number of ghost cells, which are added to each side of the subdomain. The values for the ghost cells come from

the neighbouring subdomains, therefore, the subdomains have to communicate with each other and send and receive the values for their ghost cells from each other, as illustrated in Fig. 4.3. At each MPI exchange $4 * n\_ghost$ cells are needed for each dimension. For example, if continuous boundary conditions are applied, the ghost cells at $i = 0$ and $i = 1$ of subdomain 2 receive the values from the last two cells of subdomain 1. Therefore, at the same time, a subdomain sends and receives values for/of ghostcells from/to another subdomain, where the received values are stored in a buffer array. Whereas the boundaries between the subdomains are usually always continuous to allow continuous flow inside the computational domain, the outer boundaries (the boundaries of the global computational domain) depend on the physical setting.

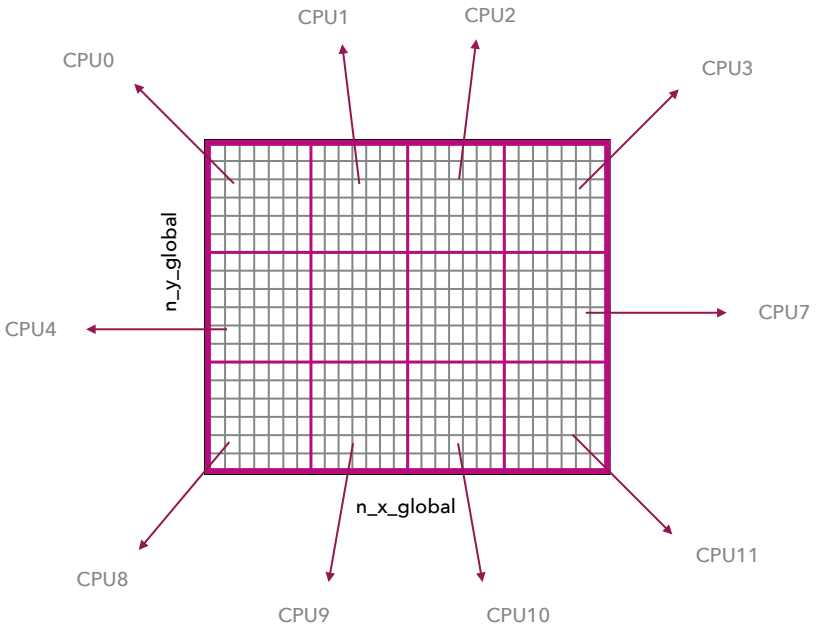

Figure 4.2: Divided computational domain into subdomains.

The code, by its construction, is a reconfigurable code. To reduce the 2D configuration to 1D, all variables in the y-direction are set to zero and leave 10 cells in the y-direction to account for the calculation of the boundaries and accommodate the finite volume scheme. Boundary conditions are problem specific and are determined by the physical setup. Usu-

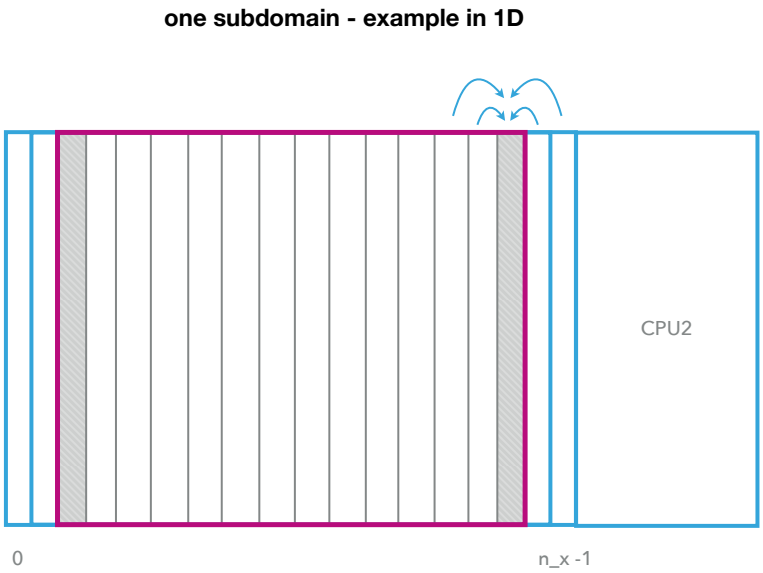

Figure 4.3: Example of one subdomain with a neighbouring subdomain (blue).

ally, we have an idea or information about what we expect to happen at the boundary and yet we still have to decide which the right ones are that are to be applied. If there are too many or not enough or inappropriate conditions applied, the mathematical problem is ill-posed and will have no or false solutions (LeVeque, 2002, p. 60). To accommodate for various problems, various boundary conditions are implemented in the code that can be exchanged or chosen as needed for every boundary of the domain. For now, these are periodic, continuous, fixed, and fixed zero value boundary conditions. Boundaries are represented as two additional rows in the computational grid (Toth, 1994). We call the two boundary cells ghost cells; the ones at the beginning of the grid we call ghost cells at $i = 0$ and $i = 1$ and the ghost cells at the end of the domain are at $i = n\_x - 1$ and $i = n\_x - 2$. Therefore, the two first grid cells are at $i = 2$ and $i = 3$ and the two last cells are at $i = n\_x - 4$ and $n\_x - 3$ of the computational domain. Figure 4.3 depicts the idea. The blue arrows in this graphic illustrate that the information of the two neighbouring cells are needed to calculate the solution for one cell. This also shows the necessity of boundaries.

Periodic boundary conditions describe a computational domain, where anything that leaves the domain on one side, must appear on the other side, which means that the ghost cells at the beginning of the domain (at $i = 0$ and $i = 1$) take the same values as the cells at the end of the domain (at $i = n\_x - 4$ and $n\_x - 3$) and the ghost cells at the end of the domain at ($i = n\_x - 1$ and $i = n\_x - 2$) must take the same values as the cells at the beginning of the domain $i = 2$ and $i = 3$. Continuous boundary conditions are implemented to make the computational domain finite and describe the absorption of a long wavelength wave that hits the boundary without reflection (LeVeque, 2002, p. 134). Therefore, the ghost cells at $i = 0$ and $i = 1$ take the same values as the first two cells at $i = 2$ and $i = 3$ and the ghost cells at the end of the domain $i = n\_x - 1$ and $i = n\_x - 2$, take the values of the last inner cell of the domain at $i = n\_x - 4$ and $n\_x - 3$. As a result, the gradient vanishes at the boundaries and reflectivity is low. This is also a good approximation of so-called free boundaries. The fixed boundary conditions impose the ghost cell values that were set at the beginning of the run at every time step, and the fixed zero boundary conditions impose a zero value to the ghost cells at every time step.

4.2 Code Structure and Functions

This code solves a system of partial differential equations. This can be done for multiple fluids in multiple dimensions and the extension is very simple from a software engineering point of view as the structure of the code allows to do so. The code uses Message Passing Interface (MPI) to make computation parallel and faster. The implementation of MPI has been done for 2D only.

The code is written in C++ and starts from the main.cpp function. Figure 4.4 comprises the overall structure of the code and its major functions. The purpose of the functions are described below. To give an idea of the overall structure: the simulation starts with initial conditions, which determine the time evolution of the simulation. From there, the solution for the next time step is calculated, starting with the function apply_RK.cpp. In this illustration, the colours represent different levels: all the functions inside the purple frames are part of the apply_RK.cpp function, all the functions inside the yellow frame are part of the calc_residual.cpp function and all the functions in with the turquoise frames, are inside the numerical_flux.cpp function. In apply_RK.cpp, the residuals (the divergence of the flux terms) are calculated first. In order to calculate the fluxes, the numerical flux is

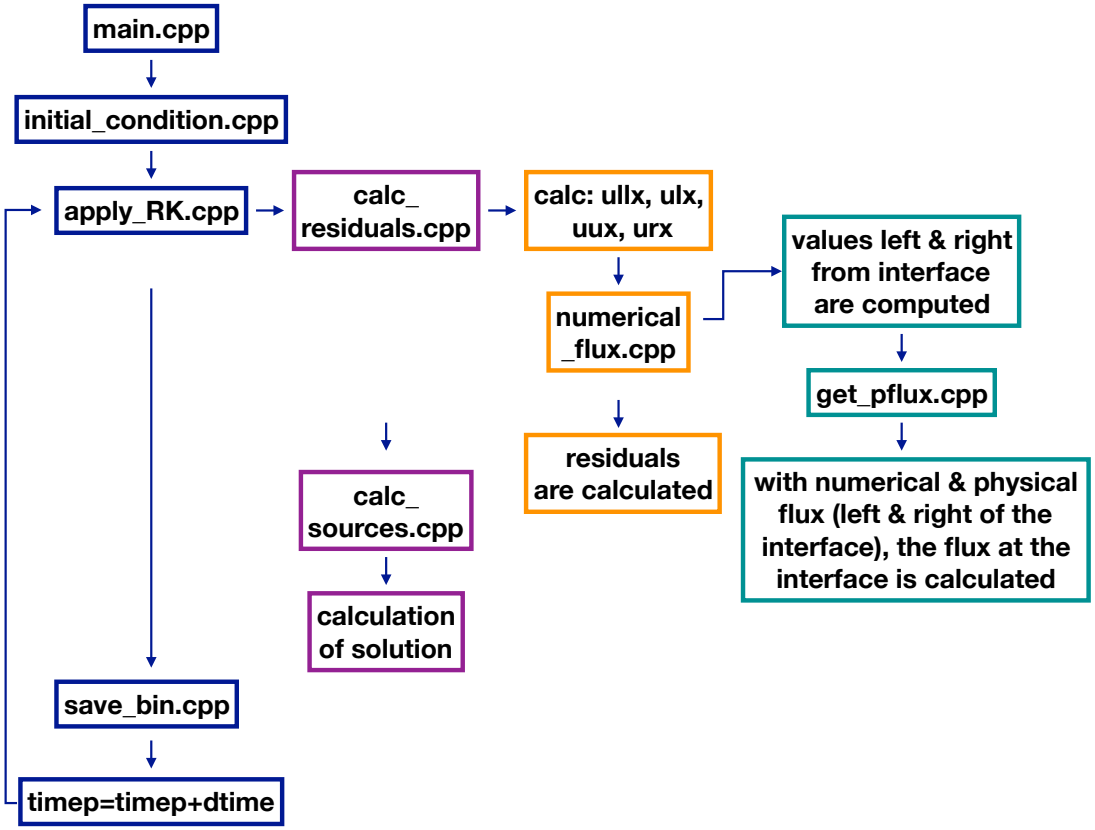

Figure 4.4: Flow diagram with the major functions of the code.

computed first. This requires the extrapolation of the cell centre in order to calculate the fluxes at the interfaces. Once the physical flux (the flux term in the equations) is determined, the residuals and source terms are computed and the solution is advanced in time.

main.cpp

In this function, all other functions are declared and the memory is allocated for the arrays. The initial condition function, the functions to calculate the temperature and convert primitive variables to conservative variables are called and boundary conditions are applied. The only variables that need to be adjusted in this function are **ch** and **cp** for the divergence cleaning scheme and the time **timep** at which the simulation is wished to be stopped.

If there is a certain physical time the simulation is supposed to stop, this can be set as

follows.

timep=timep+dtime; //physical time
if (timep >= 3.) break;

This would stop the simulation after 3 time units in physical time. The main function calls the function RK.app, where the majority of the functions are called and the solution is calculated.

initial_condition.cpp:
Here, initial conditions are set. Usually, they are given in terms of the primitive variables like density and pressure and are then converted to conservative variables. Various initial conditions have been implemented so far and are given a number, as done in the following: This can be done like this:

if type == number {initial conditions}

Each initial condition is, therefore, given a different number. This number then has to be changed in the **constants.h** file like:

const int type = number;

Depending on the situation to be simulated (the initial conditions), the physical dimension has to be adjusted. This can be done in **constants.h**:
const double start_x_global = 0.;
const double start_y_global = 0.;
const double ende_x_global = 1.;
const double ende_y_global = 1.;

apply_BC.cpp:

Boundary conditions can be changed in **apply_BC.cpp**, where we indicate at which end

of the domain the boundary is applied to in the following manner :

bc_type[rho_1_][0][0] = 0; //left
bc_type[rho_1_][0][1] = 0; //right
bc_type[rho_1_][1][0] = 2; //bottom
bc_type[rho_1_][1][1] = 2; //top

The above is an example of the boundaries in each direction for the density rho.
Different types of boundaries can be applied and are assigned the following numbers:
0 - periodic boundary conditions
1 - continuous boundary conditions
2 - global fixed boundary conditions
3 - global zero value boundary conditions
where the according number is entered (like in the example above) for the boundary (left, right, top, bottom) and quantity (rho, mom_x etc) of choice.

get_pflux.cpp:
This function calculates the physical flux exactly according to the flux terms in the equations in conservative form. This is done for the whole system of equations, in every direction.

RK.cpp:
This function contains different functions for the time stepping scheme, like the first order Runge-Kutta scheme, i.e. the explicit Euler scheme, or the forth order Runge-Kutta scheme, which are **apply_RK1.cpp** and **apply_RK4.cpp**, respectively. Each of them calls the functions **calc_residuals.cpp** and **calc_sources.cpp**, where the flux terms and source terms, respectively, are obtained. With the residuals and source term, the solutions is calculated and integrated in time.

calc_sources.cpp:
In this function, the source terms are calculated for each quantity of our system of equations, which is then added to the residual term. The function (**calc_rates.cpp**) is called here as well, which calculates the ionisation and recombination rates.

calc_rates.cpp:
In this function, the collision, ionisation, and recombination rates are calculated.

calc_residuals.cpp:
To calculate the residuals, the **numerical_flux.cpp** function is called, which returns the flux terms, with which the residuals are calculated, using a central scheme.

calc_temperature.cpp:
The temperature for each fluid is computed in this function.

con_to_prim.cpp:
Here, the primitive variables are obtained from conservative variables.

numerical_flux.cpp:
In this function, the values left and right of the interface *ul* and *uu* are calculated for each conservative variable and are then called *ulmh* and *urmh*. At $ulmj$ and *urmh* the sound speed and the Alfvén speed are calculated and the fastest propagation speed is determined. Then the physical flux (the flux terms like in the equations) is computed with the function **get_pflux.cpp**, left and right from the interface (therefore, the function is called twice and returns the array flr and fll). Then the overall flux is computed. The array $flux$ is returned, which contains the value for the flux at the interface between *ul* and *uu*. In **calc_residuals.cpp**, the flux between two interfaces is then calculated for all interfaces of the whole computational domain, before the residuals for the whole domain is calculated.

prim_to_con.cpp:
Here, the conservative variables are obtained from primitive variables.

save_bin.cpp:
The arrays are saved as binaries with this function. Where the data will be saved to and under what name, can be set as follows.

filename = “/path/to/destination/folder/name_of_data_”+timestep;

Furthermore, the number of time steps for which the data will be saved, can be determined in the **constants.h** file.

const int i_t_save = 10;

This saves every ith (here, every 10th) time step.

timestep.cpp:
In this function, the maximum information propagation speed is calculated and the time step is determined.

4.3 Code Tests

Having established how the code is set up and how the equations are solved in time and space, we now turn to its applications and the tests performed in order to verify the code and ensure that it does what it is supposed to do, i.e. to demonstrate its accuracy and stability. Therefore, to model the interaction of neutral and ionised fluids, the two sets of hydrodynamic and MHD equations are solved. The general form of the equations is as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{FL}(\mathbf{U}) = \mathbf{S}. \tag{4.1}$$

Here, $\mathbf{U}$ is the state or solution vector, $\mathbf{FL}(\mathbf{U})$ is the flux vector of $\mathbf{U}$, and $\mathbf{S}$ denotes the source term. In Cartesian geometry, the system is written out as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} + \frac{\partial \mathbf{H}}{\partial z} = \mathbf{S}, \tag{4.2}$$

where **F**, **G**, and **H** are the flux vectors in the x, y, and z-direction, respectively. The equations in the system of equations have the form of a diffusion-advection system of equations and is in dimensionless form. To secure and demonstrate that our code is accurate, various test simulations are performed. The following tests, such as the Sod shock tube test, the Brio-Wu shock test, or the 2D Orszag-Tang (OT) vortex simulation, have been carried out extensively and widely and, therefore, provide a good basis for the testing of our code. Firstly, the hydrodynamic test simulations are presented, before turning to MHD simulations and then, in the next chapter, the two-fluid simulations.

4.3.1 Hydrodynamic Simulations

For 2D hydrodynamics, which describes the neutral fluid, the vectors **U**, **F** and **G** are written as follows:

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \epsilon \end{bmatrix}, \qquad \mathbf{F} = \begin{bmatrix} \rho v_x \\ \rho v_x^2 + P \\ \rho v_x v_y \\ v_x(\epsilon + P) \end{bmatrix}, \qquad \mathbf{G} = \begin{bmatrix} \rho v_y \\ \rho v_x v_y \\ \rho v_y^2 + P \\ v_y(\epsilon + P) \end{bmatrix}. \tag{4.3}$$

Here, ρ is the fluid density, v_x and v_y are the fluid velocities in the x and y-direction, respectively. The total energy density ϵ (kinetic and thermal energy) and the gas pressure P are connected through the ideal gas equation of state, so that

$$\epsilon = \rho v^2/2 + P/(\gamma - 1). \tag{4.4}$$

This closes the system of equations. Furthermore, v^2 is defined as $v^2 = v_x^2 + v_y^2$ and γ is the adiabatic index. With the neutral fluid code, the Sod shock tube test is performed (Sod, 1978). When a fluid travels at supersonic speeds it creates a large change in pressure in a very short time, or, in other words, a shock front, which suddenly changes the state of the gas and usually leaves it with a higher pressure and temperature (Norman and Winkler, 1985). This test is one of the most fundamental ones for numerical codes and their ability to solve non-linear hyperbolic PDEs with discontinuous solutions, because there is an exact solution to compare the numerical solution to. An exact solution to

this initial value problem is presented in Lora-Clavijo *et al.* (2013). The test results in x-direction are presented in Fig. 4.6. This is a 1D simulation and the reduction of the code to 1D is achieved by setting the values of every variable in the y-direction to zero and reducing the grid cells in the y direction to 10 grid cells. Von Neumann, i.e. zero gradient boundary conditions are used. Non-linear equations of fluids inhere three types of non-linear "waves": the rarefaction wave, the contact discontinuity and the shock front (Norman and Winkler, 1985), which all occur in this Sod shock simulation too. Initial conditions for the problem in x-direction are:

$$\mathbf{U} = \begin{cases} \begin{bmatrix} \rho &=& 1 \\ P &=& 1 \\ v_x &=& 0 \\ v_y &=& 0 \end{bmatrix} & \text{for } x \leq 0.5, \\ \begin{bmatrix} \rho &=& 0.125 \\ P &=& 0.1 \\ v_x &=& 0 \\ v_y &=& 0 \end{bmatrix} & \text{for } x > 0.5. \end{cases}$$

These initial conditions describe the idea of a long 1D tube, which is divided in two halves, filled with fluids of different thermodynamic parameters like density and pressure (Danaila *et al.*, 2007). According to Danaila *et al.* (2007), the high-pressure part ($x \leq 0.5$, here: left) is called the driven section, whereas the low-pressure part ($x \geq 0.5$, here: right) is called the working section. At time $t = 0$, the gas is at rest. A sudden breakdown of the wall between the two parts, leads to a process that aims to even out the pressure and aims for pressure equilibrium, where the gas at high density and pressure expands through a rarefaction (or expansion) wave and generates a high-speed flow, which propagates into and pushes the gas of the working section. This rarefaction wave propagates continuously to the left within in a well-defined region, which grows with time. When the gas with low pressure is pushed by the gas of high pressure and density coming from the left, it generates a shock wave propagating to the right, where the two gases, the expanded and the compressed one, are separated by a contact discontinuity that travels at a constant velocity (Danaila *et al.*, 2007). Therefore, the contact discontinuity is a transition layer

which separates the fluids of significantly different densities and temperatures, but where the pressure is continuous and therefore there is no fluid flow across this surface (Norman and Winkler, 1985). With the Sod shock tube test, any hydrodynamic code can be tested for its accuracy and capability to resolve the shocks and contact discontinuities mentioned above. An evolution of the Sod shock tube simulation can be seen in Fig. 4.5. It shows the dynamics during the shock development and, furthermore, reveals its self-consistent character. It can be seen that the shape which can be seen at the end of our simulation already establishes at the beginning of the simulation and only spreads out with time. Moreover, it reveals that the first discontinuity which propagates to the right is not a shock, but a contact discontinuity, where the pressure is constant. From this evolution, the velocity at which the shock propagates can be determined as well and is about $v = 0.85$, from time $t = 0.025$ to $t = 0.2$. For this simulation, a ratio of specific heats of $\gamma = 1.4$ is used. The simulation is stopped at time $t = 0.2$ and also compared to Danaila *et al.* (2007), where a match of the solutions can be confirmed. A good resolution of the sharp change in the gradients can, for instance, be found with the OSPRE limiter with 1024 grid cells which is depicted in Fig. 4.6. Often, also a lower grid resolution is used in the literature (Danaila *et al.*, 2007). Overplotting the exact solution (solid line), allows a better understanding of the resolution in the shock regions and the accuracy of our numerical method. This also shows that the solution for the Sod shock tube test with the Minmod limiter and 1024 grid cells, as depicted in Fig. 4.7, does not resolve the sharp gradient (around $x = 0.675$) as well. This test is simulated with two different flux limiters, the OSPRE limiter and the Minmod limiter, and in five different resolutions: 128, 256, 512, 1024, and 2048 grid cells, to check the convergence of the code with each limiter. As previously discussed, a numerical method is said to be convergent if the numerical solution of the discrete expression approaches the exact solution of the differential equation as the grid size and time step go to zero (Kuzmin, 2010). Practically, convergence of a numerical scheme can be shown by running the same simulation with different resolutions, where this method also shows the global order of accuracy of the scheme used.

The convergence is investigated by calculating the numerical error (NE) for both limiters as a function of the resolution, to see how the limiters compare and it is plotted in

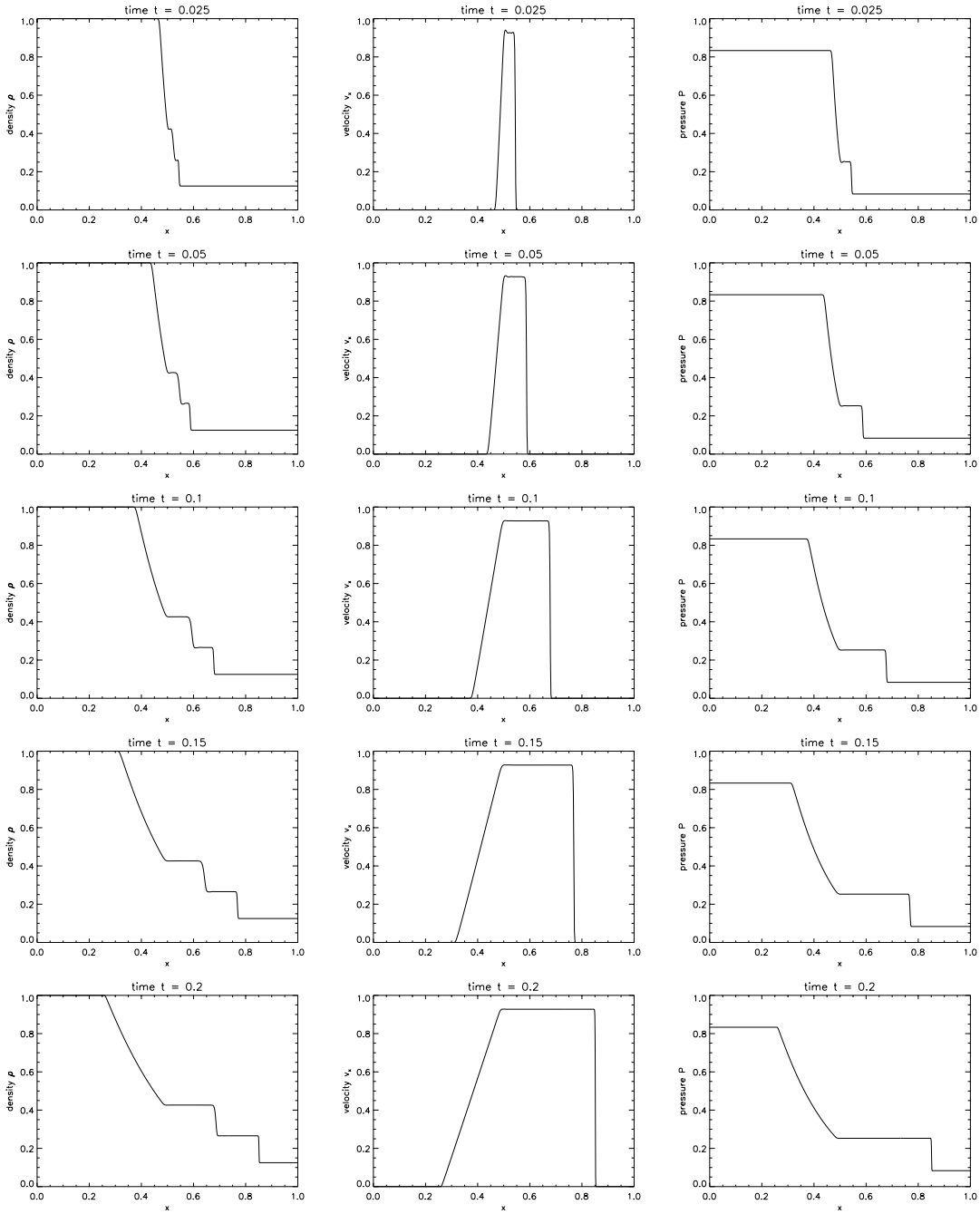

Figure 4.5: Evolution of the Sod shock tube simulation. From left to right, the density ρ, the velocity component v_x and the gas pressure P are displayed, from time $t = 0.025$ to $t = 0.1$ (top to bottom).

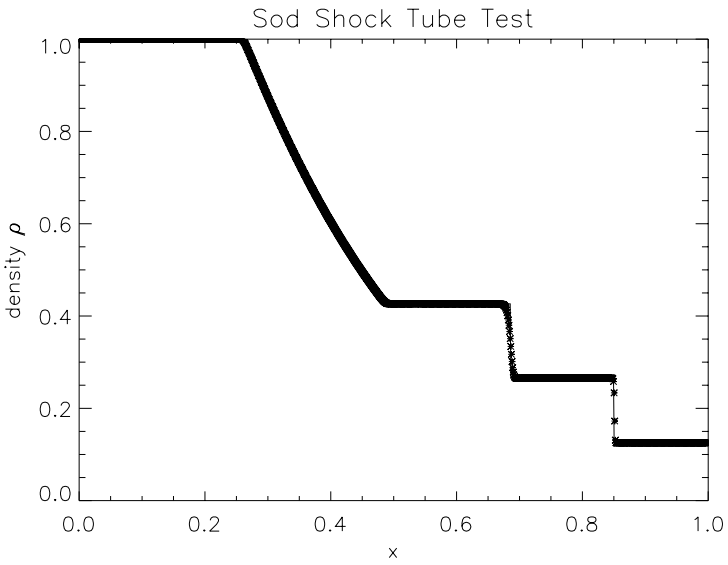

Figure 4.6: Exact solution of the Riemann problem (solid line) and the numerical solution simulated with the hydrodynamic code (asterisks), using the OSPRE flux limiter and a 1024 grid cell resolution.

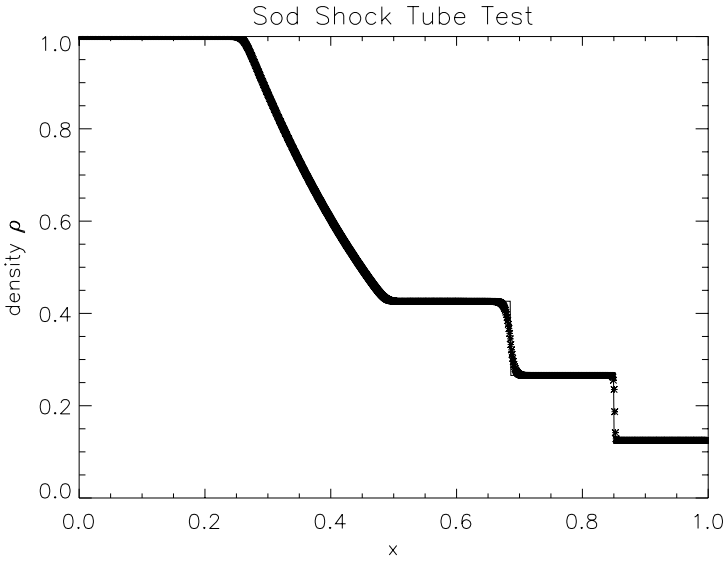

Figure 4.7: Exact solution of the Riemann problem (solid line) and the numerical solution simulated with the hydrodynamic code (asterisks), using the Minmod flux limiter and a 1024 grid cell resolution.

Fig. 4.8. The NE is calculated as follows:

$$NE = \frac{1}{N} \sum \frac{|\,(NU - A)\,|}{A}, \tag{4.5}$$

where NU represents the numerical solution, A the analytical solution and N the number of grid points in the x-direction. The summation is over all grid cells. Note that Fig. 4.8 is shown as a logarithmic plot, where the logarithm of the NE is plotted as a function of the number of grid points N. As demonstrated in Fig. 4.8, for a resolution of 1024 grid cells and with the OSPRE limiter, the exact solution is already replicated very well and for higher grid densities the accuracy does not improve significantly. With the Minmod limiter the same level of accuracy can be achieved, but at a higher grid resolution.

The black solid line shows the error for the OSPRE limiter and the red solid line the one for Minmod limiter at each grid size. For both error calculations, it can be seen that the errors for the OSPRE limiter are smaller than those for the Minmod limiter. However, the limiters converge similarly, where the overplotted linear functions (dashed for the Minmod limiter and dotted for the OSPRE limiter) reveal the gradient with which the convergence takes place, where the NE with the OSPRE limiter converges with a slope of $m = -0.76$ and with the Minmod limiter with a slope of $m = -0.80$. The dash-dotted line serves as a guide for the deviation from a linear function with a slope of $m = -1$.

To check the convergence of the error for the smooth parts of the function, the NE for the rarefaction wave part of the function (from $x = 0.29$ to $x = 0.43$ in Fig. 4.6) is investigated, which can be seen in Fig. 4.9. The numerical error decreases with an increasing number of grid cells, i.e. with smaller dx, and shows that the smaller the grid cells are, the more the computed solution approaches the analytical solution. For this shock-free region, it shows that in smoother regions the error converges faster than for the overall simulation (which include the shock regions), with an order of one, where the slope for the OSPRE limiter is $m = 0.99$ and the slope for the Minmod limiter is $m = 0.98$. Clearly, neither of the two limiters is fundamentally better here. It can be said that our hydrodynamic code replicates the analytical solution of the Sod-shock tube test with high accuracy, using any of the two limiters.

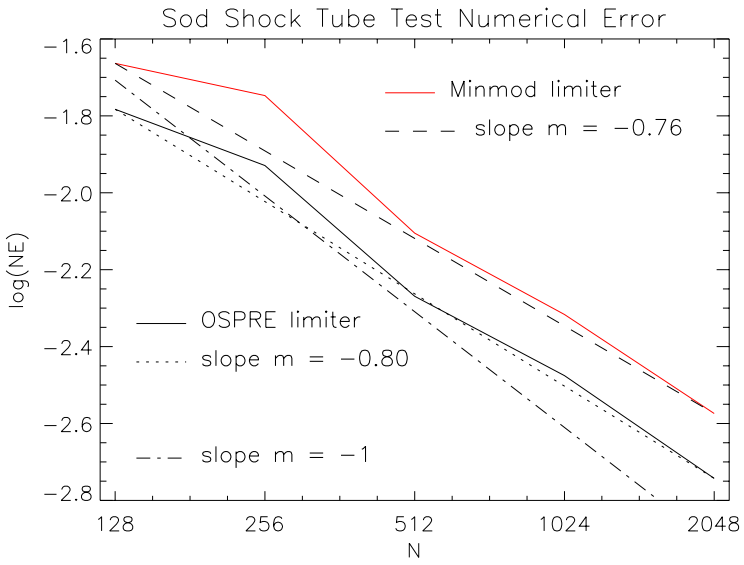

Figure 4.8: Logarithm of the Numerical Error (log(NE)) of the Sod shock tube test conducted with the OSPRE limiter (black solid line) and Minmod limiter (red solid line), plotted as a function of grid points (N). The dash-dotted line has a slope of -1 and serves as a guide for the OSPRE limiter's slope (dotted line) and the Minmod limiter's slope (dashed line).

4.3.2 Magnetohydrodynamic (MHD) Simulations

For 2D ideal MHD simulations, the vectors **U**, **F** and **G** are written as follows (Balsara, 2004):

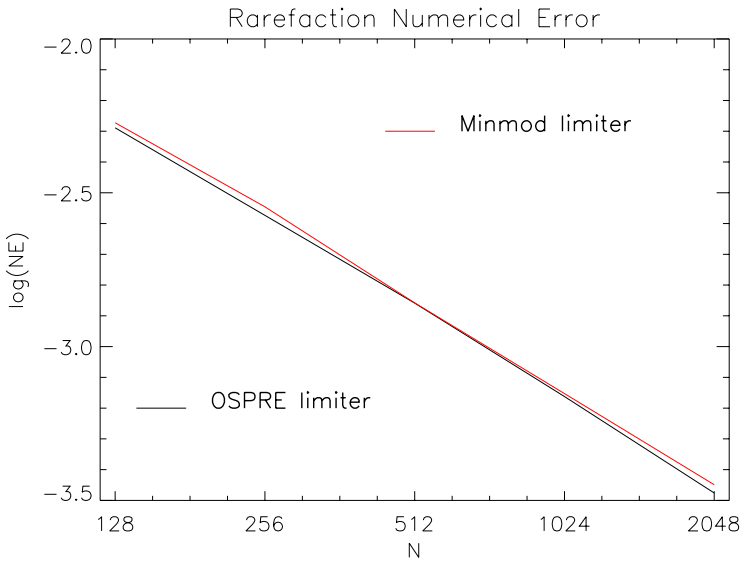

Figure 4.9: The logarithm of the Numerical Error (log(NE)) of the Sod shock tube test conducted with the OSPRE (black solid line) and Minmod limiter (red solid line) for the smooth rarefaction wave region.

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \epsilon \\ B_x \\ B_y \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \rho v_x \\ \rho v_x^2 + P + B^2/8\pi - B_x^2/4\pi \\ \rho v_x v_y - B_x B_y/4\pi \\ (\epsilon + P + B^2/8\pi) v_x - B_x(\mathbf{v}\cdot\mathbf{B})/4\pi \\ 0 \\ v_x B_y - v_y B_x \end{bmatrix},$$

$$\mathbf{G} = \begin{bmatrix} \rho v_y \\ \rho v_x v_y - B_x B_y/4\pi \\ \rho v_y^2 + P + B^2/8\pi - B_y^2/4\pi \\ (\epsilon + P + B^2/8\pi) v_y - B_y(\mathbf{v}\cdot\mathbf{B})/4\pi \\ v_x B_y - v_y B_x \\ 0 \end{bmatrix}. \tag{4.6}$$

This system essentially describes an electrically charged fluid, whose dynamics is in interaction with the electromagnetic field. Here, B_x and B_y are the x- and y-components of

the magnetic field, respectively. The total energy ϵ also includes the magnetic energy:

$$\epsilon = \rho v^2/2 + P/(\gamma - 1) + B^2/8\pi, \tag{4.7}$$

with $B^2 = B_x^2 + B_y^2$. The source term $\mathbf{S}$ may include resistivity, viscosity, gravity or a divergence cleaning term.

The 1D MHD code is tested with the Brio-Wu shock test, which is the MHD counterpart of the Sod shock tube test and is initialised with a discontinuity in the density and pressure. It provides initial conditions for the propagation of non-linear compressive waves and to test wave and shock capturing properties of an MHD solver. It involves two fast rarefaction waves, a slow compound wave, a contact discontinuity and a slow shock wave (Stone *et al.*, 1992). According to Stone *et al.* (1992), the convexity of the MHD equations implies that different modes of the same MHD wave family can propagate with the same velocities and, therefore, unlike in pure hydrodynamics, there can exist compound waves that consist of a shock wave attached to a rarefaction wave of the same family and this means that a slow shock can be attached to a slow rarefaction wave. Initial conditions are taken from Brio and Wu (1988):

$$\mathbf{U} = \begin{cases} \begin{bmatrix} \rho &=& 1 \\ P &=& 1 \\ v_x &=& 0 \\ v_y &=& 0 \\ B_x &=& 0.75 \\ B_y &=& 1 \end{bmatrix} & \text{for } x \leq 50, \\ \begin{bmatrix} \rho &=& 0.125 \\ P &=& 0.1 \\ v_x &=& 0 \\ v_y &=& 0 \\ B_x &=& 0.75 \\ B_y &=& -1 \end{bmatrix} & \text{for } x > 50. \end{cases}$$

This leads to an angle between the magnetic field lines and the shock normal of $\tan\theta = 4/3$. The evolution of the Brio-Wu shock simulation can be seen in Fig. 4.10. Like in the Sod shock tube test, this time evolution shows the dynamics during the shock development and reveals its self-consistent character. However, the shape which can be seen at the end of the simulation at time $t = 10$ only really establishes around time $t = 4$. Moreover, it reveals that the first discontinuity that propagates to the right is not a shock, but a contact discontinuity, where the pressure is constant. From this evolution, the velocity at which the shock propagates can be determined as well and is about $v = 1.37$, from time $t = 2$ to $t = 10$. The result is compared with the numerical solution in Stone *et al.* (1992), and we, therefore, choose the same physical dimensions ($x \in [0, 100]$) and specific heat ratio ($\gamma = 2$), and stop the simulation at time $t = 10$, which is presented in Fig. 4.11. The spatial dimension is normalised to 1 and the resolution is 1024×10 (x and y dimension). Our solution matches the results of the reference Stone *et al.* (1992). Again, the convergence for this simulation is studied and the numerical error is calculated for the two different limiters, as done in section 4.3.1 for the Sod shock tube test. Unlike in the Sod shock tube test case, however, there is no analytical solution. Therefore, a self-convergence test is performed, where the reference is our own numerical solution with a 2048 grid cell resolution. The NE convergence (according to Eqn. (4.3.1)) is depicted in Fig. 4.12. The red solid line is the NE for the Minmod limiter and the black solid line the one for the OSPRE limiter, where the dashed or dotted line shows the slope for each limiter, respectively. Again, the dash-dotted line serves as a guide for the deviation from a linear function with a slope of $m = -1$. We want to point out that for this error calculation and the resulting convergence plot, we cannot compare the two limiters or make conclusion about the superior limiter here, as the reference solution is not the same, because one reference solution was obtained using the OSPRE limiter and the other one using the Minmod limiter. However, assuming that for each of the limiters the high resolution solution is more accurate than the low resolution solution, we want to see how fast the solution converges as the grid cell density increases for each limiter. It can be observed that the slope of the linear function is less steep for the Minmod limiter, where $m = 1.026$, where for the OSPRE limiter the slope is $m = 1.105$, which means it moves faster to its high resolution answer. But, again, using any of the two limiters does not lead to fundamentally different results.

In 2D MHD, the well known OT vortex simulation (Orszag and Tang, 1979) shows the

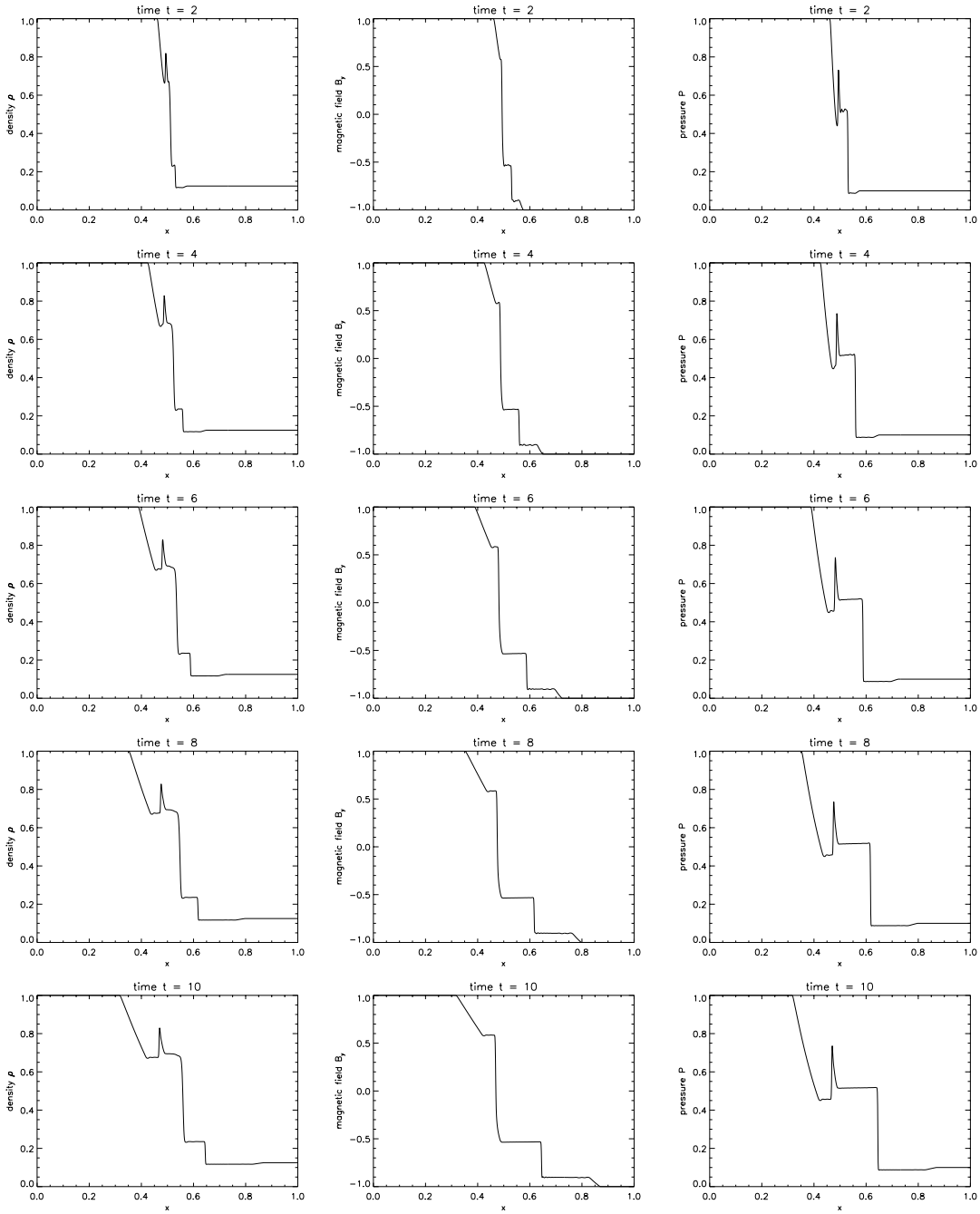

Figure 4.10: Evolution of the Brio-Wu shock. From left to right, the density ρ, the magnetic field component B_y and the gas pressure P are displayed, from time $t = 2$ to $t = 10$ (top to bottom).

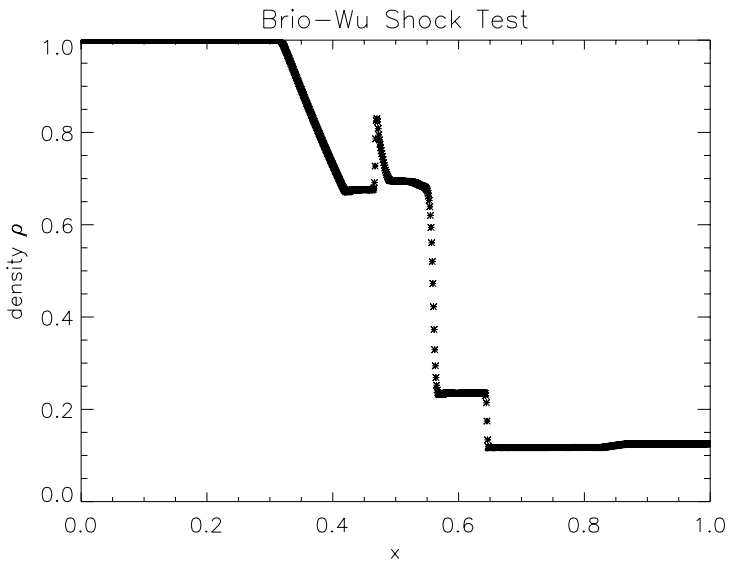

Figure 4.11: 1D MHD Brio-Wu shock test with the OSPRE limiter and a 1024 grid cell resolution.

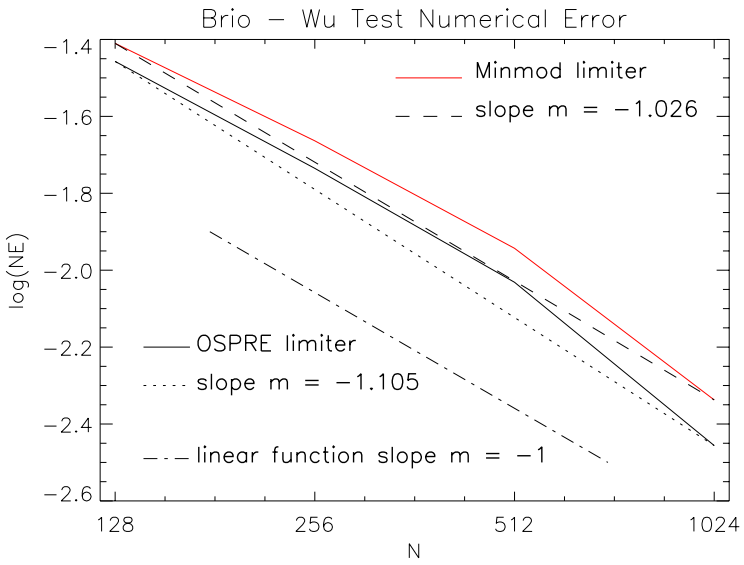

Figure 4.12: Logarithm of the Numerical Error (log(NE)) as a function of grid cell resolution (N) for the Brio-Wu shock test. This self-convergence test was conducted with the OSPRE limiter (black solid line) and with the Minmod limiter (red solid line, where the dotted line and the dashed line represents each slope, respectively. The dash-dotted line serves as a guide.

transition to supersonic 2D MHD turbulence and tests a code's robustness to MHD shock formation, shock-shock interactions and how well the $\nabla \cdot \mathbf{B} = 0$ constraint is satisfied.

Ryu *et al.* (1998) and Londrillo and Zanna (2000) present the OT vortex in a computational domain of 0 to 1, a resolution of 256×256 or grid cells, the specific heat ratio $\gamma = 5/3$ and the following initial conditions:

$$\mathbf{U} = \begin{bmatrix} \rho & = & \gamma P \\ P & = & \beta B_0^2/2 \\ v_x & = & -v_0 \sin(2\pi y) \\ v_y & = & v_0 \sin(2\pi x) \\ B_x & = & -B_0 \sin(2\pi y) \\ B_y & = & B_0 \sin(4\pi x) \end{bmatrix}.$$

Here, $B_0 = -1/\sqrt{4\pi}, \beta = 10/3, P = \beta B_0^2/2, v_0 = 1$. The initial conditions of this simulation are smooth and show a vortex profile in the velocity field and the magnetic field, which quickly forms shocks and turns into turbulence, due to non-linear interactions (Orszag and Tang, 1979). The boundary conditions are periodic everywhere.

The evolution of the OT vortex shock simulation can be seen in Fig. 4.13. This time evolution shows the dynamics during the vortex formation and the development of shock-shock interactions. The scale at which the evolution is presented is optimised and changes from time step to time step in order to enable the best possible presentation of the structures.

A replication of the OT vortex is shown in Fig. 4.14, bottom row. The top row shows a horizontal cut at $y = 0.428$ through the 2D plane and reveals the variation of the gas and magnetic pressure along this line, which allows an additional and better comparison with the reference plots. Comparing our solution (Fig. 4.14) to the reference solution Fig. 3 in Ryu *et al.* (1998), shows that the solutions match well, however, small wiggles near the boundaries, left ($x = 0.0$ to $x = 0.05$) and right ($x = 0.95$ to $x = 1$), can be observed in our solution. The difference between ours and their procedure is the implementation of a divergence cleaning scheme. The investigation of two difference divergence controlling methods are presented in the following section.

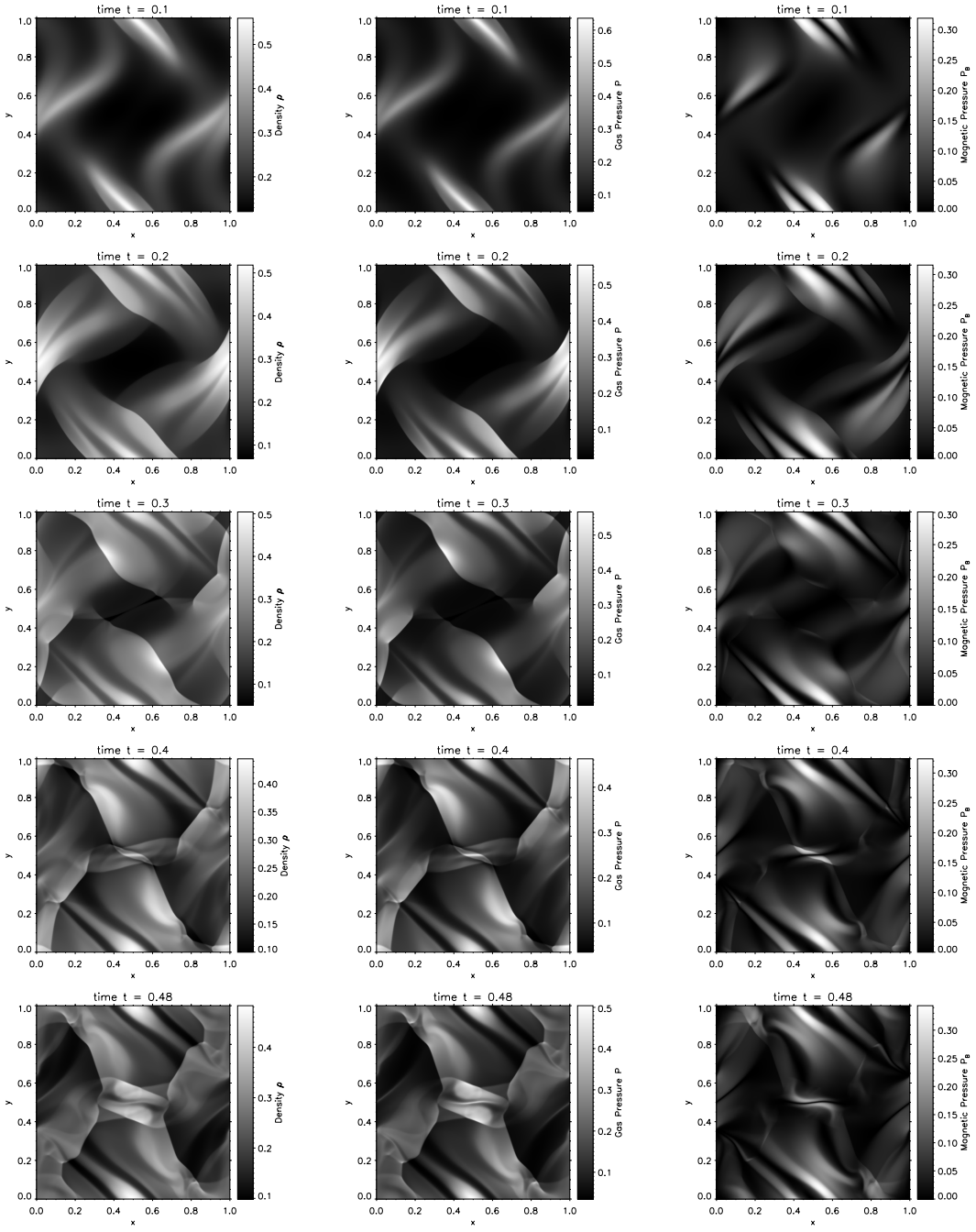

Figure 4.13: Evolution of the OT vortex. From left to right, the density, gas pressure and magnetic pressure are displayed, from time $t = 0.1$ to $t = 0.48$ (top to bottom).

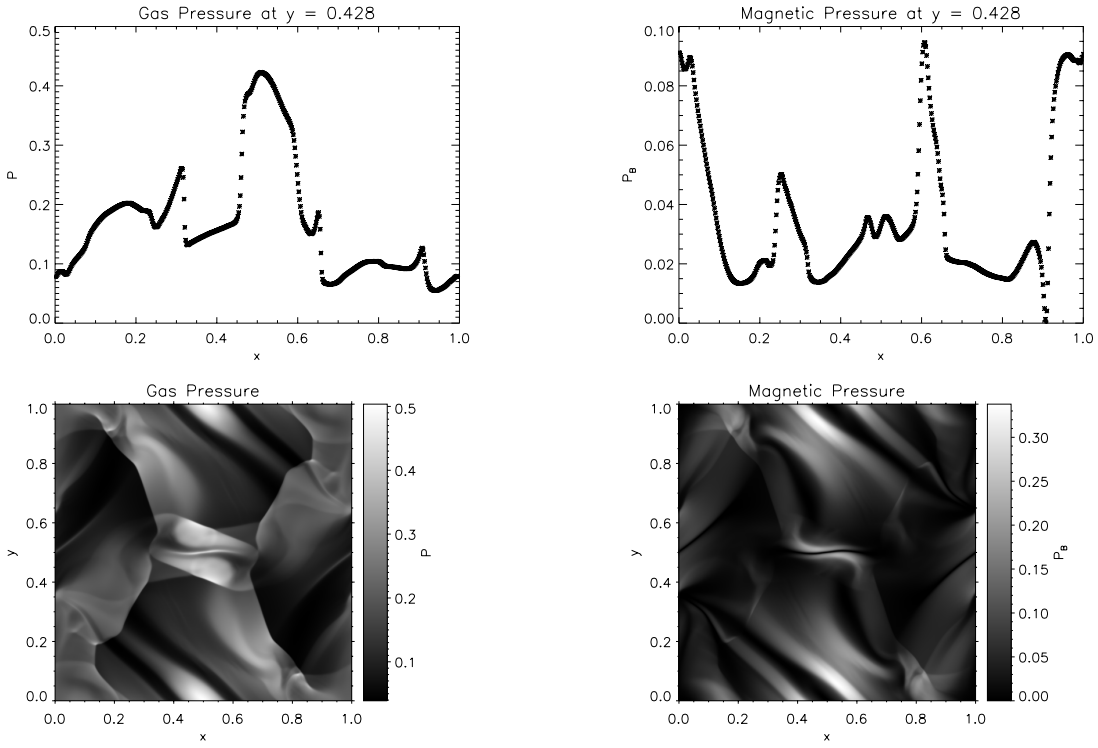

Figure 4.14: MHD Orszag-Tang vortex test for the gas pressure (left) and the magnetic pressure (right). The 1D plot (top row) shows a cut through the 2D plots (bottom row) at $y = 0.428$, where the solid line represents the variation of the gas and magnetic pressure and the asterisks are an overplot of the grid points to reveal the resolution.

4.3.3 Divergence Control

In a two-fluid plasma simulation, we deal with one fluid being embedded in a magnetic field. According to Maxwell's equations, the divergence of the magnetic field is zero. Numerically, this is not always assured due to truncation errors in the discretisation schemes (Guillet *et al.*, 2019). A non-zero numerical divergence can lead to non-physical plasma behaviour or non-physical perturbations to the flow, such as plasma acceleration along the magnetic field lines (Guillet *et al.*, 2019) or plasma flow orthogonal to the magnetic field (Balsara and Kim, 2003). There are various techniques to deal with the numerical non-zero divergence, where some involve the change of the discretisation scheme and others just an addition to the scheme.

A divergence-free magnetic field can, for example, be guaranteed by solving for the vec-

tor potential (Brandenburg and Dobler, 2002; Botha, Rucklidge, and Hurlburt, 2006) or by using a staggered grid (Arber *et al.*, 2001). In the following, two schemes are compared, which both involve an addition to the existing flux and source terms only, which means that the existing numerical scheme can be used as it is. These two divergence cleaning schemes are: the Powell source terms (Powell, 1994) and the hyperbolic divergence cleaning by Dedner *et al.* (2002). For the Powell scheme, a source term is added, which is proportional to $\nabla \cdot \mathbf{B}$ and has the following form:

$$\mathbf{S}_{Powell} = - \begin{bmatrix} 0 \\ B_x \\ B_y \\ \mathbf{v} \cdot \mathbf{B} \\ v_x \\ v_y \end{bmatrix} \nabla \cdot \mathbf{B}.$$

For the hyperbolic divergence cleaning, or the Extended Generalised Lagrange Multiplier formulation by Dedner *et al.* (2002), an additional scalar field Ψ is added to the system of equations, which is updated at every time step and added to the flux term of the induction equation and to the source term of the energy equation. Thus, the system of equations (4.6) becomes:

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \epsilon \\ B_x \\ B_y \\ \Psi \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \rho v_x \\ \rho v_x^2 + P + B^2/8\pi - B_x^2/4\pi \\ \rho v_x v_y - B_x B_y/4\pi \\ (\epsilon + P + B^2/8\pi) v_x - B_x(\mathbf{v}\cdot\mathbf{B})/4\pi \\ \Psi \\ v_x B_y - v_y B_x + \Psi \\ c_h^2 \nabla\cdot\mathbf{B} \end{bmatrix},$$

$$\mathbf{G} = \begin{bmatrix} \rho v_y \\ \rho v_x v_y - B_x B_y/4\pi \\ \rho v_y^2 + P + B^2/8\pi - B_y^2/4\pi \\ (\epsilon + P + B^2/8\pi) v_y - B_y(\mathbf{v}\cdot\mathbf{B})/4\pi \\ v_x B_y - v_y B_x + \Psi \\ \Psi \\ c_h^2 \nabla\cdot\mathbf{B} \end{bmatrix}, \tag{4.8}$$

and the source term becomes:

$$\mathbf{S}_{EGLM} = \begin{bmatrix} 0 \\ -(\nabla\cdot\mathbf{B})B_x \\ -(\nabla\cdot\mathbf{B})B_y \\ -B_x(\nabla\Psi) \\ -B_y(\nabla\Psi) \\ 0 \\ -(c_h^2/c_p^2)\Psi \end{bmatrix}.$$

Here,

$$c_h = sf\left(C_{CFL}\,\frac{dx}{dt}\right) \quad \text{and} \quad c_p = 0.18\,c_h \tag{4.9}$$

with the Courant-Friedrich-Lewy coefficient $C_{CFL} = 0.4$, to ensure that c_h is smaller than the fastest propagation speed, so that the time step is not influenced by the speed of which this information is carried away. c_h is the finite speed at which divergence errors are propagated to the boundary (Dedner *et al.*, 2002) and was chosen in terms of the smallest

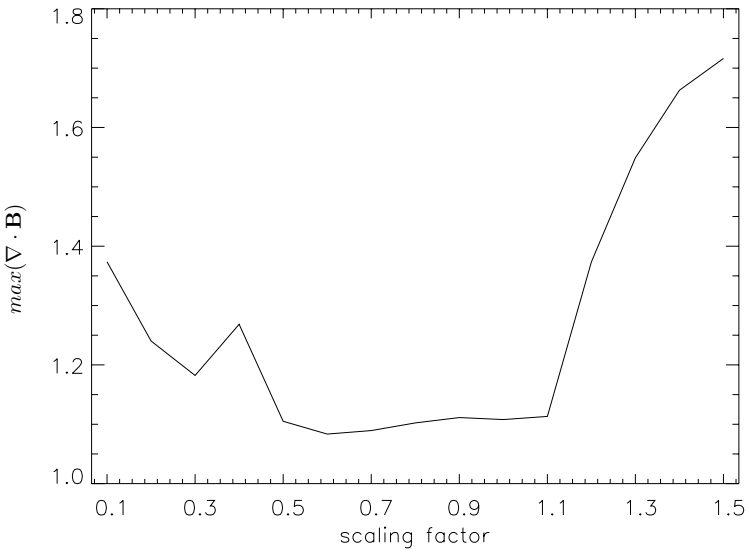

Figure 4.15: Maximum of $\nabla \cdot \mathbf{B}$ as a function of the scaling factor *sf* in Eqn. (4.9), for the OT vortex simulation at time $t = 0.48$.

maximum $\nabla \cdot \mathbf{B}$ that is yielded in test runs of the OT vortex. Fig. 4.15 shows the maximum of $\nabla \cdot \mathbf{B}$ as a function of the scaling factor sf for the OT vortex simulation at time $t = 0.48$. In this plot, the divergence (vertical axis), does not have units as the equations have been made dimensionless. It shows that the magnetic divergence is smallest when $sf = 0.6$.

In the OT vortex simulation divergence issues will result in distortions or noise in smooth post-shock regions of the flow (Guillet *et al.*, 2019). We run the OT vortex simulation with each of the divergence cleaning schemes in ideal MHD and plot the divergence of the magnetic field (Fig. 4.16) following Guillet *et al.* (2019), where $\nabla \cdot \mathbf{B}$ in cell K of volume V_K is defined as:

$$| \nabla \cdot \mathbf{B} |_K = \frac{1}{V_K} \int_K | \nabla \cdot \mathbf{B} | \, d\mathbf{x} + \frac{1}{V_K} \int_{\delta K} | B_F^n - B^n | \, dS \tag{4.10}$$

where $| \nabla \cdot \mathbf{B} |$ is the L_1 norm of the divergence of the magnetic field, $B^n = \mathbf{B} \cdot \mathbf{n}$, where $\mathbf{n}$ is the face normal vector and B_F^n is the normal component of the magnetic field at the face and defined as:

$$B_F^n = \frac{B_L^n \sqrt{\rho_L}^{-1} + B_R^n \sqrt{\rho_R}^{-1}}{\sqrt{\rho_L}^{-1} + \sqrt{\rho_R}^{-1}}, \tag{4.11}$$

where B_L^n and B_R^n are the normal component of the magnetic field *left* and *right* of the face, respectively. The same applies for ρ_R and ρ_L, which are the mass densities *left* and *right* of the face. In order to obtain Fig. 4.17, which shows the time series of the OT vortex with the two different divergence controlling methods, the L_1 norm of the global magnetic field divergence is calculated, which is given as:

$$\|\nabla \cdot \mathbf{B}\|_1 = \frac{1}{V} \sum_K V_K \mid \nabla \cdot \mathbf{B} \mid_K, \tag{4.12}$$

where V is the volume ($dxdy$). With the definition of the normalised divergence, which is:

$$divB_{norm} = \frac{\mid \nabla \cdot B \mid_K}{\mid B \mid_K} \Delta x, \tag{4.13}$$

where $\mid B \mid_K$ is the modulus of B at cell K, the global divergence of the normalised divergence is obtained:

$$\|divB_{norm}\|_1 = \frac{1}{V} \sum V_K \mid divB_{norm} \mid . \tag{4.14}$$

In Fig. 4.16, the normalised divergence of the magnetic field (Eqn. (4.13)) is plotted at time $t = 0.48$. It can be seen that with the Dedner divergence cleaning method, the regions in between the shock-shock interactions are smoother (at around $x = 0.3$ to $x = 0.7$ and $y = 0.7$ to $y = 0.9$), compared to Powell source terms and without any additional divergence control. The Powell approach is based on a non-conservative addition to the source term, which leads to an additional wave, which advects away the divergence with the flow, whereas the approach by Dedner dampens the divergence and dynamically advects with help of an additional scalar field. The advantage of the Powell scheme is its easy implementation to an already existing scheme without having to scale any free parameters like in the Dedner scheme. However, the Powell scheme only ensures that the magnetic field is locally divergence free, inside a cell but not globally divergence free, as the normal component of the magnetic field is not guaranteed to be continuous across cell interfaces (Guillet *et al.*, 2019). The numerical divergence is mainly concentrated around the shocks, as visible in all plots of Fig. 4.16. However, the scheme by (Dedner *et al.*, 2002) advects the divergence away quickly, which results in a more uniform background. The Powell method however, does not deal with strong shocks or interacting shocks well, as it does not advect the divergence away quickly enough and, therefore does not per-

form well for the OT vortex simulation without the adjustment of further parameters in the code (Guillet *et al.*, 2019). Furthermore, the Powell scheme, in fact, only advects the divergence with the flow and does not eliminate it, which is why it can cause local accumulation of divergence. Moreover, whenever there is numerical divergence errors, it will locally inject conserved quantities and, therefore, does not ensure conservation anymore. Latter, however, only has a small impact according to Guillet *et al.* (2019), but should nevertheless be checked and considered when using Powell.

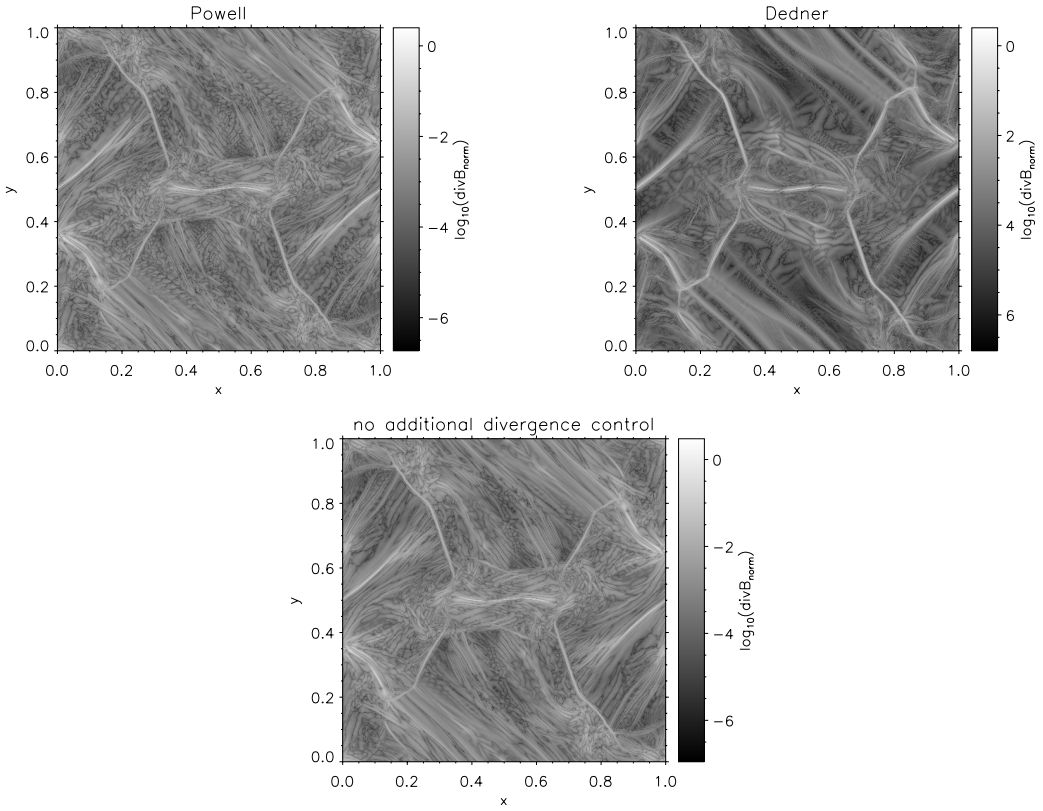

Figure 4.16: Orszag-Tang vortex simulation at time $t = 0.48$. With Powell source terms, Dedner's hyperbolic divergence cleaning, and without any additional divergence control.

As mentioned before, Fig. 4.17 shows the global divergence (Eqn. (4.12)) as a function of time with the two divergence control methods, where the solid line represents the time series with the Dedner terms and the dotted line with the Powell terms. The OT vortex is simulated up to time $t = 5$ and the plot shows that the divergence increases only at the beginning of the simulation and then stays roughly constant for the Powell scheme

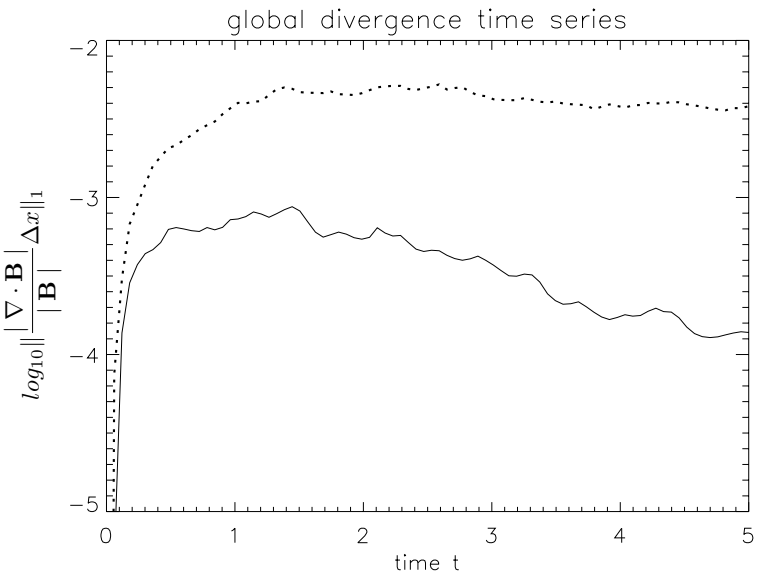

Figure 4.17: Plot of the logarithm of the normalised divergence of the magnetic field as a function of time for the Orszag-Tang vortex up to time $t = 5$. The solid line is the divergence with the hyperbolic divergence cleaning and the dotted line with the Powell source terms.

and decreases for the Dedner scheme. Based on Fig. 4.17, it can be said that the Dedner approach yields a smaller global divergence compared to Powell and, therefore, we will be using Dedner for the remaining simulations in this thesis.

The OT vortex is simulated again, this time with the divergence cleaning method of Dedner *et al.* (2002) and present our plots in Fig. 4.18. Again, the top row shows a horizontal cut at $y = 0.428$ through the 2D plane and reveals the variation of the gas and magnetic pressure along this line. The main difference now, compared to Fig. 4.14 (no divergence cleaning), is the overall smoothness, especially visible in the 1D magnetic pressure profile close to the boundaries, similar to Ryu *et al.* (1998). Before, in Fig. 4.14, small wiggles can be observed close to the *left* and *right* boundaries of the 1D profile of the magnetic pressure, which disappear with the implementation of the divergence controlling scheme and now our solution shows that a very good agreement with Fig. 3 in Ryu *et al.* (1998) is achieved.

In order to test the scaling of the code relative to the number of CPUs, the OT vortex problem with a 512×512 grid and stopped at time $t = 0.48$ is run on Oswald (Northum-

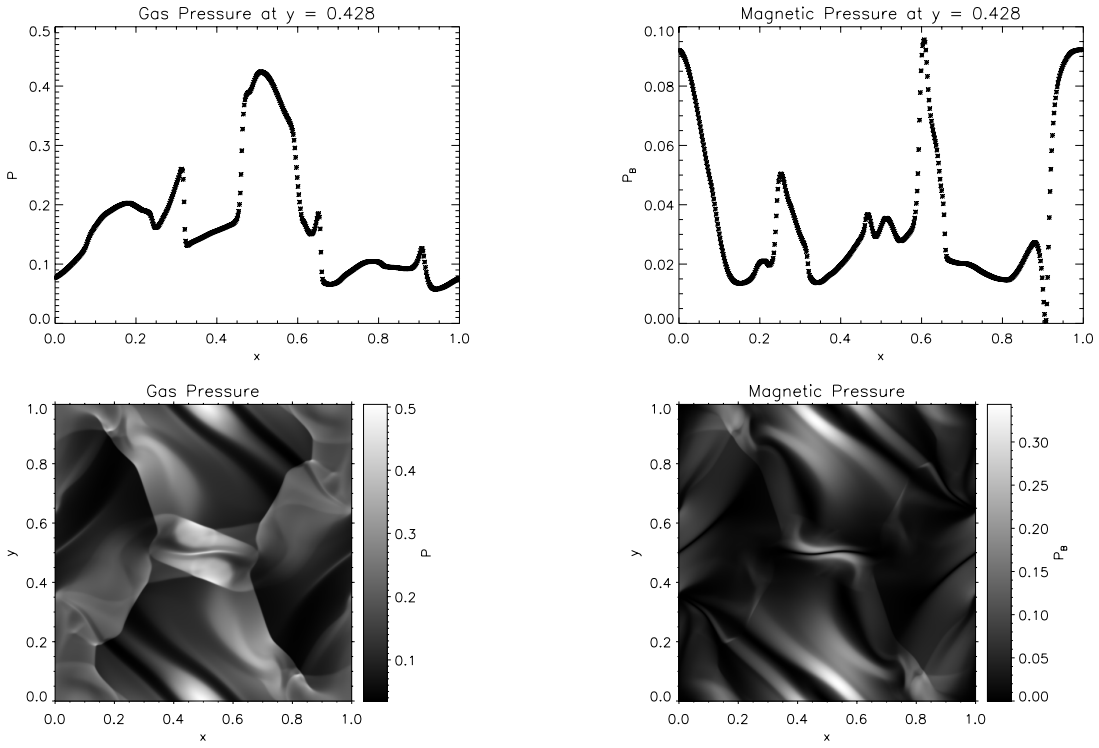

Figure 4.18: MHD Orszag-Tang vortex test with Dedner's hyperbolic divergence cleaning for the gas pressure (left) and the magnetic pressure (right). The 1D plot (top row) shows a cut through the 2D plots (bottom row) at $y = 0.428$.

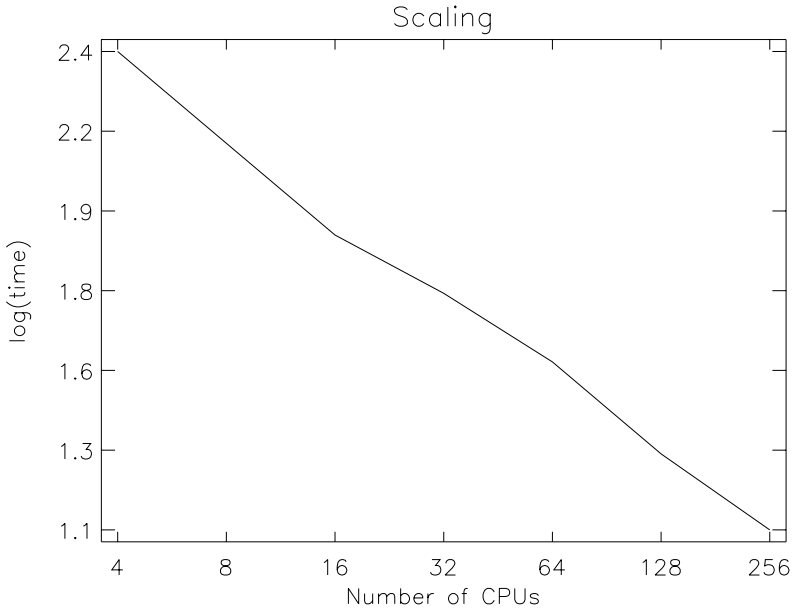

Figure 4.19: Scaling results of the code with a fixed problem setup (2D two-fluid OT vortex) and a varying number of CPUs.

bria's High Performace Computer (HPC)) with varying numbers of CPUs, ranging from 4 to 256 CPUs. This scaling is shown in Fig. 4.19, where the y-axis shows the logarithmic values of the real time needed for this simulation. For this test and the chosen number of CPUs, the performance of the code scales as

$$\text{time} = 10^{2.9}\,(\text{number of CPUs})^{-0.75}.$$

Chapter 5

Two-Fluid MHD Code and Simulations

In the previous chapters, the hydrodynamic and MHD equations were introduced (Chapter 2), the numerical implementation and set-up were presented and the code verification was demonstrated (Chapter 4). The partial ionisation effects captured with our two-fluid MHD code and the simulations of partially ionised plasmas are presented in the following.

Firstly, for these simulations, the model has been non-dimensionalised. The advantage of making equations non-dimensional and, therefore, unit-less is that it simplifies the problem formulation and can also reduces numerical round-off errors (Danaila *et al.*, 2007, p. 218). Moreover, it can reduce the number of parameters involved or allows the better comparison of parameters (Yunus A. Cengel, 2002, p. 356). This non-dimensionalisation or scaling is done with help of a reference parameter, which can be a characteristic scale or a physical quantity. In general, the benefit of non-dimensionalising in terms of units of a physical quantity that makes sense for the problem under study, is that the numbers we yield from running the code, immediately has physical meaning. In MHD this could be a non-dimensionalisation in terms of the Alfvén velocity or the sound speed for example. In this work, a non-dimensionalisation related to the Alfvén velocity is used, following Hillier, Takasao, and Nakamura (2016). The choice for the non-dimensionalisation depends on the subject under study and often a few choices make physical sense. For example, in Hillier (2019), the system has been non-dimensionalised in terms of the sound speed. Non-dimensionalisation works as follows. Basically, we say that the quantity in the "real" world q_{real}, with dimensions and units, equals the quan-

tity, i.e. the number, in the code q_{code} times some scaling factor $q_{scaling}$, therefore, as an example $q_{real} = q_{code} * q_{scaling}$. Therefore, the non-dimensionalised quantity in the code $q_{code} = q_{real}/q_{scaling}$. This means that the number the code yields, has to be multiplied by the scaling factor to obtain the physical quantity with dimensions and units. This scaling factor has units and a certain size. If we want to normalise our quantity, this size would be a set maximum of the quantity. For the simulations in this chapter, the velocity is non-dimensionalised in terms of the Alfvén velocity v_A, where the bulk Alfvén velocity is normalised to 1 and the collision frequency is 1, too. The density ρ is non-dimensionalised by the total density ρ_{tot}, which is also normalised to 1. The time t is given in terms of the inverse of the collision frequency and, therefore, it is $t = 1/\alpha_c(T_0)\rho_{tot}$. And since the velocity $v = L/t$ or $L = v_A t$, the previous non-dimensionalisation leads to the non-dimensionalisation of the length scale with $L = v_A/(\alpha_c(T_0)\rho_{tot})$. This means that after the fluid has traveled through one unit length, it is fully coupled. Because the total density and the collision frequency are also normalised to 1, it follows from this normalisation that $\alpha_c(T_0) = 1$. The magnetic field is normalised by $B = B_{dim}/\sqrt{4\pi}$, or, in terms of the Alfvén velocity $B = v_A\sqrt{\rho_{tot}}$. The magnetic pressure becomes $P_B = B^2/2$ and the magnetic energy term becomes $E_B = B^2$. If there was no magnetic field, i.e. $B = 0$, the non-dimensionalisation can also be done through the sound speed, or if the collision frequency is zero, the time could also be non-dimensionalised in terms of the characteristic length and time scales. The temperature is calculated with the ideal gas law and in non-dimensionalised form it is given with $T_n = P_n\gamma_n/\rho_n$ and $T_p = P_p\gamma_p/2\rho_p$, for the neutral and ionised fluid, respectively.

5.1 Two-Fluid Simulations

The two sets of hydrodynamic and MHD equations are numerically solved and the fluids are coupled through collisions, which is mathematically expressed with a collision frequency term in the source term, Eqn. (2.142). First, for testing purposes, the two fluids were decoupled by setting the collision rate to zero and each of the fluids are simulated separately in the two-fluid code. Ionisation and recombination rates, γ_{ion} and γ_{rec}, Eqns. (2.144) and (2.145), respectively, can be turned on to investigate effects of ionisation and recombination on the fluid structure in the simulation. With the two-fluid code, the 1.5D slow-mode MHD shock as initialised in Hillier, Takasao, and Nakamura (2016) (with col-

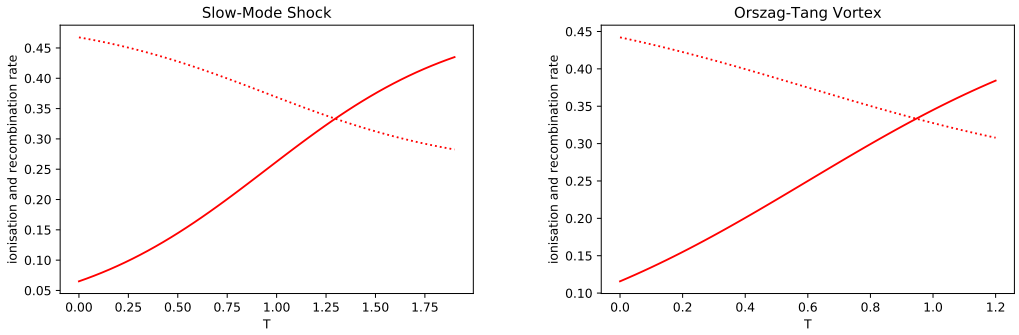

Figure 5.1: The ionisation rate (solid line) and recombination rate (dotted line), according to Eqn. (2.144) and Eqn. (2.145), respectively, for the slow-mode shock by Hillier, Takasao, and Nakamura (2016) *left*, where $T_{max} = 1.9$ and for the OT vortex *right*, where $T_{max} = 1.2$.

lision terms only) is replicated to secure that also this two-fluid configuration of the code is reliable and to use this simulation as a reference simulation. Additionally, γ_{ion} and γ_{rec} are activated and the effects of ionisation and recombination on this shock are investigated. In 2D, the two-fluid Orszag-Tang vortex is initiated, first with collision frequency rates only and then with ionisation and recombination rates included.

The effect of γ_{ion} and γ_{rec} on the fluids, shock properties and the development of the energy conversions throughout the simulation are investigated. As described before, the momentum and energy variations, which result from changes in the ionised and neutral fractions, are taken into account through the terms containing γ_{ion} and γ_{rec}. The ionisation fraction can vary through γ_{ion} and γ_{rec} terms in the continuity equations. Because the calculation of the ionisation and recombination rates require a maximum temperature to adjust those rates to the system under study and maximise their effect, this maximum temperature varies from problem to problem. Here, the maximum temperatures are obtained from simulations without ionisation and recombination included. In Fig. 5.1, the relation of ionisation and recombination is illustrated for both of the simulations that are studied.

The plots show the ionisation and recombination cross-over for the slow-mode shock (left) and for the OT vortex (right) as a function of temperature. It reveals that ionisation and recombination both occur in both our simulation cases and at what temperatures ionisation dominates and when recombination dominates.

The ambient temperature T_0 is calculated with $T_0 = P/\rho R_g$, with $R_g = 0.56$ being the gas constant, expressed in terms of the normalisation in Hillier, Takasao, and Nakamura (2016). The gas pressure and density are $P = P_n + P_p$ and $\rho = \rho_n + \rho_p$, respectively. The temperature is computed using $T_n = P_n\gamma_n/\rho_n$ and $T_p = P_p\gamma_p/2\rho_p$. A specific heat ratio of $\gamma = 5/3$ and the Minmod limiter are used. Like in Hillier, Takasao, and Nakamura (2016), the dynamic timescales are assumed to be smaller than collisional timescales and, therefore, a simple numerical scheme for the time integration can be applied.

5.1.1 1.5D Slow-Mode Shock

The solar corona is very hot, which means there is a lot of energy per unit mass. This energy might come from magnetic reconnection, as this can release a lot of stored magnetic energy (Hillier, Takasao, and Nakamura, 2016). In magnetic reconnection there are two important processes that release this energy, or in other words, convert magnetic energy to fluid energy (kinetic, thermal): one is the Joule heating and the second is the work of magnetic field on the plasma, post-reconnection. Fast magnetic reconnection is a result of standing slow-mode shock created by magnetic field relaxation (Hillier, Takasao, and Nakamura, 2016). Hence, slow-mode shocks are of interest because they are the source of fast magnetic reconnection and, therefore, provide the mechanism for converting magnetic energy to fluid energy, i.e. the heating of the system.
In MHD, the three characteristic wave speeds, namely slow, Alfvén and fast, lead to a variety of shock transitions (Snow and Hillier, 2019). The slow-mode or switch off shock is the transition from super-slow to sub-slow flow speeds. Hillier, Takasao, and Nakamura (2016) investigated a 1.5D slow mode or switch-off shock and its evolution in time in a partially ionised plasma and a two-fluid setting. In this setting, the initial conditions show a constant pressure and density background, but a discontinuity in the vertical component of the magnetic field. Note that Hillier, Takasao, and Nakamura (2016) set ionisation and recombination rates to zero. We replicated the 1.5D two-fluid shock in Hillier, Takasao, and Nakamura (2016) to demonstrate the physical reliability of the two-fluid code and use it as a reference solution in order to investigate the effects of the inclusion of γ_{ion} and γ_{rec}. The same initial conditions are used as in Hillier, Takasao, and Nakamura (2016) and are:

$$\begin{bmatrix} \rho_p &= & \zeta_1\rho_{tot} \\ P_p &= & 2\zeta_1/(\zeta_2+2\zeta_1)P_{tot} \\ v_{xp} &= & 0 \\ v_{yp} &= & 0 \\ B_x &= & 0.3B_0 \\ \rho_n &= & \zeta_2\rho_{tot} \\ P_n &= & \zeta_2/(\zeta_2+2\zeta_1)P_{tot} \\ v_{xn} &= & 0 \\ v_{yn} &= & 0 \end{bmatrix},$$

$$B_y = \begin{cases} -B_0, & \text{if } x > 1, \\ B_0, & \text{if } x \leq 1. \end{cases}$$

$P_{tot} = 0.15$ is the total gas pressure and $B_0 = 1$ is the magnetic field. Initial charged and neutral fluid fractions for the density are $\zeta_1 = 0.1$ and $\zeta_2 = 0.9$, respectively. Continuous boundary conditions are applied. The simulation is stopped at time $t = 1$. Our numerical result is presented in Fig. 5.3 and matches Fig. 2 in Hillier, Takasao, and Nakamura (2016). A self-convergence test is performed, with the equation given in Eqn. (4.3.1) and displayed in Fig. 5.2, to test the code's convergence for the shock simulation by Hillier, Takasao, and Nakamura (2016). It can be said that the NE converges with a slope of order 1, similar to the Sod shock tube and Brio-Wu shock test in Sec. 4.3.1 and 4.3.2, respectively.

Due to the initial conditions, the development of the ionised fluid dominates the dynamics of this slow-mode shock evolution, as everything is in equilibrium, except the magnetic field. The discontinuity in the vertical magnetic field component influences the ionised fluid directly and influences the neutral fluid only through collisions of the neutral fluid with the ionised fluid. A fast-mode rarefaction wave and a slow-mode shock form in the ionised fluid. The rarefaction wave is visible in the plots of the B_y, v_x and v_y components, between $x = 2.7$ and $x = 3.4$, in Fig. 5.3 and travels away from the initial position of the discontinuity in the magnetic field at approximately the ion Alfvén speed (Hillier, Takasao, and Nakamura, 2016). Behind the rarefaction wavefront, the ionised fluid moves at a speed of 0.4 in the negative horizontal direction towards the slow-mode shock, while the velocity of the neutral fluid is close to zero. This results in a drift velocity, $v_D = v_n - v_p$,

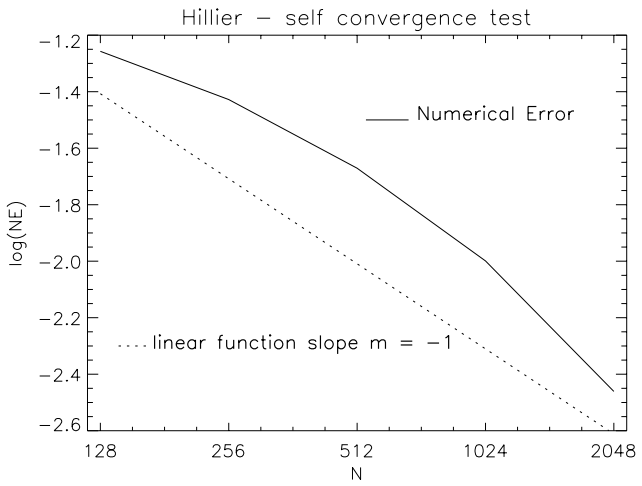

Figure 5.2: Self convergence test for Hillier shock test with the Minmod limiter. The logarithm of the Numerical Error (log(NE)) for the density as a function of grid cell resolution (N) is plotted with the solid line, where our reference is our solution at a 4096 grid cell resolution. The dotted line is a reference slope.

and comes into play in the source terms, Eqn. (2.134), of the momentum equations. From Eqn. (2.134), it can also be seen that this difference in speeds acts as an energy source for the neutral fluid, while it extracts energy from the ionised fluid. Hillier, Takasao, and Nakamura (2016) show that this results in a higher local temperature for the neutral fluid at the slow-mode wavefront, as revealed by the temperature plot in Fig. 5.3. This results in a Sedov-Taylor-like expansion of the neutral fluid, also known as a blast wave, which manifests in the enhanced neutral gas pressure as well as the separation of the slow-mode shock fronts in the neutral and ionised fluids. Furthermore, it leads to a depletion of the neutral density around $x = 0$.

The simulation of the slow-mode shock is repeated with the inclusion of γ_{ion} and γ_{rec}, Eqns. (2.144) and (2.145). Figure 5.4 shows the result with ionisation rate (γ_{ion}) and recombination rate (γ_{rec}) and is compared with Fig. 5.3, which is without γ_{ion} and γ_{rec}, where both figures show the shock structure at time $t = 1$. Figure 5.5, shows the ratio (without γ_{ion} and γ_{rec} / with γ_{ion} and γ_{rec}) for each quantity.

Starting the comparison with the temperature plots, it can be observed that the temperature profile changes with the inclusion of γ_{ion} and γ_{rec}. With the collision term only, i.e. without γ_{ion} and γ_{rec}, the temperatures of both fluids are similar except at the slow-mode

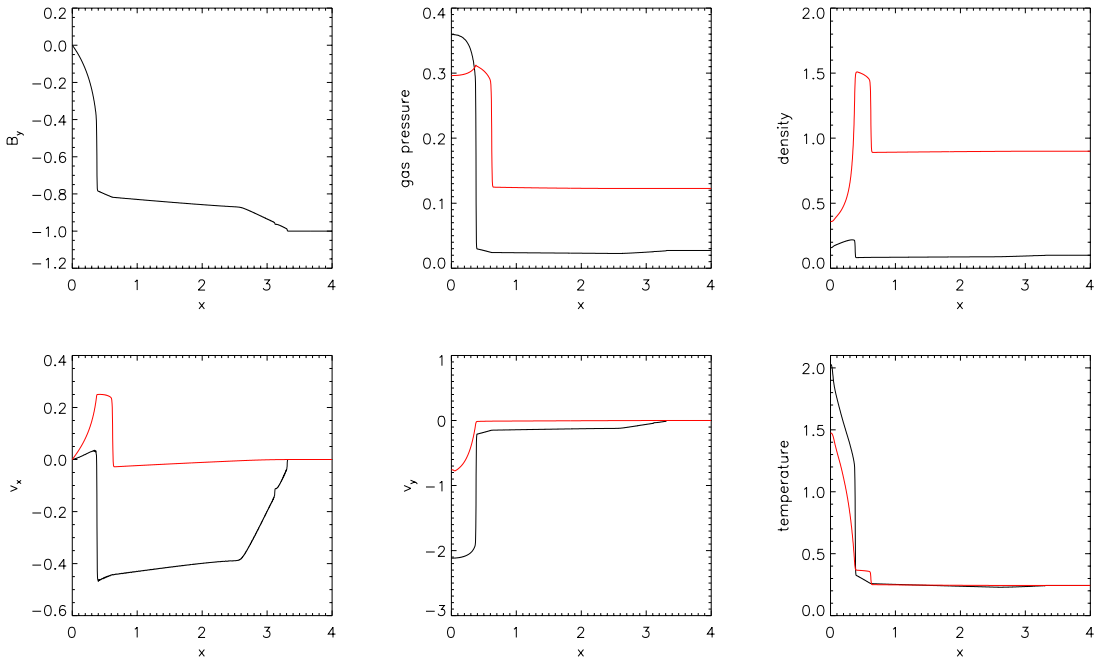

Figure 5.3: Replication of Hillier, Takasao, and Nakamura (2016) slow mode shock. From left to right are presented the spatial distribution of the vertical magnetic field component B_y, gas pressure, mass density, horizontal velocity component v_x, vertical velocity component v_y, and the temperature for the neutral (red) and ionised (black) fluid at time $t = 1$.

shock front, where $T_p > T_n$ by a factor of 0.7. In contrast, with γ_{ion} and γ_{rec} the temperatures are the same at the shock front and elsewhere $T_p < T_n$ by a factor of 0.6. This is because γ_{ion} and γ_{rec} influence the densities of the fluids and act as sinks and sources, which affects their temperature. The temperature dependence of γ_{ion} and γ_{rec} are given in Eqns. (2.144) and (2.145), and illustrated Fig. 5.1, left for this slow-mode shock simulation. For example, for the initial ambient neutral temperature $T_n = 0.25$, the $\gamma_{ion} = 0.1$ and $\gamma_{rec} = 0.45$. The expressions for the source terms, Eqn. (2.134), show that this leads to a source in the plasma continuity equation and a sink in the neutral continuity equation. In addition to the temperature differences, these terms cause a change in the mass density and gas pressure of both the ionised and neutral fluid. For the mass densities, ρ_p increases by a factor of 2.2 and ρ_n decreases by a factor of 0.8. For the gas pressure, both P_p and P_n decrease by one order of magnitude with the inclusion of γ_{ion} and γ_{rec}.

The source terms in the momentum equations, Eqn. (2.134), show that γ_{ion} and γ_{rec}

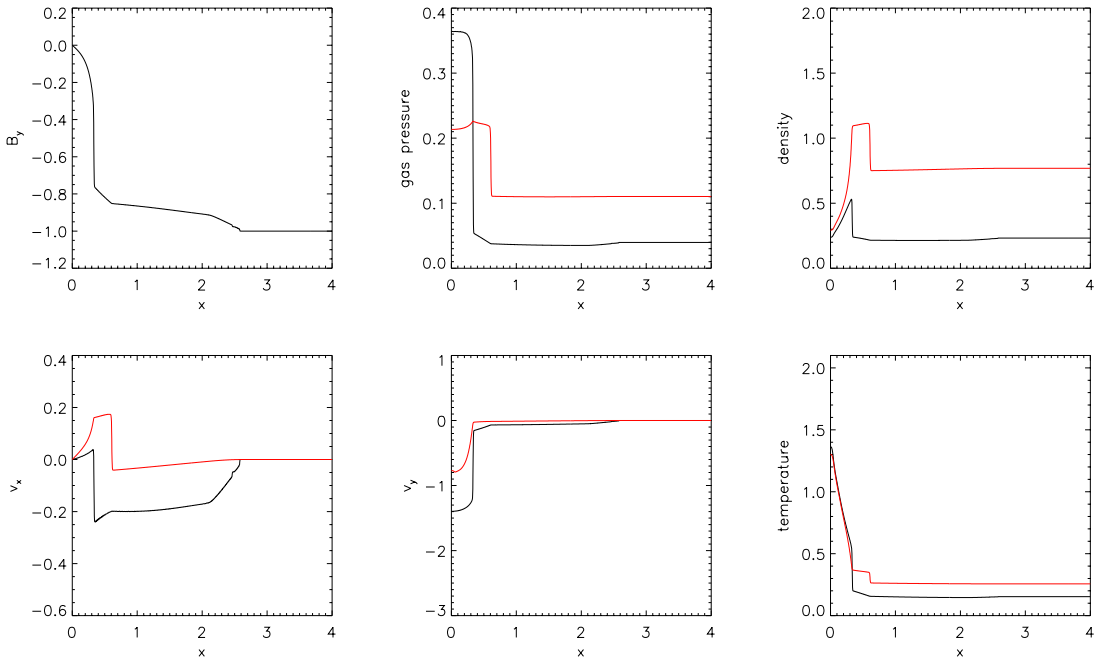

Figure 5.4: Hillier slow mode shock with ionisation and recombination rates, Eqns. (2.144) and (2.145), included. From left to right are presented the spatial distribution of the vertical magnetic field component B_y, gas pressure, mass density, horizontal velocity component v_x, vertical velocity component v_y, and the temperature for the neutral (red) and ionised (black) fluid at time $t = 1$.

come into effect when the velocities are non-zero. From Fig. 5.3 and Fig. 5.4, it can be seen that v_x and v_y are zero only where the rarefaction wave and slow-mode shock have not affected the background plasma. This means that γ_{ion} and γ_{rec} influence the physics only in the region between the rarefaction wavefront and the slow-mode shock.

In the magnetic field plot, it can be seen that the rarefaction wavefront without γ_{ion} and γ_{rec}, as in Fig. 5.3, has a finite width of 0.7, from $x = 2.7$ to $x = 3.4$. With γ_{ion} and γ_{rec}, as in Fig. 5.4, it has a finite width of 0.5, from $x = 2.1$ and $x = 2.6$, which means that it moves more slowly in the x-direction. The fast mode rarefaction wave is responsible for driving the inflow of material towards the slow-mode shock. For this slower rarefaction wave, the inflow speed of the ionised fluid is halved when γ_{ion} and γ_{rec} are included. This is in contrast with the speed of the neutral fluid in the x-direction, which stays similar in magnitude.

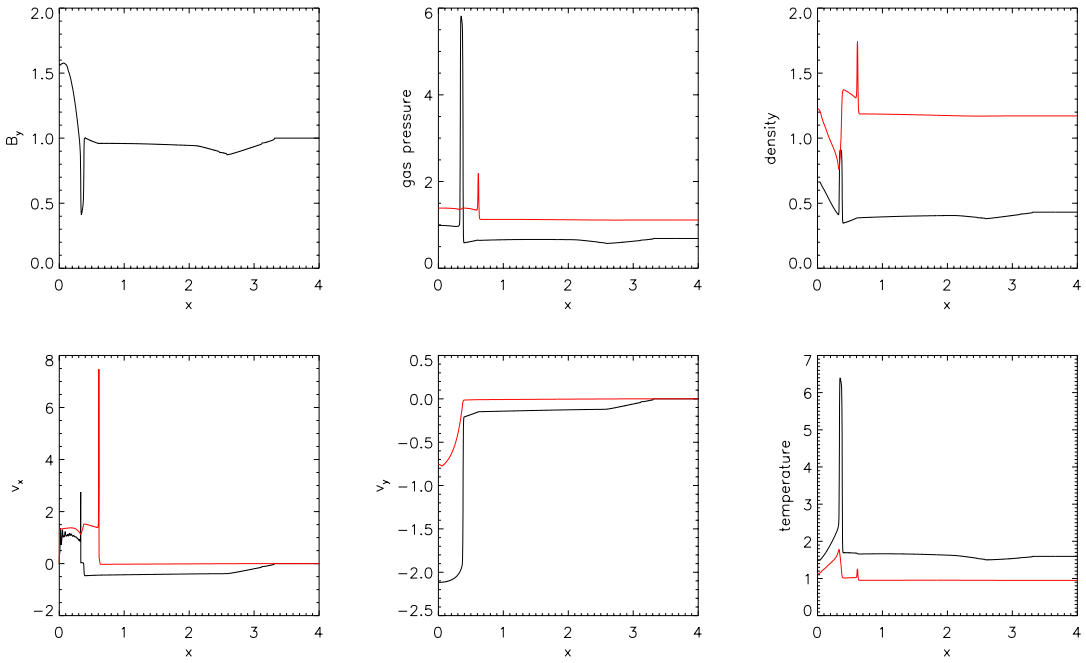

Figure 5.5: Hillier shock, ratios of quantities without to with ionisation and recombination. The ratio of the ionised fluid is presented in *black* and the ratio of neutrals in *red*.

Furthermore, the structure of the slow-mode shock is affected by the inclusion of γ_{ion} and γ_{rec}. Post-shock, there is a high-velocity jet in the y-direction. With γ_{ion} and γ_{rec}, the speed of this jet, visible in v_y, decreases to half for the ionised fluid and increases marginally for the neutral fluid. In the x-direction, the post-shock speeds, as can be seen in v_x, of the ionised fluid as well as the neutral fluid decrease to half, with the inclusion of γ_{ion} and γ_{rec}. The position of the slow-mode shock fronts are similar, as can be seen in Figs. 5.3 and 5.4. The slow-mode shock shows sub-structures which are present with or without the inclusion of γ_{ion} and γ_{rec}. The post-shock pressure and mass density of the neutral fluid decrease by a factor of 0.7 for both quantities, with the inclusion of γ_{ion} and γ_{rec}. In contrast, for the ionised fluid, the gas pressure stays the same, whereas the mass density increases by a factor of 2.4, with the inclusion of γ_{ion} and γ_{rec}.

To further investigate and understand the evolution of the shock when γ_{ion} and γ_{rec} are included, the time series of the kinetic, thermal, and magnetic energies are plotted in Fig. 5.6. These quantities are obtained by summing over the whole domain for each time step. This reveals the energy conversion throughout the shock evolution. The green lines

represent the simulation with γ_{ion} and γ_{rec} included. The solid line is the ionised fluid and the dotted line the neutral fluid. Table 5.1 depicts the difference in energies at the initial state and at $t = 0.2$ for simulations with and without ionisation and recombination.

		E at $t = 0$	E at $t = 1$ with $\gamma_{ion} = \gamma_{rec} = 0$	ΔE from $t = 0$ to $t = 1$	E at $t = 1$ with $\gamma_{ion}, \gamma_{rec}$	ΔE from $t = 0$ to $t = 1$	difference in Δ E	ratio of E at $t = 1$
$E_{kinetic}$	ionised	0	145.6	+145	97.4	+97	48	0.67
$E_{kinetic}$	neutral	0	36.1	+36	28.4	+28	8	0.79
$E_{thermal}$	ionised	168.7	303.3	+80%	367.2	+117%	37%	1.21
$E_{thermal}$	neutral	754.1	890.9	+18%	758.2	+0.005%	18%	0.85
$E_{magnetic}$		2230.8	1777.8	-20%	1902.7	-15%	5%	1.07

Table 5.1: 1.5D slow-mode shock: energies with and without the inclusion of γ_{ion} and γ_{rec}, at time $t = 1$.

The table is structured as follows. The first column (*E at* $t = 0$) contains the energy values of each fluid at the beginning of the simulation, i.e. time $t = 0$. The second column (*E at* $t = 1$ *with* $\gamma_{ion} = \gamma_{rec} = 0$) contains the values of the energies at the end of the simulation, i.e. at $t = 1$ when ionisation and recombination terms are not included, i.e. the fluids are only coupled through collisions. If we subtract the amount of energy at the beginning of the simulation from the energy at the end ((*E at* $t = 1$*)* - *(E at* $t = 0$)), the difference of the energy (ΔE *from* $t = 0$ *to* $t = 1$) is obtained and this difference for the simulations without ionisation and recombination, is presented in the third column (ΔE *from* $t = 0$ *to* $t = 1$). This tells us how much energy is gained or lost over the time of the simulation. For this third column, the kinetic energies are not given in percentage unlike the other energies, because they are initialised with zero velocity. Hence, the numbers given are in non-dimensionalised units. The fourth column (*E at* $t = 0$) contains the energy values of each fluid at the end of the simulation, i.e. time $t = 1$, when ionisation and recombination rates included. If we subtract the energy at the beginning from the energy at the end (*E at* $t = 1$ - *E at* $t = 0$), but this time for simulations with ionisation and recombination included, the difference of the energy that is gained or lost over time, is obtained and they are presented in the fifth column, ΔE *from* $t = 0$ *to* $t = 1$. The sixth column (*difference in* ΔE), reveals the difference in energy (that is gained or lost over the simulation) with ionisation and recombination and without. The seventh column shows the ratios of the energies (with γ_{ion} and γ_{rec} / without γ_{ion} and γ_{rec}) at the end of the simulation, i.e. at $t = 1$. Note that this last column shows the difference of the energy values only at the end of the simulation and does not reflect the overall change over time or how much this change actually contributes to the overall energy gain or loss over time.

First of all, it can be said that the total energy is conserved; the loss of magnetic energy is balanced by the gain of kinetic and thermal energy. That the energy is conserved can be determined from Table 5.1 by adding up the energies at the beginning of the simulation when time $t = 0$ and comparing those with the magnitude of energy at the end of the simulation, when $t = 1$. This confirms the conservation of energy with a deviation of 0.0032% with collision rates only and of 0.0095% with γ_{ion} and γ_{rec}, over the time span. From the source terms, Eq. (2.134), it can be seen that the γ_{ion} and γ_{rec} play a role in the momentum and energy equations only when the velocities are non-zero. In the momentum equations, the terms containing the velocities are of similar size, which means that none of them dominates the physics. Similarly, in the energy equations, all the terms affect

the time evolution of the energy in a similar way. This means that γ_{ion} and γ_{rec} do not influence the energy evolution in a qualitative way, as shown in Fig. 5.6, where it can be seen that the graphs do not change their shape, but only the magnitude of the quantities change.

To investigate the effect of γ_{ion} and γ_{rec}, the simulation without these terms is used as the reference against which the changes in energy are measured. From Table 5.1 it can be seen that the kinetic energy of both, the ionised and neutral fluid, increases by a smaller amount with the introduction of γ_{ion} and γ_{rec}. This is due to the lower velocities of both fluids when γ_{ion} and γ_{rec} are included. This is also reflected in the ratio (with γ_{ion} and γ_{rec} / without γ_{ion} and γ_{rec}) of the kinetic energy, where the ratios are 0.67 and 0.79 for the ionised fluid and the neutral fluid, respectively. These ratios demonstrate that the overall change in kinetic energy for the neutral fluid is less than for the ionised fluid. For both fluids the thermal energy increases overall, but when γ_{ion} and γ_{rec} are included, the effect on the ionised fluid is larger than on the neutral fluid, as shown by the ratios in Table 5.1. The decrease of the thermal pressure of the neutral fluid is due to the decrease of its density and gas pressure. In Fig. 5.6, an overall decrease of the magnetic energy is revealed. In Table 5.1 the ratio of the magnetic energy at $t = 1$ shows that the decrease is reduced with the inclusion of γ_{ion} and γ_{rec}. This is because the whole process is slowed down; the discontinuity in the magnetic field accelerates the ionised fluid, which then gains kinetic energy. With γ_{ion} and γ_{rec}, the ionised fluid accelerates more slowly, therefore, the velocity has decreased and the energy stays with the magnetic energy instead of being converted to kinetic energy.

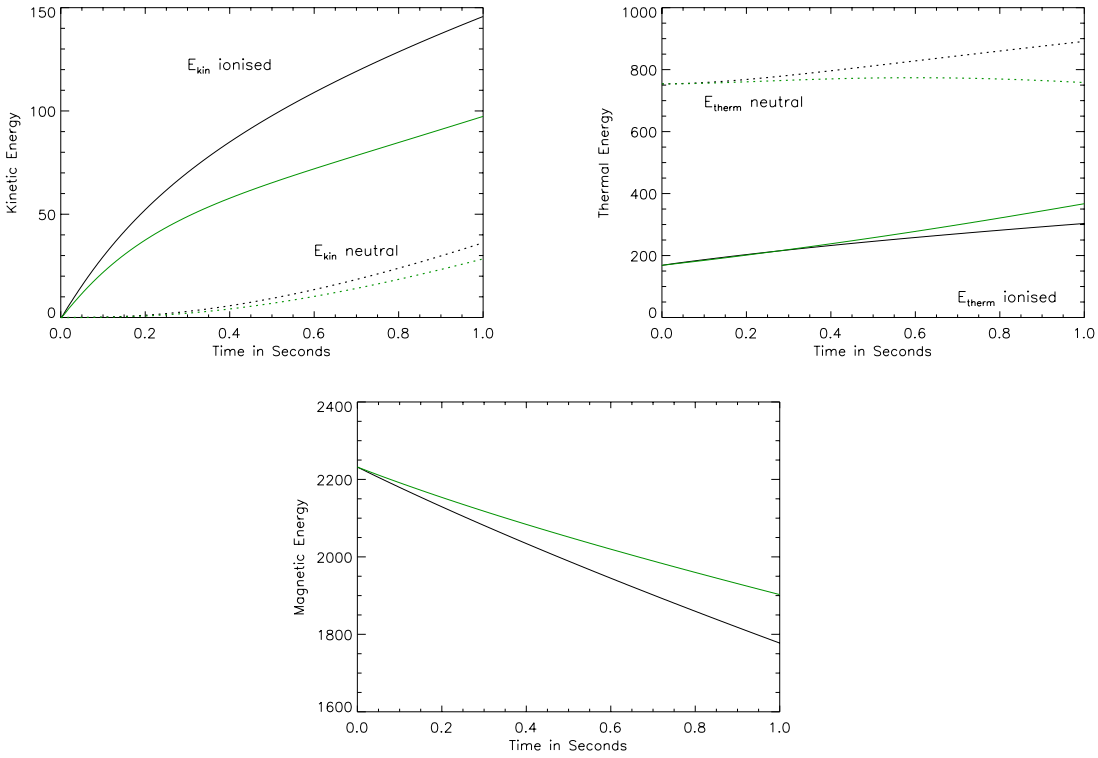

Figure 5.6: Kinetic energy, thermal energy and magnetic energy time series for the Hillier, Takasao, and Nakamura (2016) slow mode shock with collision rates only (black lines) and additional ionisation and recombination rates (green lines). The solid lines represent the ionised fluid and the dotted line the neutral fluid.

5.1.2 2D Orszag-Tang Vortex

We revisit the OT vortex simulation and run it this time with the two-fluid code, or in other words, in a partially ionised plasma setting. The conventional OT vortex simulation (Orszag and Tang, 1979) shows the transition to supersonic 2D MHD turbulence and tests a code's robustness to MHD shock formation, shock-shock interactions and how well the $\nabla \cdot \mathbf{B} = 0$ constraint is satisfied. This solenoidal condition is obtained from Maxwell's equations. Numerically, this is not always assured due to truncation errors in the discretisation schemes (Guillet *et al.*, 2019). Therefore, the divergence cleaning scheme introduced by Dedner *et al.* (2002) is implemented in the code and presented in Sec. 4.3.3. Ryu *et al.* (1998) simulate the OT vortex for ideal MHD equations and our successful replication of it, with a resolution of 512×512 pixels, is presented in Sec. 4.3.2. The

initial conditions in 2D for the two-fluid OT vortex are:

$$\begin{bmatrix} \rho_p &= & \gamma P \\ P_p &= & \beta B_0^2/2 \\ v_{xp} &= & -v_0 \sin(2\pi y) \\ v_{yp} &= & v_0 \sin(2\pi x) \\ B_x &= & -B_0 \sin(2\pi y) \\ B_y &= & B_0 \sin(4\pi x) \\ \rho_n &= & \gamma P \\ P_n &= & \beta B_0^2/2 \\ v_{xn} &= & 0 \\ v_{yn} &= & 0 \end{bmatrix}.$$

Here, $B_0 = -1/\sqrt{4\pi}$, $\beta = 10/3$, $P = \beta B_0^2/2$, $v_0 = 1$ and the boundaries are periodic everywhere. For verification purposes, the two fluids were decoupled in the code by setting the collision frequency, ionisation, and recombination terms to zero. This means, the equations have a RHS of zero. A solution exactly like in Fig. 4.18 for the ionised fluid is retrieved, while the neutral fluid stays in equilibrium and does not form a vortex. Now, coupling the two fluids again by switching on the collision terms in the source term and repeating the simulation, a vortex formation in both fluids can be observed. For this simulation, the collision frequency lies between 0.8 and 1 and the initial neutral fraction is 50%. The OT vortex plots for the coupled two fluids are shown in Fig. 5.7. The horizontal cut at $y = 0.428$ through the 2D plane (1D plots) reveals the variation of the gas and magnetic pressure along this line. As can be seen in the plots of the neutral gas pressure and temperature, the neutral fluid forms a vortex as well, which, however, is less defined. This is expected, because even though neutrals do not react to the forces of the magnetic field and their initial velocity is zero, they are coupled to the ionised fluid and, therefore, move with the ionised fluid. To what extent they move with the ionised fluid and form a vortex, depends on the strength of the coupling. If the coupling is strong, momentum is exchanged quicker and the neutral fluid would form a vortex quicker too. If there is weak coupling, and collisions between the neutral fluid and the ionised fluid are rare, it would take more time for the neutral fluid to form a vortex. The collision frequency evolves in time and lies between 0.8 and 1 for the OT vortex simulation, calculated with Eqn. (2.142). γ_{ion} and γ_{rec} are included and the OT vortex simulations is run again, which

is shown in Fig. 5.8. As the changes are not visible in the 2D plots of this figure, the 1D cut through the computational domain serves as the better means of comparison.

The OT vortex simulation with collision terms only and with ionisation and recombination included are compared. In order to do so, we also plot the spatial distribution of the magnetic field component B_y, the gas pressure, the mass density, the velocity components v_x and v_y, and the temperature along a line across a shock region in the OT vortex simulation, as marked in Fig. 5.9. For this region, the Figures 5.10 and 5.11 reveal the spatial distribution and magnitude of the physical quantities therein. They depict the behaviour of the fluids along the marked line across the shock front, where most changes happen, as it is highly dynamic and non-linear. That means, if we want to see the differences, this region is the best place to look at. An even better insight is given when looking at each of the fluids individually. In Fig. 5.12, the physical quantities along the line across the shock for the ionised fluid only are plotted and in Fig. 5.13, the quantities for the neutral fluid only are plotted, where the solid line represents the simulation with collisions only and the dotted line the simulation with γ_{ion} and γ_{rec}.

Additionally, the energy conversion during the vortex formation is of interest. Plotting the energy time series (Fig. 5.14) illustrates the difference between the energy conversion during the vortex formation when only collision rates are included (black lines) and when, additionally, γ_{ion} and γ_{rec} are included (green lines). Throughout the simulation, the total energy is conserved.

Table 5.2 depicts the difference in energies at the initial state and at $t = 0.2$ for simulations with and without ionisation and recombination. In the previous section, Sec. 5.1.1, the headers of this table have been explained in more detail. That the energy is conserved can be determined from Table 5.2 by adding up the energies at the beginning of the simulation when time $t = 0$ and comparing those with the magnitude of energy at the end of the simulation, when $t = 0.48$. This confirms the conservation of energy with a deviation of 0.0019% with collision rates only and of 0.0013% with γ_{ion} and γ_{rec}, over the time span.

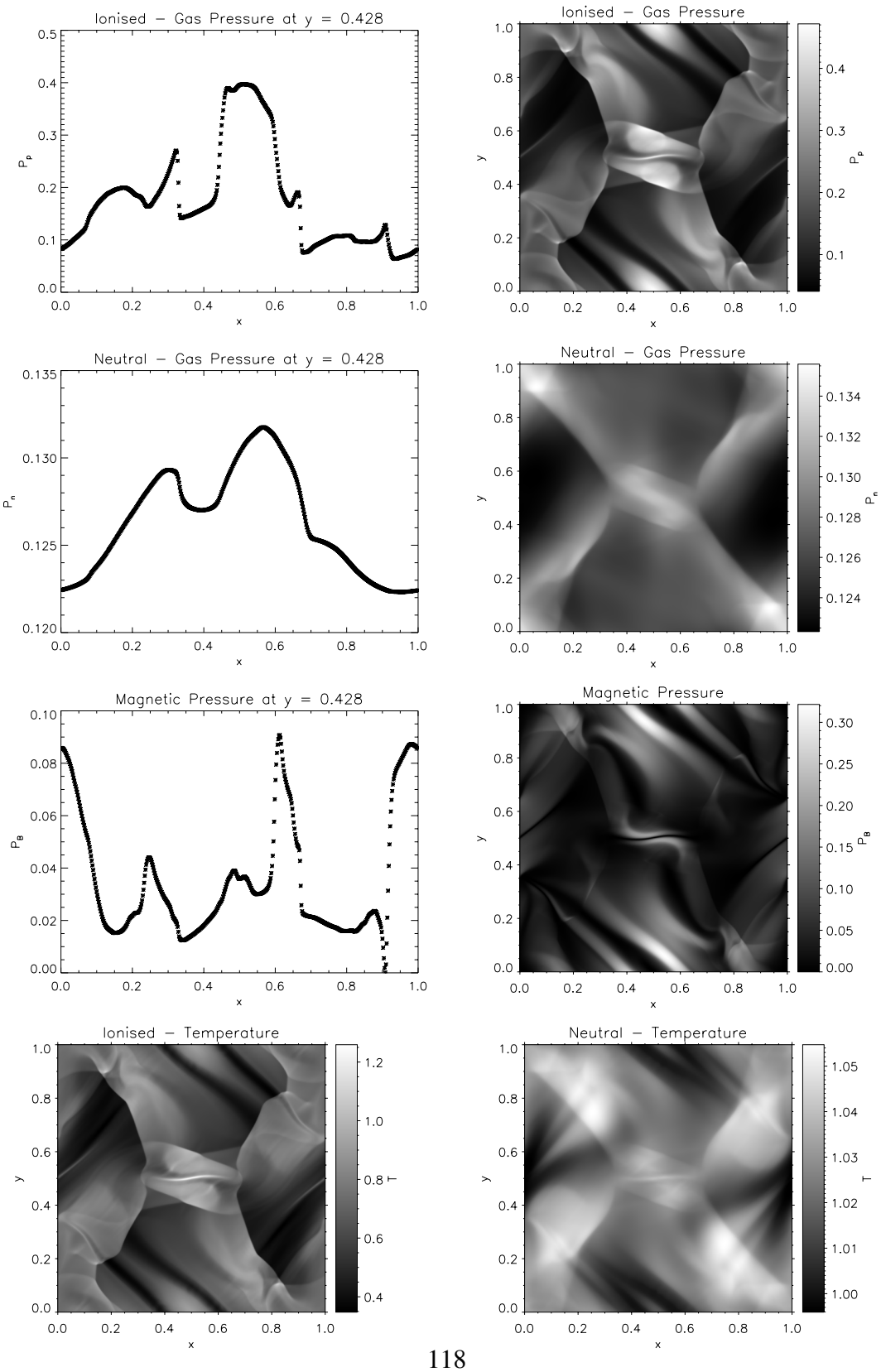

Figure 5.7: Two-fluid Orszag-Tang vortex test. The cut through the computational domain (1D plots) shows the variation of the physical quantity in the 2D plane at $y = 0.428$ for the ionised fluid's gas pressure (top row), the neutral fluid's gas pressure (second row) and the magnetic pressure (third row). The bottom row plots show the temperature distribution for each fluid.

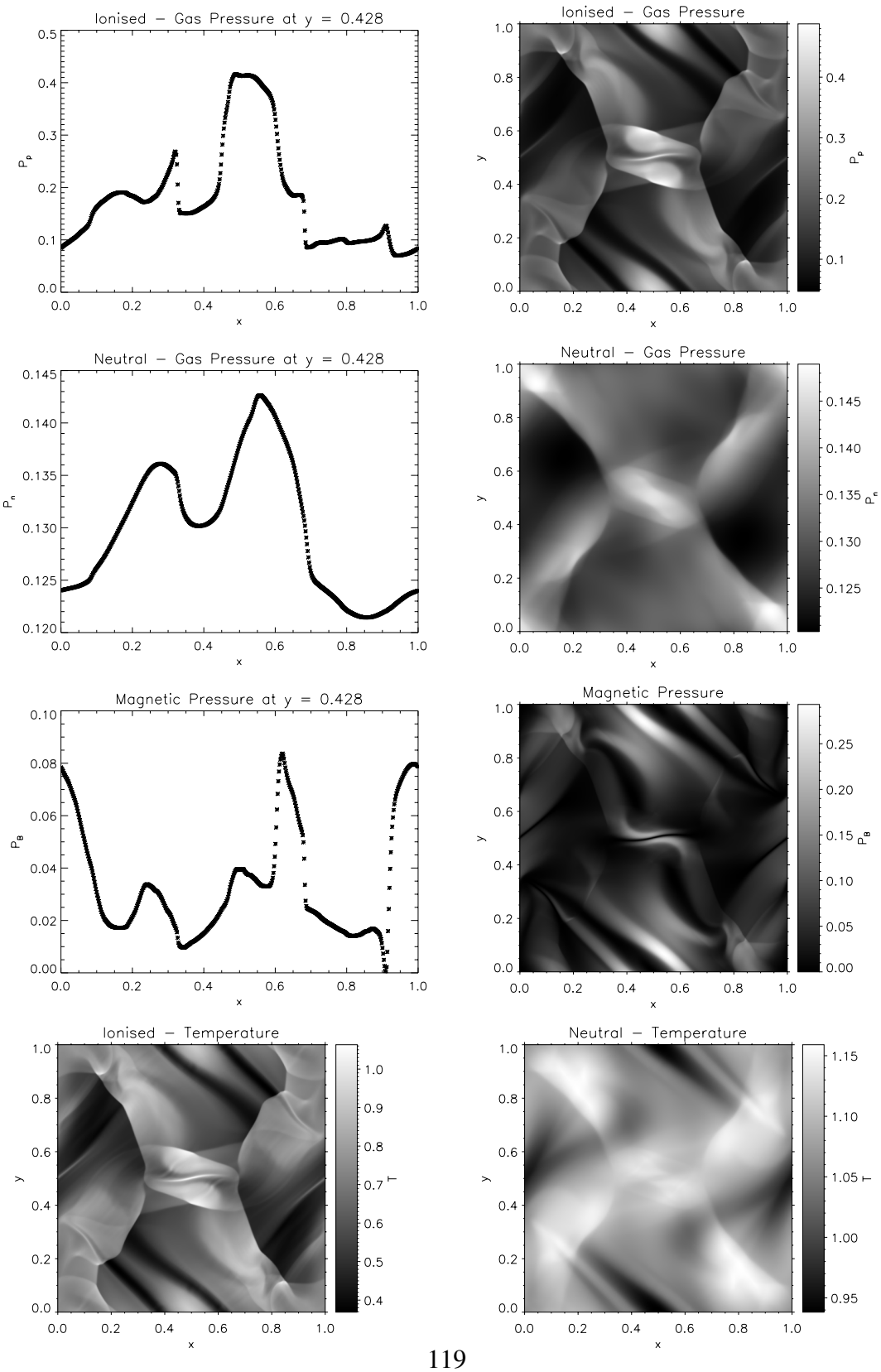

Figure 5.8: Two-fluid Orszag-Tang vortex test, with γ_{ion} and γ_{rec} included. The cut through the computational domain (1D plots) shows the variation of the physical quantity in the 2D plane at $y = 0.428$ for the ionised fluid's gas pressure (top row), the neutral fluid's gas pressure (second row) and the magnetic pressure (third row). The bottom row plots show the temperature distribution for each fluid.

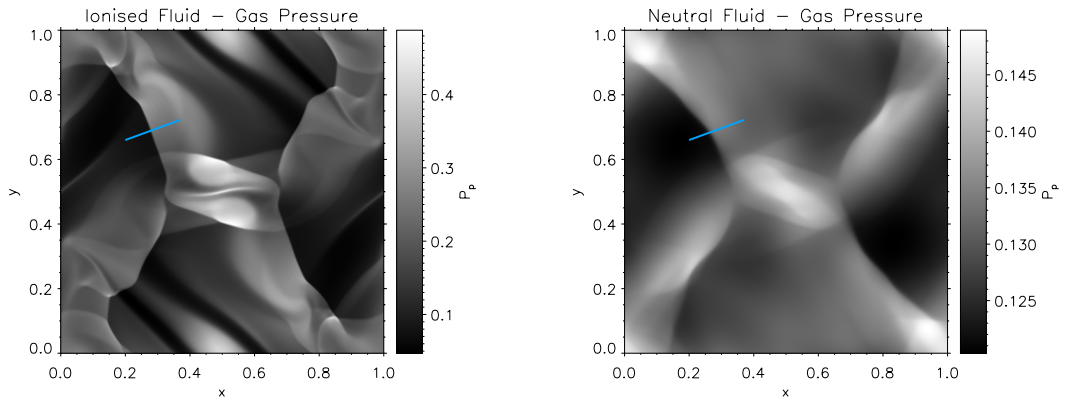

Figure 5.9: 1D profiles of physical parameters along the blue line are presented in Fig. 5.12 for the ionised fluid and for the neutral fluid in Fig. 5.13

.

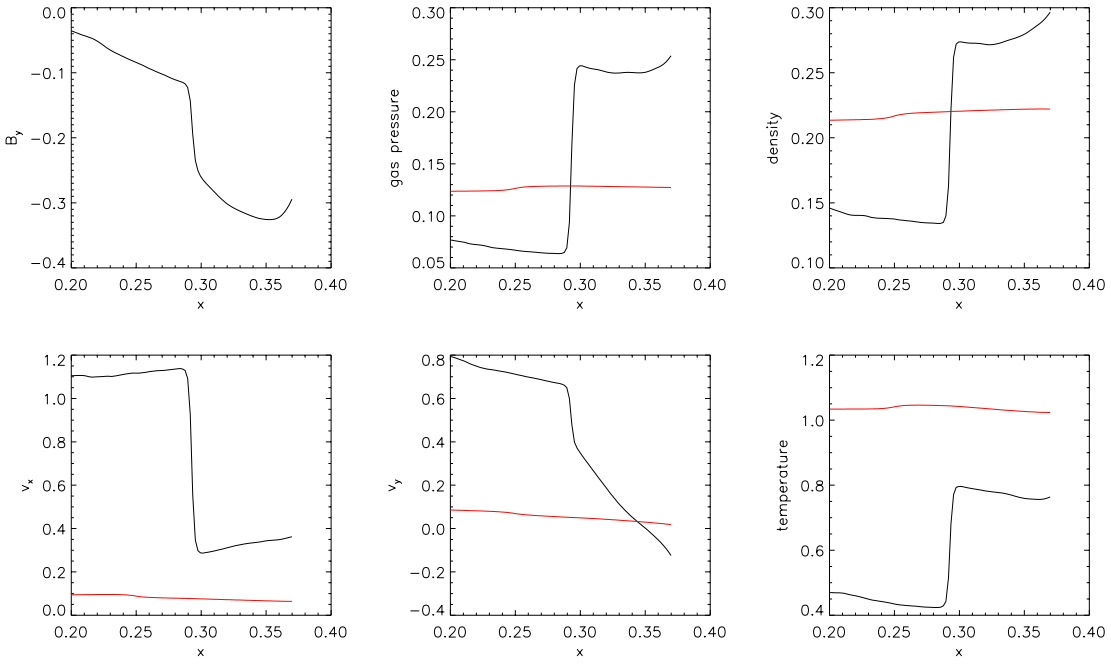

Figure 5.10: 1D profiles - no ionisation and recombination rates. From left to right are presented at time $t = 0.48$: the spatial distribution of the magnetic field component B_y, gas pressure, mass density, velocity components v_x and v_y, and the temperature.

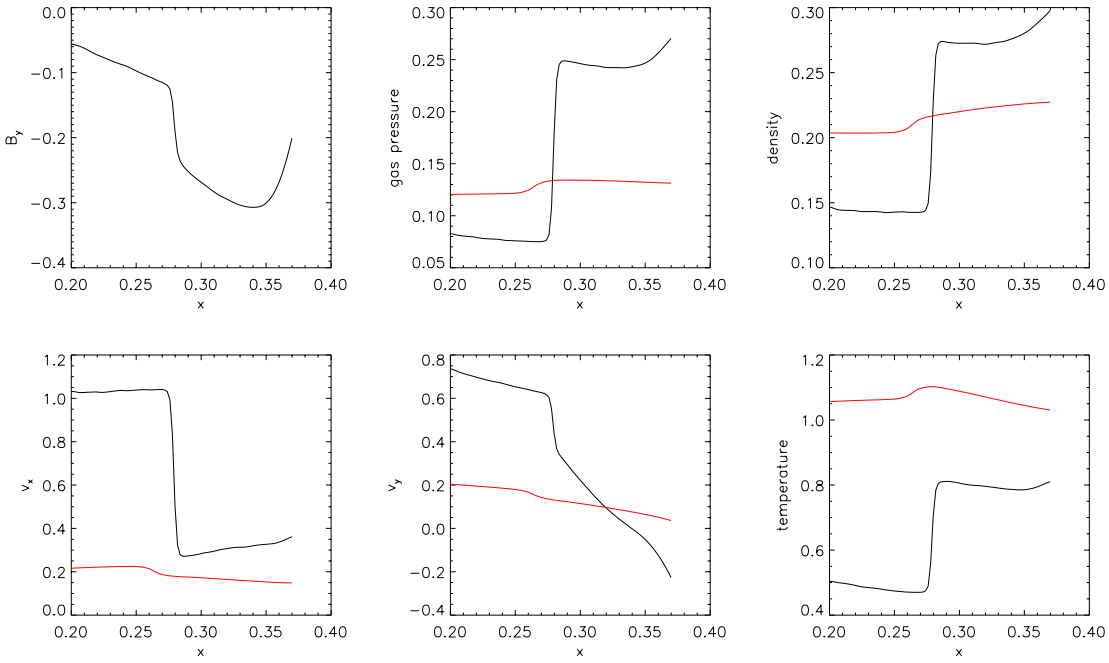

Figure 5.11: 1D profiles - with ionisation and recombination rates. From left to right are presented at time $t = 0.48$: the spatial distribution of the magnetic field component B_y, gas pressure, mass density, velocity components v_x and v_y, and the temperature.

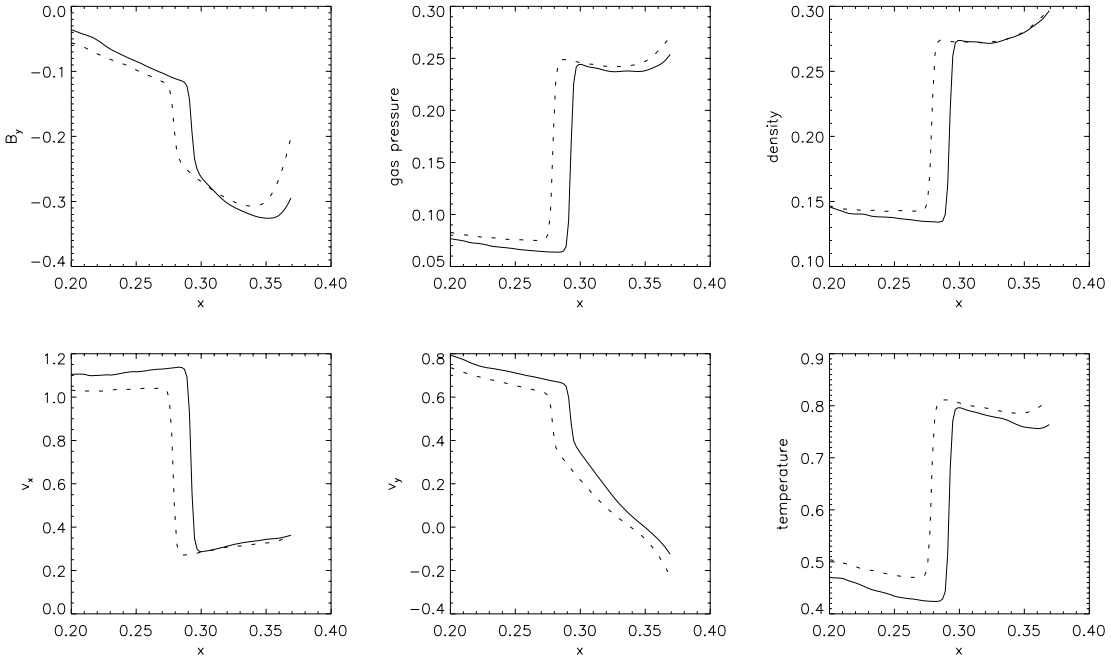

Figure 5.12: Ionised fluid: collision rate only (solid line) and with ionisation and recombination rates in (dotted line). From left to right are presented at time $t = 0.48$: the spatial distribution of the magnetic field component B_y, gas pressure, mass density, velocity components v_x and v_y, and the temperature.

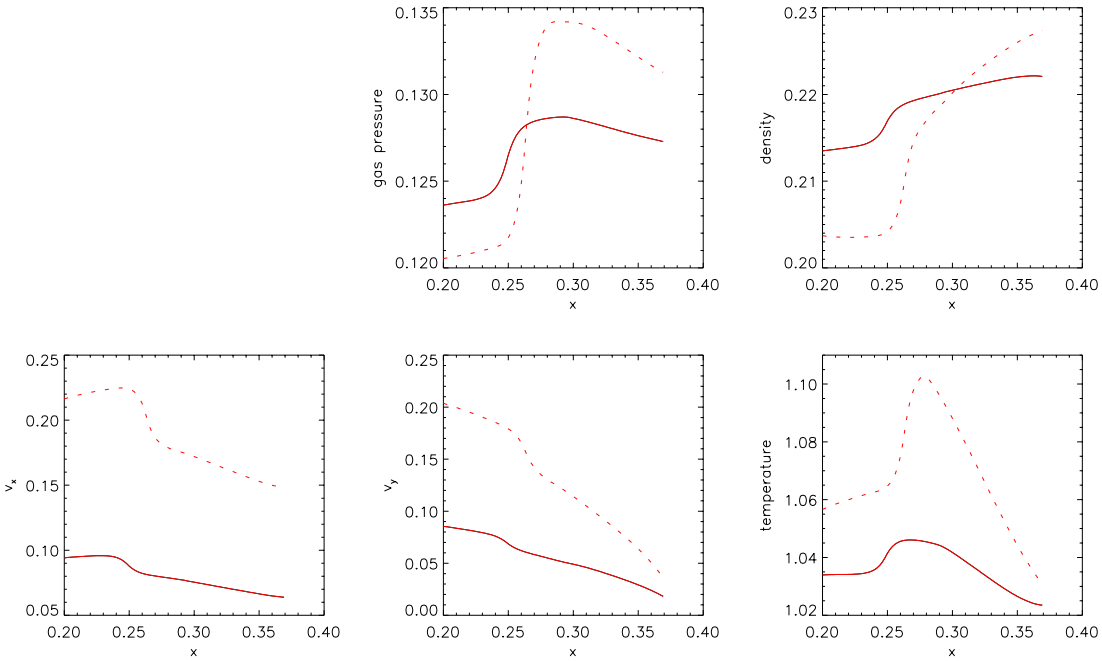

Figure 5.13: Neutral fluid: collision rate only (solid line) and with ionisation and recombination rates in (dotted line). From left to right are presented at time $t = 0.48$: the spatial distribution of the gas pressure, mass density, velocity components v_x and v_y, and the temperature.

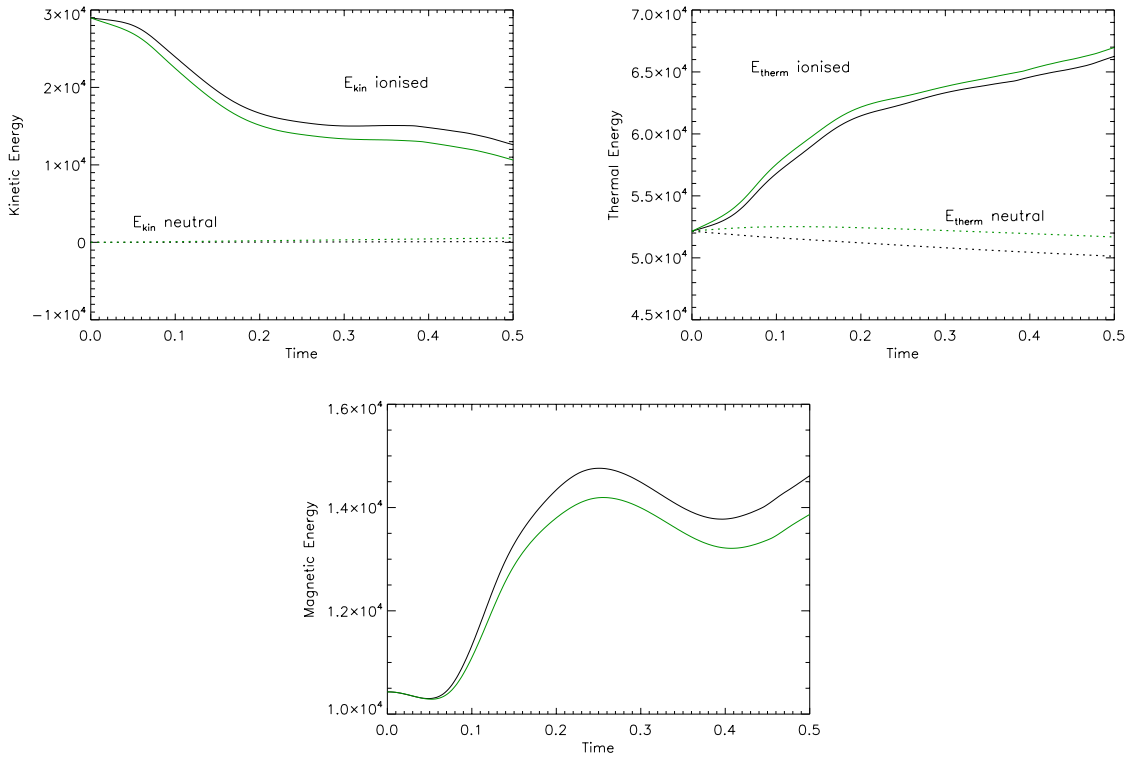

Figure 5.14: Kinetic energy, thermal energy and magnetic energy time series for the Orszag-Tang vortex with collision rates only (black lines) and additional ionisation and recombination rates (green lines). The solid lines represent the energy of the ionised fluid and the dotted line the energy of the neutral fluid.

		E at $t = 0$	E at $t = 0.48$ with $\gamma_{ion} = \gamma_{rec} = 0$	ΔE from $t = 0$ to $t = 0.48$	E at $t = 0.48$ with $\gamma_{ion}, \gamma_{rec}$	ΔE from $t = 0$ to $t = 0.48$	difference in Δ E	ratio of E at $t = 0.48$
$E_{kinetic}$	ionised	28973.3	10968.8	-62%	9064.3	-69%	7%	0.83
$E_{kinetic}$	neutral	0	111.3	+111	604.4	+604	493	5.43
$E_{thermal}$	ionised	52151.9	67523.9	+29%	68185.7	+31%	2%	1.01
$E_{thermal}$	neutral	52151.9	49941.9	+4%	51504.5	+1%	3%	1.03
$E_{magnetic}$		10430.4	15158.9	+45%	14346.6	+37%	8%	0.95

Table 5.2: Orzsag-Tang vortex: energies with and without the inclusion of γ_{ion} and γ_{rec}, at time $t = 0.48$.

Now, comparing the plots without γ_{ion} and γ_{rec} (Fig. 5.7), to the plots with γ_{ion} and γ_{rec} (Fig. 5.8), the most obvious difference is the neutral gas pressure, which has increased and is about a quarter of the magnitude of the ionised fluid's gas pressure. The peaks of the neutral gas pressure increased by 5% for the left peak at $x = 0.27$ and by 8% for the right peak at $x = 0.58$. The gradient leading to each peak changes from $m = 0.035$ to $m = 0.074$ for the left peak and from $m = 0.03$ to $m = 0.074$ for the right peak. From Fig. 5.8, the damping of the magnetic field is manifested in a smoother profile, where small-scale spatial variations in the amplitude at around $x = 0.5$ are smaller than with collision rates only (Fig. 5.7).

Compared to simulations with collision rates only, the lower magnetic field pressure, with γ_{ion} and γ_{rec} included, might be caused by the slower development of the vortex formation. This is also visible in the magnetic energy plot in Fig. 5.14 where the magnetic energy increases. From Table 5.2 it can be seen that there is an increase of 37% when γ_{ion} and γ_{rec} are included, compared to 45% without γ_{ion} and γ_{rec}. This slowing down of the development is visible in the plots of Fig. 5.12, where γ_{ion} and γ_{rec} are included: the velocity plots show that the x- values are positive, which means the shock is travelling in the positive x-direction. Because the shock front is found at a lower x value, it can be inferred that the shock has not traveled as far as without γ_{ion} and γ_{rec}, where the difference in the shock position for the ionised fluid is $\Delta x = 0.012$. One can also see this in the energy plots, Fig. 5.14, where a stronger decrease in kinetic energy of the ionised fluid can be observed when γ_{ion} and γ_{rec} are included. Table 5.2 shows that this difference in the change of kinetic energy for the ionised fluid is 7%. The neutral temperature values lead to a domination of ionisation rate over the recombination rate, Fig. 5.1. From Eqn. (2.134), it follows that this leads to a sink in the momentum equation for the ions and a source of momentum for the neutrals.

Looking at the neutral fluid plots (Fig. 5.13), it can be seen that it is the other way around: the shock front has traveled further in the neutral fluid, when including γ_{ion} and γ_{rec}, where $\Delta x = 0.010$, and the velocities have more than doubled. This is also reflected in Table 5.2, where the ratio (with γ_{ion} and γ_{rec} / without γ_{ion} and γ_{rec}) shows an increase of the neutral fluid's kinetic energy of 5.43. For the ionised fluid (Fig. 5.12), with γ_{ion} and γ_{rec} included, there are changes in the v_x and v_y velocities (each decreases by 7%), in the temperature (increases by 10%) and a shift of the shock front (with $\Delta x = 0.012$).
The bigger change can be seen for the neutral fluid when compared to the ionised fluid.

At the shock front, as in Fig. 5.13, the temperature of the neutral fluid increases from $T = 1.045$ (without γ_{ion} and γ_{rec}) to $T = 1.12$ (with γ_{ion} and γ_{rec}). The neutral gas pressure and mass density show a different behaviour around the shock. Post-shock ($x = 0.2$ to $x = 0.26$), with γ_{ion} and γ_{rec} included, the gas pressure and mass density show a lower value than before the shock (from $x = 0.28$ to $x = 0.4$), compared to without γ_{ion} and γ_{rec}. Similarly, the y-component of the magnetic field in Fig. 5.12 shows a lower magnitude post-shock (by 20%) and a higher magnitude pre-shock, when γ_{ion} and γ_{rec} are included. Furthermore, the total gas pressure over the whole domain increases for the ionised fluid and decreases for the neutral fluid, which is reflected in the thermal energy plot, Fig. 5.14. However, this decrease of the neutral fluid's thermal energy is less by 3% when γ_{ion} and γ_{rec} are included, while the increase of the ionised fluid's thermal energy is by 2%, see Table 5.2. Figure 5.14 and Table 5.2 show that the magnetic energy increases without γ_{ion} and γ_{rec} by 45% and with γ_{ion} and γ_{rec} included by 37%, which implies that less energy has been converted to magnetic energy over the vortex formation.

Finally, we want to briefly compare the two-fluid OT vortex simulation to the ideal MHD, single-fluid simulation of the previous chapter, Sec. 4.3.2, where a fully ionised, ideal plasma is assumed. Therefore, the energy plots in Fig. 5.14 are compared to Fig. 5.15. It can be observed that the sole inclusion of collisions slows down the process or inhibits the driving mechanism, which is the velocity field; without collisions, the kinetic energy loss is smaller, and hence, the energy stays in the velocity field and drives the vortex, where also a stronger increase of magnetic energy can be observed. As less magnetic energy is converted to thermal energy in the ideal MHD simulation, the thermal energy increases less strong. This suggests that collisions inhibit the vortex formation and evolution, and the inclusion of ionisation and recombination does so even stronger, so that the driver is even less efficient.

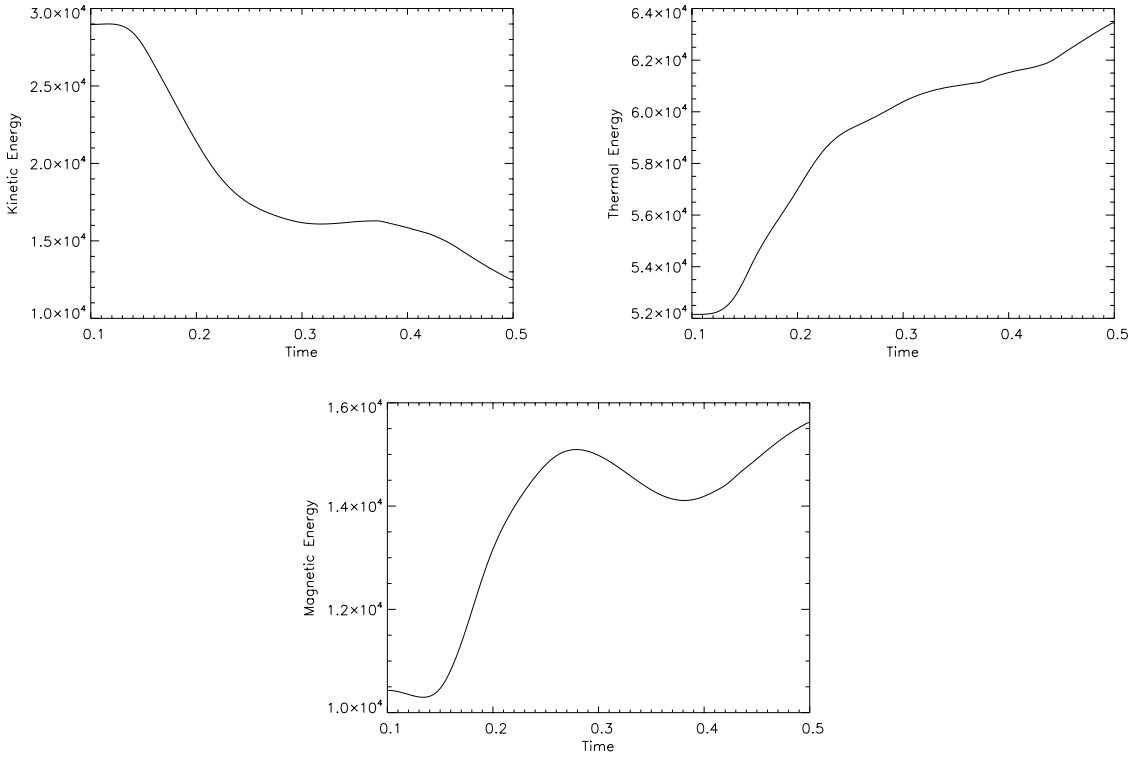

Figure 5.15: Ideal MHD (single fluid): Kinetic energy, thermal energy and magnetic energy time series for the Orszag-Tang vortex.

Chapter 6

Conclusions

6.1 Conclusion With Regards to Two-Fluid Simulations

The two systems of equations (hydrodynamic and MHD) are coupled with a collision term α_c and form a two-fluid code. To determine if the two-fluid code produces expected results, the 1.5D slow-mode shock, which is simulated in Hillier, Takasao, and Nakamura (2016) (with collision terms only), is replicated and our results show a good agreement. Also with collision rates only, the OT vortex is initialised in the two-fluid setting.

Furthermore, the effects of ionisation and recombination on the slow-mode shock and the Orszag-Tang (OT) vortex simulation are studied and in order to achieve this, ionisation and recombination rates, γ_{ion} and γ_{rec}, respectively, are added. The effect of ionisation and recombination on both fluids, the ionised and the neutral fluid, are compared to simulations where collisions are the only coupling mechanism between the fluids. This means that now the two fluids do not only interact through collisions, but ionisation and recombination are an additional coupling mechanism and their effect on the fluid properties and the energy conversion is studied. Mathematically, this is manifested in the source terms in Eqn. (2.134), where terms related to ionisation and recombination (γ_{ion} and γ_{rec}) act as sinks and sources in the density, momentum and energy equations. As can also be seen in the source term expression, Eqn. (2.134), is that if the velocities are all zero, then γ_{ion}, γ_{rec} and α_c do not come into effect, because all of these terms are related to the velocity of the fluids, either the drift velocity ($v_D = v_n - v_p$) or the velocity of the individual fluid. If both fluids' velocities are non-zero and exactly the same, the drift velocity is zero and

there are no collision effects, as both fluids move as one, but there are effects related to γ_{ion} and γ_{rec}, as these are dependent on the individual fluid velocity. If the velocities of both fluids are different, there exists a drift velocity which affects the collision term and, therefore, shows that the degree of coupling through collisions is dependent on the drift velocity. This drift velocity changes with the inclusion of γ_{ion} and γ_{rec}.
For example, in the 1.5D slow-mode shock simulation, the ionised fluid's velocity decreases with γ_{ion} and γ_{rec} with respect to the neutral fluid's velocity, which can be seen in the velocity plots of Fig. 5.4 and, therefore, the drift velocity decreases. Our solution of the slow-mode shock as in Hillier, Takasao, and Nakamura (2016), serves as a reference solution where only collisions are included. The discontinuity in the magnetic field is responsible for the domination of the ionised fluid in terms of the dynamics and the neutral fluid is only affected as it is coupled to the ionised fluid through collisions. Additionally, we add γ_{ion} and γ_{rec} and investigate the effect on the shock properties of both fluids. γ_{ion} and γ_{rec} are in the source terms, Eqn. (2.134), of the continuity equation and are responsible for the mass exchange and the resulting change of the temperature of both fluids. γ_{ion} and γ_{rec} terms influence the development of the shock by slowing down the ionised fluid and the rarefaction wave development, visible in the shift of the wavefront by $\Delta x = 0.7$, see Fig. 5.4. This is due to the increasing interaction of the two fluids when γ_{ion} and γ_{rec} terms are added to the source terms, which results in an enhanced momentum exchange, where the momentum is transferred from the ionised fluid to the neutral fluid.

In that process, the kinetic energy does not increase as much as without γ_{ion} and γ_{rec} and there is less energy conversion from magnetic energy to kinetic energy, while the total energy is conserved, see Fig. 5.6. An overview of the energy changes relative to simulations where $\gamma_{ion} = \gamma_{rec} = 0$, at time $t = 0.48$, can be found in Table 5.1. The overall kinetic energy of both fluids increases. However, with γ_{ion} and γ_{rec}, the increase is less by 48 normalised units for the ionised fluid and for the neutral fluid it is less by 8 normalised units, which can be seen in Table 5.1. The energy that has not been converted to kinetic energy stays with the magnetic energy, where an overall decrease can be observed, but with γ_{ion} and γ_{rec}, this decrease is less by 5% (Table 5.1).

Furthermore, it can be said that γ_{ion} and γ_{rec} influence the shock evolution and structure in both fluids, which can be seen in their density or velocity profiles (Fig. 5.4). The density of the ionised fluid increases by a factor of 2.4, whereas the density of the neutral fluid decreases by a factor of 0.7, due to a source and a sink in the source term of each

fluid's continuity equation, respectively. The velocities in the x-direction decrease to half for both fluids. The post-shock jet, visible in the velocity in the y-direction, halves for the ionised fluid and marginally increases for the neutral fluid with γ_{ion} and γ_{rec}.

In 2D, the OT vortex simulation in the two-fluid setting is performed, with the initial conditions leading to a vortex formation in the ionised fluid only and the successful coupling between the ionised and neutral fluid can be confirmed, as a vortex has formed in the neutral fluid as well. If there was no coupling through collisions, the neutral fluid would stay in equilibrium as it initially has zero velocity and does not react to forces of the magnetic field. Similarly to the slow-mode shock, γ_{ion} and γ_{rec} also have an effect of slowing down the ionised fluid (the velocity decreases by 7%) and the development of the vortex formation, but it speeds and heats up the neutral fluid, the velocity doubles and the temperature increases from $T = 1.045$ to $T = 1.12$. This, again, is due to the additional interaction of the fluids when γ_{ion} and γ_{rec} terms are added to the source term. Furthermore, recombination dominates for most parts of the simulation, which leads to heating of the neutral fluid, visible in its energy source term, Eqn. (2.134).

Moreover, there is a shift of the shock front, visible in every plot of Fig. 5.12 and Fig. 5.13, for the ionised fluid ($\Delta x = -0.012$) and the neutral fluid ($\Delta x = +0.010$), respectively. Whereas, compared to collision terms only, the shock has not traveled as far in the ionised fluid, it has traveled further for the neutral fluid when γ_{ion} and γ_{rec} are included.

Figure 5.14 illustrates the energy conversion throughout the vortex evolution. The total energy is conserved. Regarding the thermal energy, the ionised fluid shows an overall increase, while the neutral fluid shows an overall decrease (Fig. 5.14). With γ_{ion} and γ_{rec}, the thermal energy increases by additional 2% for the ionised fluid, see Table 5.2, and decreases by 3% less for the neutral fluid. During the vortex formation, the overall kinetic energy decreases for the ionised fluid and increases for the neutral fluid. In Table 5.2 the ratio (with γ_{ion} and γ_{rec} / without γ_{ion} and γ_{rec}) shows an increase of the neutral fluid's kinetic energy of 5.43 with γ_{ion} and γ_{rec} included. It can also be seen that the kinetic energy of the ions decreases more by 7% with the inclusion of γ_{ion} and γ_{rec}. The magnetic energy increases overall without γ_{ion} and γ_{rec} and with γ_{ion} and γ_{rec}. It increases less by 8% at time $t = 0.48$ with γ_{ion} and γ_{rec}, as can be seen in Table 5.2.

To sum up the two-fluid investigations: The ionised and neutral fluids are connected

through a collision rate as well as ionisation and recombination rates (γ_{ion} and γ_{rec}). A 1.5D slow-mode shock that is initialised with a discontinuity in the magnetic field is studied, as well as a 2D Orszag-Tang vortex, which is initialised with a vortex profile in its velocity field and magnetic field. γ_{ion} and γ_{rec} are included in both systems and the resultant time evolution are measured. Quantitative changes to the energy flows are measured that are specific to each system. However, the overall time evolution of each system stays similar to its behaviour without γ_{ion} and γ_{rec} for the duration of the simulations. Note that the details of the energy evolution across both systems are different, since they are initialised in fundamentally different ways. A common phenomenon across the systems can be observed, namely that the inclusion of γ_{ion} and γ_{rec} results in the decrease of the movement of the ionised fluid.

6.2 Summary

Partially ionised plasma is ubiquitous in space and in order to simulate astrophysical phenomena accurately, a 2D two-fluid MHD code is developed to account for partially ionised plasma effects, based on the finite volume Kurganov-Tadmor scheme and written in C++. The advantage of this scheme is that it is Riemann-solver free, which makes computation faster, and it also exhibits a small numerical viscosity, independent of Δt, which makes the code stable and accurate for much smaller time steps than usual schemes allow for.

The code consists of two sets of equations, one for the neutral fluid (hydrodynamic equations) and one for the ionised fluid (ideal MHD equations). The system of equations is integrated in time with the forward Euler or fourth-order Runge-Kutta scheme. The performance of our code is validated by comparing our equations and initial conditions to a published reference and the code is verified by performing tests in the hydrodynamic, ideal MHD, and the two-fluid setting in 1D and 2D.

The 1D hydrodynamic sod shock tube test and its MHD equivalent, the Brio-Wu shock test, are very common tests to check a code's ability to handle discontinuities and shocks, in a neutral fluid and with a magnetic field, respectively. Our results show a very good agreement between our simulations and the reference solutions.

In 2D, the MHD code is tested with the Orszag - Tang (OT) vortex simulation, which

is commonly used in numerical MHD to test shock formation, shock-shock interactions, and to reveal the transition to supersonic turbulence. The vortex is initialised with a smooth profile in the velocity field and in the magnetic field, but quickly turns turbulent. Furthermore, two divergence controlling schemes are explored, the Powell source terms (Powell, 1994) and the Extended Generalised Lagrange Multiplier (EGLM) by Dedner *et al.* (2002). The results of the simulations with these schemes are compared to Guillet *et al.* (2019). The divergence controlling method by Dedner *et al.* (2002) yields the better result, i.e. a smaller global divergence of the magnetic field and a smoother solution in the non-shock regions. Our OT vortex simulation is compared to Fig. 3 in Ryu *et al.* (1998). Besides the 2D illustration of the vortex, Ryu *et al.* (1998) also show a 1D cut through the 2D computational plane, for a better comparison. Both, our 1D and 2D plots, match theirs and we, again, find that our code provides the expected and correct solution. In the two-fluid regime, the successful verification can be confirmed too. Additionally, simulations with the two-fluid code reveal the effects of ionisation and recombination that lead to interesting fluid behaviour and energy conversion, as discussed in the previous Section, Sec. 6.1.

To conclude, the code developed here provides a new tool to study magnetised plasmas, including the more complex partially ionised plasma. Due to the separate treatment of the species independently of their coupling degree, the range of its application is high; with this code, the solar atmosphere could be studied not only in the partially ionised plasma regime, i.e. the lower solar atmosphere, but also in the fully ionised plasma or the ideal MHD regime, i.e. the solar corona. As partially ionised plasmas and their proper two- or multi-fluid description have not been established for a long time in solar physics, the code offers an additional tool to investigate an enormous amount of physical phenomena and situations that could contribute to solving open questions about the Sun, like the coronal heating problem.

Chapter 7

Future Avenues of Investigation

The code developed here provides a tool to unlock new investigations into the lower solar atmosphere and will allow new physics on the Sun to be explored. As discussed in the introduction, phenomena like magnetic reconnection or wave damping are influenced by the partial ionisation state of a plasma and show different characteristics, compared to the single-fluid MHD simulations, and this also depends on the ionisation fraction. In the solar atmosphere, the density, temperature and pressure varies by one hundred orders of magnitudes throughout the solar atmosphere and this can make the code with the current ionisation and recombination rates unstable. This means that for the purpose of studying the solar atmosphere in the future, more elaborate ionisation and recombination rates and a time stepping scheme that can account for the collision time scale, have to be implemented. Currently, there is a separation of flux and source term time integration. This means that for the simulations with the current code to be very accurate, both, flux and source terms, have to have the same magnitude and act on same length and time scales. However, if collisions are very frequent for example, the source term dominates and an advanced time stepping scheme, which accounts for the time of the collisions, has to be applied. The aim is to implement the solar atmosphere in the initial conditions and have a self-constistent two-fluid model where ionisation and recombination terms handle the conversion of species self-consistently, as well as the energy and temperature profiles.

Furthermore, in order to continue the investigation that has started with this project, some more cases could be studied to confirm the findings of this work and improve our understanding about ionisation and recombination effects in a partially ionised plasma.

The Orszag-Tang vortex offers a wide range of investigations, as it exhibits many complex flow features and, for example, could be studied in terms of varying coupling degrees.

Another aspect that could also be explored is instabilities. In the following, preliminary results are presented. With a finite difference code, which is central in space and forward in time, the Rayleigh-Taylor instability and the Kelvin-Helmholtz instability are simulated in the hydrodynamic regime.

7.1 Rayleigh-Taylor Instability (RTI)

With a finite difference code, which was written in the beginning of this PhD project, the effect of gravitational acceleration on the Rayleigh-Taylor instability (RTI) was simulated and we would like to revisit it the two-fluid regime. The RTI is the instability of the interface between two fluids of different densities, where the denser fluid is on top of the lighter fluid or when they are accelerated towards each other (Chandrasekhar, 1961, p. 428). This is interesting because on the Sun, as a self-gravitating fluid body, the gravitational acceleration does not simply depend on the gravitating mass, as Newton's law states. In this more complex case, the gravitational acceleration depends on the density distribution within the star. According to the standard model of our Sun, the density varies by ten orders of magnitudes and the gravitational acceleration changes by one order of magnitude. This variation may affect various instabilities present in the solar plasma. The effects of spatially varying gravitational acceleration on the Rayleigh-Taylor instability (RTI) are simulated. The initial conditions are:

$$\mathbf{U} = \begin{cases} \begin{bmatrix} \rho &=& 1 \\ P &=& 2.5 + \rho g y \\ v_x &=& 0 \\ v_y &=& 0 \end{bmatrix} & \text{for } y \leq 0, \\ \begin{bmatrix} \rho &=& 2 \\ P &=& 2.5 + \rho g y \\ v_x &=& 0 \\ v_y &=& 0 \end{bmatrix} & \text{for } y \geq 0. \end{cases}$$

where a random perturbation is applied at the interface. A set-up of the simulation is illustrated in Fig. 7.1.

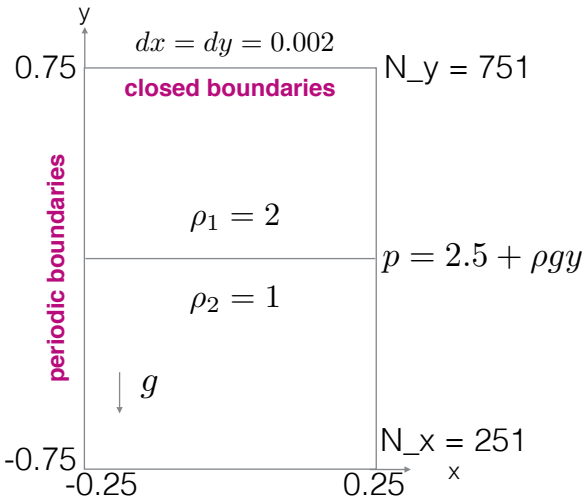

Figure 7.1: Set-up for the simulation of the RTI.

The RTI describes the instability of a dense fluid on top of a light fluid, due to gravity or acceleration of the fluid system. In that process, a certain flow pattern, characteristic for the RTI, evolves, which yields information about the physical properties of the fluids involved (e.g. density ratio, velocity) (Kull, 1991). Considering a magnetic field, various astrophysical phenomena can be described (solar atmospheric flux tubes, prominences) and conclusions about the magnetic field strength can be drawn (Díaz, A. J., Khomenko, E., and Collados, M., 2014; Hillier, 2016).

The code developed solves the governing fluid dynamic equations with a FD Lax-Friedrichs scheme, which is second-order (central difference) in space and first-order (forward Euler) in time. The flow structures are compared for three different gravitational acceleration set-ups, illustrated in Fig. 7.2 at different times.

In Fig. 7.3, the density plots are shown. From *left* to *right*, the gravitational acceleration varies from case 1 to case 3 as illustrated in Fig. 7.2 and from *top* to *bottom* the evolution at different times $t_0 = 0s, t_1 = 3s, t_2 = 6s$ is shown.

In Fig. 7.4, the velocity plots of the same are shown.

It would be interesting to study the case of varying gravitational acceleration further, analysing it with the linear stability analysis, with a magnetic field (i.e. in MHD) and also

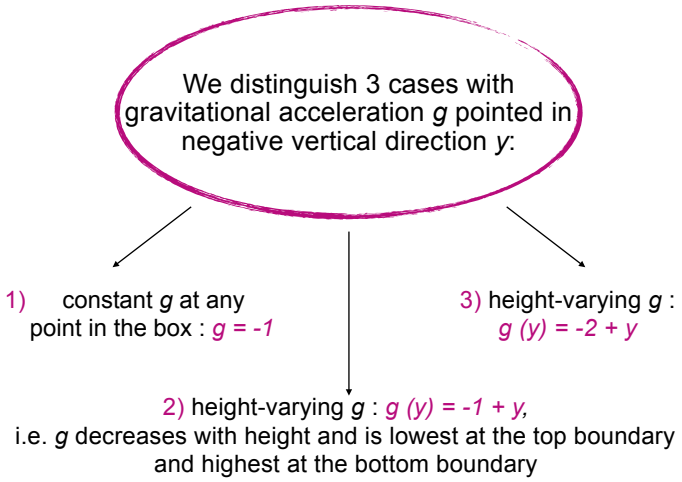

Figure 7.2: RTI, three cases of gravitational acceleration.

in the two-fluid setting. Furthermore, a more evolved code like our new two-fluid MHD code, would allow studying fine structures of the RTI more thoroughly, due its higher accuracy and resolution.

7.2 Kelvin-Helmholtz Instability (KHI)

The KHI is a common instability in fluid dynamics, but also fundamental in magnetised plasmas. KHIs can be seen in various astrophysical phenomena (McNally, Lyra, and Passy, 2012). For this instability to develop, there is usually either a shear in the velocity of the fluid or two fluids are in pressure equilibrium, but have significantly different densities and opposing velocities, which leads to the formation of vortices. Those vortices are sources of secondary instabilities, which can then lead to creation of turbulence (McNally, Lyra, and Passy, 2012). If the code successfully produces KHIs, it shows that it can handle multi-phase flows and the interaction between them. The initial condition for this simulation of the KHI are:

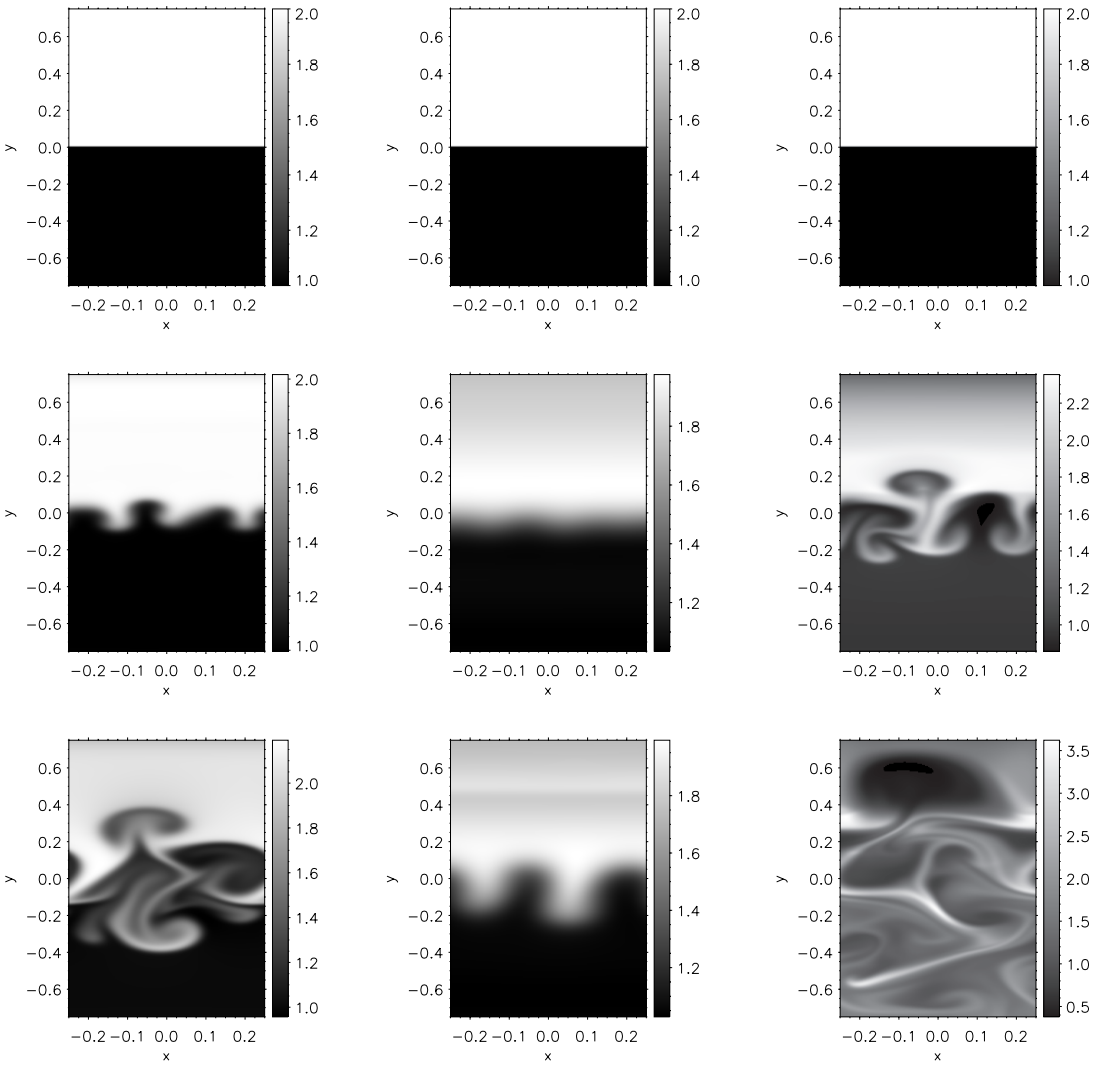

Figure 7.3: RTI density plots for the three different gravitational acceleration values (left to right: case 1, case 2 and case 3, as in Fig. 7.2, at three instances in time, top to bottom row: $t = 0$, $t = 3sec$ and $t = 6sec$.

$$\mathbf{U} = \begin{cases} \begin{bmatrix} \rho &=& 1 \\ P &=& 1 \\ v_x &=& 0 \\ v_y &=& 0 \end{bmatrix} & \text{for } 0.3 \geq y \geq 0.7, \\ \begin{bmatrix} \rho &=& 0.5 \\ P &=& 1.0 \\ v_x &=& -1 \\ v_y &=& 0 \end{bmatrix} & \text{for } y \geq 0.3 \text{ and } \leq 0.7. \end{cases}$$

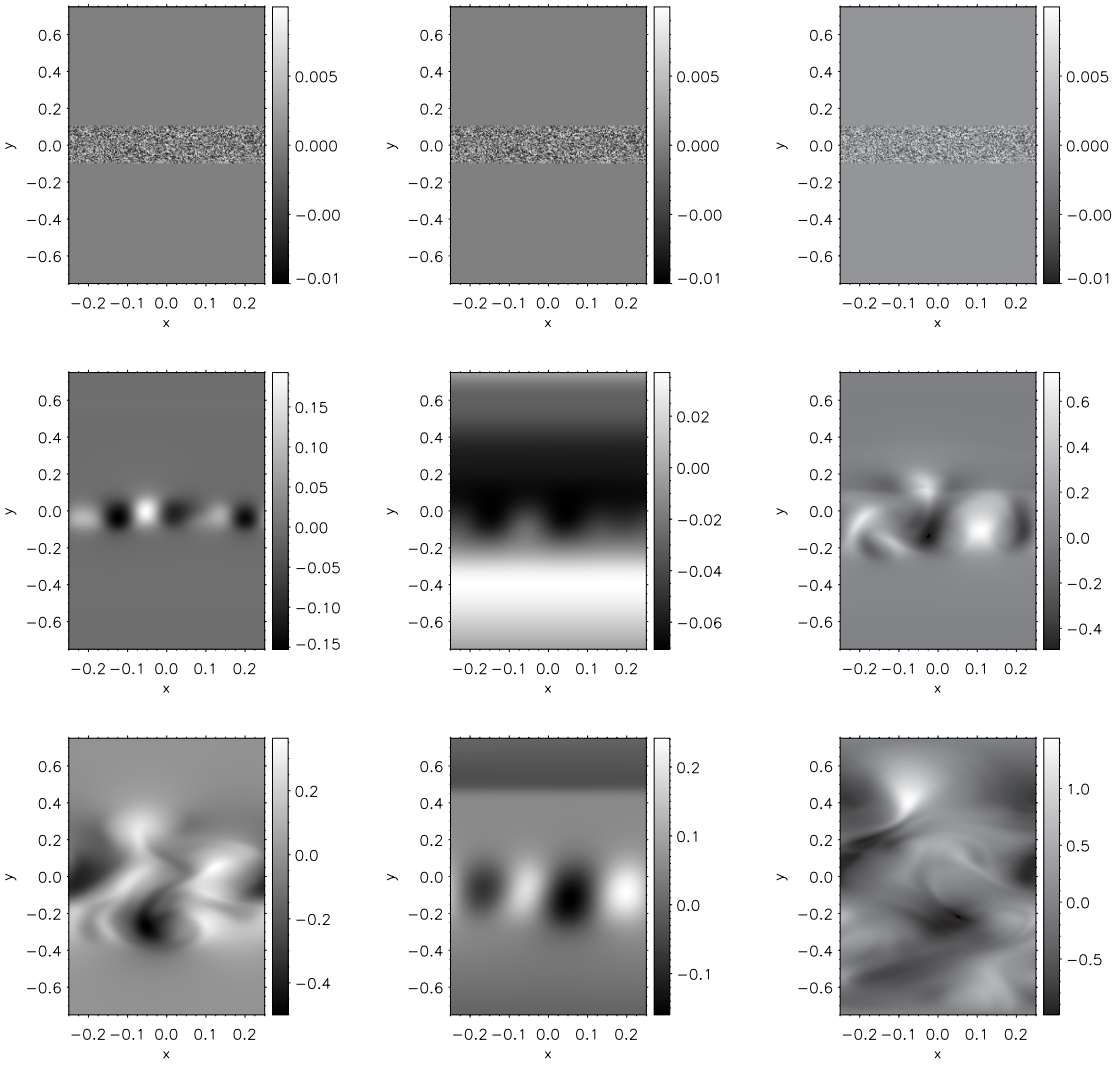

Figure 7.4: RTI velocity plots for the three different gravitational acceleration values (left to right: case 1, case 2 and case 3, as in Fig. 7.2, at three instances in time, top to bottom row: $t = 0$, $t = 3sec$ and $t = 6sec$.

with a perturbation $pert = 0.005sin(12.56x)$ at the interfaces. The KHI simulation is plotted in Fig. 7.5.

2D two-fluid simulations of the KHI have been investigated by Hillier (2019) in terms of their dynamics when the two fluids are decoupled and when the magnitude of their coupling varies. They found that the density of the two fluids can be coupled, while the velocity is decoupled. It would be interesting to see if this velocity decoupling still occurs

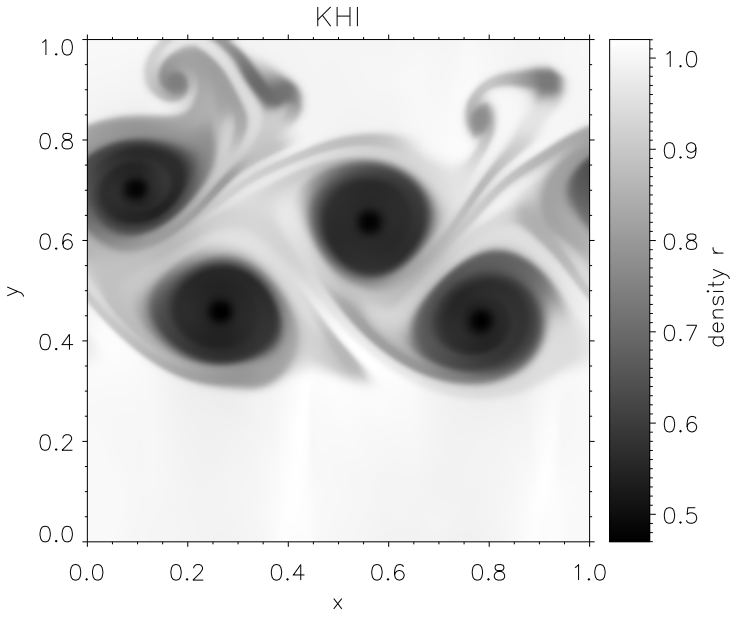

Figure 7.5: Kelvin-Helmholtz instability.

when ionisation and recombination take place.

Appendix A

Running the Code

The code can be compiled and run as follows.
Compiling the code :

In the terminal ($ indicates the terminal environment) type:

$ make clean (not necessary if the header file has not been changed)

$ make

Running the code:

$ mpirun -n number of cores ./code

Here, the number of cores requested for running the simulation is determined by the number of central processing units (CPUs) in x and y direction, n_cpu_x and n_cpu_y, respectively, which can be found in constants.h.

number of cores = n_cpu_x * n_cpu_y;

Reading in data in IDL, can be done as in Fig. A.1. This is an example how to read in data for the ideal MHD simulations. The *assoc*-function in IDL assigns a block of

```
path = '/path/to/data/'
file = file_search(path+'name_of_data*', COUNT=nFiles)
;this corresponds to n_x_global and n_y_global in constants.h
n_x = 512
n_y = 512
for i = 0, nFiles-1 do begin
  close, 1
  openr, 1, file[i]
  a = assoc(1, dblarr(n_x, n_y))
  x = a(0)
  y = a(1)
  rho_1 = a(2)
  mom_x_1=a(3)
  mom_y_1=a(4)
  mom_z_1 = a(5)
  ene_1=a(6)
  b_x = a(7)
  b_y = a(8)
  b_z = a(9)
  v_x = a(10)
  v_y = a(11)
  v_z = a(12)
  pre_1 = a(13)
  tem_1 = a(14)
endfor
close, 1
end
```

Figure A.1: Example of IDL routine to read in the data created by the code.

size $n\_x \times n\_y$ to every quantity that has been saved in the code. This means, the order in which the data for each quantity are saved in the c++ programme is important for the assignment of each block in the idl programme.

The saving procedure saves the x and y arrays and the whole solution vector, which means all variables: density, momentum, energy, magnetic field, velocity, pressure and temperature.

Check list:
Before running the code, it is good to check the following points:

- initial condition
- boundary condition
- save_bin name
- time / duration of the simulation and time step number
- CFL number and gamma
- x and y dimensions
- limiter

Appendix B

The Code

In the following, the entire 2D two-fluid code is presented, with all its functions. The code is written in C++ and currently constructed for 2D simulations. In some functions (e.g. get_pflux.cpp) of the code, the third dimension has been already added, even though it has not been used yet.

CONSTANTS.H

```
//physical constants
extern double ch;
extern double cp;
extern double T0;
const int n_cpu_x = 2;
const int n_cpu_y = 2;

const double CFL = 0.2;

const int n_t = 10000000;
const int i_t_save = 500; //save every i_t-th timestep

const int type = 6; //type of initial condition

const double pi = 3.14159265;
const double k_B = 1.380658e-16; //Boltzmann constant in erg / K (cgs)
const double m_e = 9.1095e-28; //g
const double m_1 = 1.6726231e-24; //molar mass ion
const double m_2 = m_e + m_1; //mass hydrogen cgs
const double m_in = m_1 * m_2 / (m_1 + m_2);
const double sigma2 = 25e12; // sigma = 50km, sigma2 = sigma*sigma


const double Rg = 0.56; // Hillier
//const double Rg = 8.314e7; // gas constant in cgs units: erg K mol

const int n_var = 27; // see below
const int n_consVar = 14; //number of conservative variables (rho, mom_x, mom_y, ene, B)


const int n_dim = 2;
const int n_ghost = 2;

const double density_threshold = 1.0e-20;
const double pressure_threshold = 1.0e-20;

const double gam = 5./3.;
const int n_x_global = 512;
const int n_y_global = 512;

const int n_x = (n_x_global / n_cpu_x) + 2 * n_ghost;
const int n_y = (n_y_global / n_cpu_y) + 2 * n_ghost;


//spatial domain

/*
//Hillier
const double start_x_global = -5.;
const double ende_x_global = 5.;

const double start_y_global = -5.;
const double ende_y_global = 5.;
*/


//RTI
//const double start_x_global = -0.5;
```

```
//const double start_y_global = -0.75;
//const double ende_x_global = 0.5;
//const double ende_y_global = 0.75;



//Orszag Tang
const double start_x_global = 0.;
const double start_y_global = 0.;
const double ende_x_global = 1.;
const double ende_y_global = 1.;




/*
//solar atmosphere
const double start_x_global = 0.;
const double start_y_global = 0e8;

const double ende_x_global = 1e8;
const double ende_y_global = 1.9e8;
 */


const double dx = (ende_x_global - start_x_global) / (1.0 * n_x_global);
const double dy = (ende_y_global - start_y_global) / (1.0 * n_y_global);

// _1_ mhd (ionised fluid)
const int rho_1_ = 0;
const int mom_x_1_ = 1;
const int mom_y_1_ = 2;
const int mom_z_1_ = 3;
const int ene_1_ = 4;

const int b_x_ = 5;
const int b_y_ = 6;
const int b_z_ = 7;

const int psi_ = 8;

// 2_ hydro (neutral fluid)

const int rho_2_ = 9;
const int mom_x_2_ = 10;
const int mom_y_2_ = 11;
const int mom_z_2_ = 12;
const int ene_2_ = 13;


const int v_x_1_ = 14;
const int v_y_1_ = 15;
const int v_z_1_ = 16;
const int pre_1_ = 17;

const int v_x_2_ = 18;
const int v_y_2_ = 19;
const int v_z_2_ = 20;
const int pre_2_ = 21;
```

```
const int a_ion_ = 22;
const int a_rec_ = 23;
const int f_col_ = 24;

const int tem_1_ = 25;
const int tem_2_ = 26;
```

INITIAL_CONDITION.CPP

```
/*
 various initial conditions
 * set the initial condition in constants: type = (number);
 */

#include<iostream>
#include<cmath>
#include"constants.h"

#include <string.h>
#include <fstream>
#include <sstream>

#include "mpi.h"

void get_xyindex(int, int*, int*);


void initial_condition(double*** u, double** x, double** y, double** gg)

{
int i,j;

double radius, pert;

int ind_cpu, i_cpu_x, i_cpu_y;
double start_x_in_global, start_y_in_global;

MPI_Comm_rank(MPI_COMM_WORLD, &ind_cpu);
get_xyindex(ind_cpu, &i_cpu_x, &i_cpu_y);
//get_xyindex(ind_cpu, i_cpu_x, i_cpu_y);

start_x_in_global = start_x_global + i_cpu_x * dx * (n_x - 2 * n_ghost); //start of each cpu in global
frame
start_y_in_global = start_y_global + i_cpu_y * dy * (n_y - 2 * n_ghost);



//assign x and y values

      for (i = 0; i < n_x; i++)
      {
         for (j = 0 ; j< n_y ; j++)
         {
            x[i][j] = start_x_in_global + i*dx - n_ghost*dx          // for each cpu
         }
      }



      for (i = 0; i < n_x; i++)
      {
         for (j = 0 ; j< n_y ; j++)
         {
            y[i][j] = start_y_in_global + j*dy - n_ghost*dy;

         }
```

```
    }


// ------------------------------------------------------------------------
//
////i.c. for sod shock tube test in x
if (type == 1)
{


        for (i = 0; i < n_x; i++)
           {
              for (j = 0 ; j< n_y ; j++)
                 {

                          if (x[i][j] < 0.5)
                    {
                    u[rho_1_][i][j] = 1.;
                    u[pre_1_][i][j] = 1.;
                    u[v_x_1_][i][j] = 0.;
                    u[v_y_1_][i][j] = 0.;
                    u[v_z_1_][i][j] = 0.;
                    u[b_x_][i][j] = 0.75;
                    u[b_y_][i][j] = 1.;
                    u[b_z_][i][j] = 0.;

                    u[rho_2_][i][j] = 1.;
                    u[pre_2_][i][j] = 1.;
                    u[v_x_2_][i][j] = 0.;
                    u[v_y_2_][i][j] = 0.;
                    u[v_z_2_][i][j] = 0.;
                          }



                    if (x[i][j] >= 0.5)
                    {
                    u[rho_1_][i][j] = 0.125;
                    u[pre_1_][i][j] = 0.1;
                    u[v_x_1_][i][j] = 0.;
                    u[v_y_1_][i][j] = 0.;
                    u[v_z_1_][i][j] = 0.;
                    u[b_x_][i][j] = 0.75;
                    u[b_y_][i][j] = -1.;
                    u[b_z_][i][j] = 0.;

                    u[rho_2_][i][j] = 0.125;
                    u[pre_2_][i][j] = 0.1;
                    u[v_x_2_][i][j] = 0.;
                    u[v_y_2_][i][j] = 0.;
                    u[v_z_2_][i][j] = 0.;
                    }
                 }
           }


}
```

```
//i.c. for Kelvin Helmholtz instability
if (type == 4) {

        for (i = 0; i < n_x; i++)
           {
              for (j = 0 ; j< n_y ; j++)
                 {
                 pert= 0.06* sin(x[i][j]*12.56);

                    //if ((y[i][j] < 0.4 + pert) || (y[i][j] > 0.6 + pert)) //two interfaces
                       if (y[i][j] < 0.5 + pert) //one interface
                    {
                    u[rho_1_][i][j] = 1.;
                             u[v_x_1_][i][j] = 0.;
                             u[v_y_1_][i][j] = 0.;
                             u[v_z_1_][i][j] = 0.;
                    u[pre_1_][i][j] = 1.;
                             u[b_x_][i][j] = 0.;
                    u[b_y_][i][j] = 1.;
                    u[b_z_][i][j] = 0.;
                             u[rho_2_][i][j] = 1.;
                             u[v_x_2_][i][j] = 0.;
                             u[v_y_2_][i][j] = 0.;
                             u[v_z_2_][i][j] = 0.;
                    u[pre_2_][i][j] = 1.;



                    }

                    else
                    {
                    u[rho_1_][i][j] = 0.5;
                             u[v_x_1_][i][j] = -1.;
                             u[v_y_1_][i][j] = 0.;
                             u[v_z_1_][i][j] = 0.;
                    u[pre_1_][i][j] = 1.;
                             u[b_x_][i][j] = 0.;
                    u[b_y_][i][j] = 1.;
                    u[b_z_][i][j] = 0.;
                             u[rho_2_][i][j] = 0.5;
                             u[v_x_2_][i][j] = -1.;
                             u[v_y_2_][i][j] = 0.;
                             u[v_z_2_][i][j] = 0.;
                    u[pre_2_][i][j] = 1.;


                    }


/*u[rho_][i][j]=x[i][j];
u[pre_][i][j]=y[i][j];
*/
                        }
              }
```

```
}



//i.c. for Rayleigh Taylor instability
if (type == 5) {

                        double p00 =2.5;
                        double k =0.;
                        double rho1 = 1.;
                        double rho2 = 2.;
                        double g0 = -2.;
                        double p0 = p00 + rho2 * k *0.125;
                        double gg_ic[n_y];


//srand (time(NULL) + ind_cpu);
srand (ind_cpu);

        for (i = 0; i < n_x; i++)
           {
              for (j = 0 ; j < n_y ; j++)
                 {

                        //gg[i][j] = -1.; //const gravitational acceleration
                        //gg[i][j] = -1. * y[i][j]; //-1 gravitational acceleration
                        gg[i][j] = g0 + k * y[i][j]; //-2 gravitational acceleration
                 gg_ic[j] = -gg[i][j];


                    if (y[i][j] < 0.)
                    {
                    u[rho_1_][i][j] = rho1;
                              u[v_x_1_][i][j] = 0.;
                              u[v_y_1_][i][j] = 0.;
                              u[v_z_1_][i][j] = 0.;
                              u[b_x_][i][j] = 0.;
                    u[b_y_][i][j] = 1.;
                    u[b_z_][i][j] = 0.;
                              u[rho_2_][i][j] = rho1;
                              u[v_x_2_][i][j] = 0.;
                              u[v_y_2_][i][j] = 0.;
                              u[v_z_2_][i][j] = 0.;

                                   if (y[i][j] > -0.1)
                                   {
                                   u[v_y_1_][i][j] = 0.01 * (1. - 2. * rand() / (double)RAND_MAX);
                                   u[v_y_2_][i][j] = 0.01 * (1. - 2. * rand() / (double)RAND_MAX);
                                   }



                        u[pre_1_][i][j] = p0+ (-g0) * ( 0.5 * rho2  - rho1 * y[i][j]) - k * (0.125 * rho2 - 0.5 *
rho1 * y[i][j] * y[i][j]);
                        u[pre_2_][i][j] = p0+ (-g0) * ( 0.5 * rho2  - rho1 * y[i][j]) - k * (0.125 * rho2 - 0.5 *
rho1 * y[i][j] * y[i][j]);

                    }
```

```
                        else
                        {
                        u[rho_1_][i][j] = rho2;
                                u[v_x_1_][i][j] = 0.;
                                u[v_y_1_][i][j] = 0.;
                                u[v_z_1_][i][j] = 0.;
                                u[b_x_][i][j] = 0.;
                        u[b_y_][i][j] = 1.;
                        u[b_z_][i][j] = 0.;
                                u[rho_2_][i][j] = rho2;
                                u[v_x_2_][i][j] = 0.;
                                u[v_y_2_][i][j] = 0.;
                                u[v_z_2_][i][j] = 0.;
                                      if (y[i][j] < 0.1)
                                      {
                                      u[v_y_1_][i][j] = 0.01 * (1. - 2. * rand() / (double)RAND_MAX);
                                      u[v_y_2_][i][j] = 0.01 * (1. - 2. * rand() / (double)RAND_MAX);
                                      }

                        u[pre_1_][i][j] = p0 + (-g0) * rho2 * ( 0.5 - y[i][j]) - k * rho2 * (0.125 -0.5 * y[i][j] * y[i]
[j]);
                                u[pre_2_][i][j] = p0 + (-g0) * rho2 * ( 0.5 - y[i][j]) - k * rho2 * (0.125 -0.5 * y[i]
[j] * y[i][j]);
                        }
                         }
              }


}




//i.c. for Orszag Tang vortex test

if (type == 6) {

double beta, M, v0, B0, p0;

beta = 10/3.;
M = 1.;
v0 = 1.;
B0 = -(1./sqrt(4.*pi));
p0 = beta * (B0*B0)/2.;
//T0 = 1.0e-6;

          for (i = 0; i < n_x; i++)
             {
                for (j = 0 ; j< n_y ; j++)
                   {

                         // Ryu1998 and Londrillo2000

                                u[rho_1_][i][j] = gam * p0;
                        u[pre_1_][i][j] = p0;
                                //u[rho_1_][i][j] = (25./36.) * pi;
                        //u[pre_1_][i][j] = (5./12.) * pi;
                        u[v_x_1_][i][j] = - sin(2.*pi*y[i][j]);
                        u[v_y_1_][i][j] = sin(2.*pi*x[i][j]);
```

```
                    u[v_z_1_][i][j] = 0.;
                    u[b_x_][i][j] = - B0 * sin(2.*pi*y[i][j]);
                    u[b_y_][i][j] = B0 * sin(4.*pi*x[i][j]);
                    u[b_z_][i][j] = 0.;

                    u[rho_2_][i][j] = gam * p0;
                    u[pre_2_][i][j] = p0;
                    u[v_x_2_][i][j] = 0.;
                    u[v_y_2_][i][j] = 0.;
                    u[v_z_2_][i][j] = 0.;

                    T0 = (u[pre_1_][i][j] + u[pre_2_][i][j]) / ((u[rho_1_][i][j] + u[rho_2_][i][j]) * Rg);
                }
            }

}
```

```
//i.c. for an explosion
if (type == 7)
{

double x0 = ende_x_global * 0.5;
double y0 = ende_y_global * 0.5;


        for (i = 0; i < n_x; i++)
            {
                for (j = 0 ; j< n_y ; j++)
                    {

                    u[rho_1_][i][j] = 0.01;
                    u[pre_1_][i][j] = 0.01;
                    u[v_x_1_][i][j] = 0.;
                    u[v_y_1_][i][j] = 0.;
                    u[v_z_1_][i][j] = 0.;
                    u[b_x_][i][j] = 0.;
                    u[b_y_][i][j] = 0.;
                    u[b_z_][i][j] = 0.;


                    u[rho_2_][i][j] = 1.;
                    u[pre_2_][i][j] = 1. + exp((-(x[i][j] - x0)*(x[i][j] - x0) - (y[i][j] - y0)*(y[i][j] - y0))/0.001) *
pow(10.0,6.0);
                    u[v_x_2_][i][j] = 0.;
                    u[v_y_2_][i][j] = 0.;
                    u[v_z_2_][i][j] = 0.;


                    }
            }

}
```

```
//i.c. for flow with obstacle
if (type == 8)
{

        for (i = 0; i < n_x; i++)
            {
                for (j = 0 ; j< n_y ; j++)
                    {
                        if ((y[i][j] <= 0.5) && (x[i][j] <= 0.5) && (x[i][j] >= 0.3))
                        {
                        u[rho_1_][i][j] = 10.;
                        u[pre_1_][i][j] = 1.;
                        u[v_x_1_][i][j] = 0.;
                        u[v_y_1_][i][j] = 0.;
                        u[v_z_1_][i][j] = 0.;
                        u[b_x_][i][j] = 0.;
                        u[b_y_][i][j] = 0.;
                        u[b_z_][i][j] = 0.;


                        u[rho_2_][i][j] = 10.;
                        u[pre_2_][i][j] = 1.;
                        u[v_x_2_][i][j] = 0.;
                        u[v_y_2_][i][j] = 0.;
                        u[v_z_2_][i][j] = 0.;
                        }


                        else
                        {
                        u[rho_1_][i][j] = 0.01;
                        u[pre_1_][i][j] = 1.;
                        u[v_x_1_][i][j] = 1.;
                        u[v_y_1_][i][j] = 0.;
                        u[v_z_1_][i][j] = 0.;
                        u[b_x_][i][j] = 0.;
                        u[b_y_][i][j] = 0.;
                        u[b_z_][i][j] = 0.;


                        u[rho_2_][i][j] = 0.01;
                        u[pre_2_][i][j] = 1.;
                        u[v_x_2_][i][j] = 1.;
                        u[v_y_2_][i][j] = 0.;
                        u[v_z_2_][i][j] = 0.;
                        }


                    }
            }

}

// ------------------------------------------------------------------------

// ------------------------------------------------------------------------
```

```
//solar atmosphere

//reading in the VAL C model
//create new arrays


// ------------------------------------------------------------------------
if (type == 10) {

std::string File;

File = "./initmod_hs.dat";
std::ifstream FileStream;

FileStream.open(File, std::ios::in);
if (! FileStream) std::cout << "unable to open file for reading" << std::endl;


int row;

//read in the first line of the file which contains the number of elements or rows
FileStream >> row;



double *rho, *z, *int_e;

rho = new double[row];
z = new double[row];
int_e = new double[row];


for (i = row-1; i >= 0; i--)
{

  FileStream >> z[i];
  FileStream >> rho[i];
  FileStream >> int_e[i];

  z[i] = -z[i];
  //internal energy per unit volume
  //int_e[i] = int_e[i] * rho[i];  //not for initmod data

}
FileStream.close();



//fill y[][] and u[][][] arrays with interpolated values

//z or y axis
//1. find dz

double dz, index;
int int_index, next_index;

dz = z[1] - z[0];
```

```
std::cout << "dz = " << dz << std::endl;


for (i = 0; i < n_x; i++)
{
    for (j = 0; j < n_y; j++)
    {

        gg[i][j] = -27400.0;
        // index of y on z axis

            index = (y[i][j] - z[0]) / dz;
            //std::cout << index << std::endl;
            int_index = index; //no need to cast
            next_index = int_index + 1;

            //interpolate the data
            u[rho_1_][i][j] = 0.01*((rho[next_index] - rho[int_index]) * (index-int_index) + rho[int_index]);
            u[pre_1_][i][j] = 0.01*(((int_e[next_index] - int_e[int_index]) * (index-int_index) +
int_e[int_index]) * (gam - 1.));

            u[rho_2_][i][j] = 0.99*((rho[next_index] - rho[int_index]) * (index-int_index) + rho[int_index]);
            u[pre_2_][i][j] = 0.99*(((int_e[next_index] - int_e[int_index]) * (index-int_index) +
int_e[int_index]) * (gam - 1.));


                u[v_x_1_][i][j] = 0.0;
                u[v_y_1_][i][j] = 0.0;
                u[v_z_1_][i][j] = 0.0;

                u[v_x_2_][i][j] = 0.0;
                u[v_y_2_][i][j] = 0.0;
                u[v_z_2_][i][j] = 0.0;

                u[b_x_][i][j] = 50.;
                u[b_y_][i][j] = 85.;
                u[b_z_][i][j] = 0.0;


    }
}




//deallocate memory

delete [] rho;
delete [] z;
delete [] int_e;

}



// ------------------------------------------------------------------------
//Hillier
if (type == 11) {
```

```
   double B0, zeta_1, zeta_2, rho_tot, P_tot;

   rho_tot = 1.;
   P_tot = 0.15;
   B0 = 1.;
   zeta_1 = 0.1;
   zeta_2 = 0.9;



    for (i = 0; i < n_x; i++)
            {
               for (j = 0 ; j< n_y ; j++)
                  {

                   if (x[i][j] <= 0.)
                     {
                        u[b_y_][i][j] = B0;
                     }

                     else
                     {
                        u[b_y_][i][j] = -B0;
                     }



                     u[rho_1_][i][j] = zeta_1 * rho_tot;
                     u[pre_1_][i][j] = ((2. * zeta_1) / (zeta_2 + 2. * zeta_1)) * P_tot;
                     u[v_x_1_][i][j] = 0.;
                     u[v_y_1_][i][j] = 0.;
                     u[v_z_1_][i][j] = 0.;
                     u[b_x_][i][j] = 0.3 * B0;
                     u[b_z_][i][j] = 0.;


                     u[rho_2_][i][j] = zeta_2 * rho_tot;
                     u[pre_2_][i][j] = (zeta_2 / (zeta_2 + 2. * zeta_1)) * P_tot;
                     u[v_x_2_][i][j] = 0.;
                     u[v_y_2_][i][j] = 0.;
                     u[v_z_2_][i][j] = 0.;

         u[tem_1_][i][j] = u[pre_1_][i][j]/ (2. * Rg * u[rho_1_][i][j]); //temperature MHD / charged fluid
         u[tem_2_][i][j] = u[pre_2_][i][j] / (Rg * u[rho_2_][i][j]); //temperature HD / neutral fluid


                        //calculate ambient temperature:
                  T0 = (u[pre_1_][i][j] + u[pre_2_][i][j]) / ((u[rho_1_][i][j] + u[rho_2_][i][j]) * Rg); //case 3

                  }
               }


}


//end
}
```

APPLY_BC.CPP

```
#include <iostream>
#include <cmath>
#include "constants.h"
#include "mpi.h"



int ind_cpu(int i_cpu_x, int i_cpu_y)
{
return i_cpu_x + n_cpu_x * i_cpu_y;
}



void get_xyindex(int ind_cpu, int *i_cpu_x, int *i_cpu_y)
{
*i_cpu_y =  ind_cpu / n_cpu_x;
*i_cpu_x =  ind_cpu - n_cpu_x * *i_cpu_y;
}



//index of neighbouring cpu

int neighbour(int i_cpu_x, int i_cpu_y, int dir, int boundary) //neighbour: 0 = left or bottom, 1 = right
or top
{
   int neighbour_x, neighbour_y, ind_neighbour;
  //check direction

   if((dir == 0) && (boundary == 0)) //x direction, left neighbour
   {
      neighbour_x = i_cpu_x -1;
      if (neighbour_x == -1) neighbour_x = n_cpu_x -1; //closes the circut in a row / column
      neighbour_y = i_cpu_y; //upper and lower cpu index stays the same
   }

   if((dir == 0) && (boundary == 1)) //x direction, right
   {
      neighbour_x = i_cpu_x +1;
      if (neighbour_x == n_cpu_x) neighbour_x = 0;
      neighbour_y = i_cpu_y;
   }

   if((dir == 1) && (boundary == 0)) //y direction, bottom
   {
      neighbour_y = i_cpu_y -1;
      if (neighbour_y == -1) neighbour_y = n_cpu_y -1;
      neighbour_x = i_cpu_x;
   }

   if((dir == 1) && (boundary == 1)) //y direction, top
   {
      neighbour_y = i_cpu_y + 1;
      if (neighbour_y == n_cpu_y) neighbour_y = 0;
      neighbour_x = i_cpu_x;
```

```
    }

    ind_neighbour = neighbour_x + n_cpu_x * neighbour_y;
    return ind_neighbour;


  }



int is_inner(int i_cpu_x, int i_cpu_y, int i_dim, int i_boundary) //to check if a cpu's boundary is a
global boundary
{ //return 0 if it is a global boundary
    //0 = no inner, hence global boundary

    if ((i_dim == 0) && (i_cpu_x == 0) && (i_boundary == 0)) //x direction, first cpu, left boundary
    {
        return 0;
    }

    if((i_dim == 0) && (i_cpu_x == n_cpu_x -1) && (i_boundary == 1))
    {
        return 0;
    }

    if((i_dim == 1) && (i_cpu_y == 0) && (i_boundary == 0))
    {
            return 0;
    }

    if((i_dim == 1) && (i_cpu_y == n_cpu_y -1) && (i_boundary == 1))
    {
            return 0;
    }
        else return 1;
}



//-------------------------------------------------------------------------------------------------------------
------------------


void apply_BC(double*** u0, double*** ufix)
{


    //0 - periodic
    //1 - continuous
    //2 - fixed

    int i_var, i_dim, i_boundary, i_ghost, i_run, ind_from_x, ind_to_x,ind_from_y,
            ind_to_y, run_ind, i_buffer, i_boundary_to, i_boundary_from;

    int ind_cpu;
    int i_cpu_x, i_cpu_y;
```

```
    int ind_neighbour_to, ind_neighbour_from;

    int bc_type[n_var][n_dim][2]; //variables, dimension 0 = x-direction/ 1 =y-direction, boundary 0 =
bottom/left - 1 = top/right




    int is_inner(int i_cpu_x, int i_cpu_y, int i_dim, int i_boundary); //declare function to check if it is a
global boundary

    MPI_Status status;


//call index functions


MPI_Comm_rank(MPI_COMM_WORLD, &ind_cpu); // get the index of the cpu
//std::cout << "in apply_BC : ind_cpu: " << ind_cpu << std::endl;
get_xyindex(ind_cpu, &i_cpu_x, &i_cpu_y);  // get the i_cpu_x and i_cpu_y or the index of the cpu
in  x and y direction
//std::cout << "I_CPU_X, I_CPU_Y = " << i_cpu_x << " " << i_cpu_y << std::endl;

    bc_type[rho_1_][0][0] = 0; //left
    bc_type[rho_1_][0][1] = 0; //right
    bc_type[rho_1_][1][0] = 0; //bottom //2 RTI
    bc_type[rho_1_][1][1] = 0; //top     //2 RTI

    bc_type[mom_x_1_][0][0] = 0;
    bc_type[mom_x_1_][0][1] = 0;
    bc_type[mom_x_1_][1][0] = 0;
    bc_type[mom_x_1_][1][1] = 0;

    bc_type[mom_y_1_][0][0] = 0;
    bc_type[mom_y_1_][0][1] = 0;
    bc_type[mom_y_1_][1][0] = 0;
    bc_type[mom_y_1_][1][1] = 0;

    bc_type[mom_z_1_][0][0] = 0;
    bc_type[mom_z_1_][0][1] = 0;
    bc_type[mom_z_1_][1][0] = 0;
    bc_type[mom_z_1_][1][1] = 0;

    bc_type[ene_1_][0][0] = 0;
    bc_type[ene_1_][0][1] = 0;
    bc_type[ene_1_][1][0] = 0; //2rti
    bc_type[ene_1_][1][1] = 0; //2rti

    bc_type[b_x_][0][0] = 0;
    bc_type[b_x_][0][1] = 0;
    bc_type[b_x_][1][0] = 0;
    bc_type[b_x_][1][1] = 0;

    bc_type[b_y_][0][0] = 0;
    bc_type[b_y_][0][1] = 0;
    bc_type[b_y_][1][0] = 0;
    bc_type[b_y_][1][1] = 0;

    bc_type[b_z_][0][0] = 0;
```

```
    bc_type[b_z_][0][1] = 0;
    bc_type[b_z_][1][0] = 0;
    bc_type[b_z_][1][1] = 0;

    bc_type[psi_][0][0] = 0;
    bc_type[psi_][0][1] = 0;
    bc_type[psi_][1][0] = 0;
    bc_type[psi_][1][1] = 0;

//-------------------------------------

    bc_type[rho_2_][0][0] = 0; //left
    bc_type[rho_2_][0][1] = 0; //right
    bc_type[rho_2_][1][0] = 0; //bottom
    bc_type[rho_2_][1][1] = 0; //top

    bc_type[mom_x_2_][0][0] = 0;
    bc_type[mom_x_2_][0][1] = 0;
    bc_type[mom_x_2_][1][0] = 0;
    bc_type[mom_x_2_][1][1] = 0;

    bc_type[mom_y_2_][0][0] = 0;
    bc_type[mom_y_2_][0][1] = 0;
    bc_type[mom_y_2_][1][0] = 0;
    bc_type[mom_y_2_][1][1] = 0;

    bc_type[mom_z_2_][0][0] = 0;
    bc_type[mom_z_2_][0][1] = 0;
    bc_type[mom_z_2_][1][0] = 0;
    bc_type[mom_z_2_][1][1] = 0;

    bc_type[ene_2_][0][0] = 0;
    bc_type[ene_2_][0][1] = 0;
    bc_type[ene_2_][1][0] = 0;
    bc_type[ene_2_][1][1] = 0;
```

```
  for (i_var = 0; i_var < n_consVar; i_var++ )
   {
      for( i_dim = 0; i_dim < n_dim; i_dim++)
      {
        if(i_dim == 0) //x direction
          {
           run_ind = n_y -1;
          }

        if(i_dim == 1) //y direction
          {
           run_ind = n_x -1;
          }
```

```
//creating buffers
int buffer_size=(run_ind + 1) * n_ghost; //+1 because run_ind = n_x -1
double* sendbuffer = new double[buffer_size];
 double* recvbuffer = new double[buffer_size];



    for (i_boundary = 0; i_boundary <= 1 ; i_boundary ++)
    {
       for (i_ghost = 0; i_ghost < n_ghost; i_ghost++)
       {
           for (i_run = 0; i_run <= run_ind; i_run++)
          {



            if (i_dim == 0 ) // if in x direction
              {
                 ind_from_y = i_run;
                    if (i_boundary == 0) // left boundary
                    {
                       ind_from_x = n_ghost + i_ghost;
                    }
                    if (i_boundary == 1) //right boundary
                    {
                       ind_from_x = n_x - 2 * n_ghost + i_ghost;
                    }


              }

            if (i_dim == 1 ) //y direction
             {
                ind_from_x = i_run;

                    if (i_boundary == 0) // bottom boundary
                    {
                       ind_from_y = n_ghost + i_ghost;
                    }
                    if (i_boundary == 1) //top boundary
                    {
                       ind_from_y = n_y - 2 * n_ghost + i_ghost;
                    }


              }



              //construct sendbuffer
           i_buffer = i_run + i_ghost * run_ind; //index of my buffer
         //filling sendbuffer
           sendbuffer[i_buffer]= u0[i_var][ind_from_x][ind_from_y];


          } //i run
       } // i_ghost
```

```
                //close loop to fully create sendbuffer to be able to call mpi_sendrecv and send it
to neighbour and receive from neighbour



                i_boundary_from = i_boundary; //send from this boundary
                i_boundary_to = 1 - i_boundary_from;    //opposite boundary - receive into this
boundary

                ind_neighbour_to = neighbour(i_cpu_x,i_cpu_y,i_dim,i_boundary_from);  //
neighbour of the boundary we are at (i_boundary_from), so the one we send it to
                        ind_neighbour_from = neighbour(i_cpu_x,i_cpu_y,i_dim,i_boundary_to);  //
neighbour of the opposite boundary (i_boundaru_to), the one we get the boundary from: if we are
at left boundary we receive from the RIGHT NEIGHBOUR'S inner cells at left boundary

                        //call mpi_sendrecv to send the buffer to and receive one from (for
parameter details check MPI_sendrecv example online)
                MPI_Sendrecv(sendbuffer, buffer_size, MPI_DOUBLE, ind_neighbour_to, 10,
                        recvbuffer, buffer_size, MPI_DOUBLE, ind_neighbour_from, 10,
                MPI_COMM_WORLD, &status);

                        //MPI_Barrier(MPI_COMM_WORLD);


                //unpack recvbuffer and fill boundaries




        for (i_ghost = 0; i_ghost < n_ghost; i_ghost++)
        {
            for (i_run = 0; i_run <= run_ind; i_run++)
           {

                if (i_dim ==0 ) // if in x direction
                {
                    ind_to_y = i_run;

                        if (i_boundary == 1)
                       {
                            ind_to_x = 0 + i_ghost;
                        }
                        if (i_boundary == 0)
                            {
                            ind_to_x = n_x - 1 * n_ghost + i_ghost;
                        }


                }

               if (i_dim == 1 ) //y direction
                {
                   ind_to_x = i_run;

                        if (i_boundary == 1)
                        {
                            ind_to_y = 0 + i_ghost;
                        }
                        if (i_boundary == 0)
```

```
                    {
                        ind_to_y = n_y - 1 * n_ghost + i_ghost;
                    }

                }

                i_buffer = i_run + i_ghost * run_ind; //index of my buffer

                        u0[i_var][ind_to_x][ind_to_y] = recvbuffer[i_buffer];



            } //i_run


        } //i_ghost



    } //i_boundary

// delete memory for buffers

delete [] sendbuffer;
delete [] recvbuffer;
    } //i_dim



} //i_var


for (i_var = 0; i_var < n_consVar; i_var++ )
{
    for( i_dim = 0; i_dim < n_dim; i_dim++)
    {
        if(i_dim == 0) //x direction
                {
                        run_ind = n_y -1;
                }

        if(i_dim == 1) //y direction
                {
                        run_ind = n_x -1;
                }


        for (i_boundary = 0; i_boundary <= 1 ; i_boundary ++)
                {
                for (i_ghost = 0; i_ghost < n_ghost; i_ghost++)
                {
                 for (i_run = 0; i_run <= run_ind; i_run++)
```

```
                {


//if outer boundary (= is_inner==0), apply global boundary conditions
                if (is_inner(i_cpu_x, i_cpu_y, i_dim, i_boundary) == 0)
                { //otherwise, periodic boundary conditions are applied

                  //"preparations" for continuous BC:
                        if (i_dim == 0 ) // if in x direction
                        {
                                ind_to_y = i_run;
                                ind_from_y = i_run;

                        if (i_boundary == 0)
                        {
                                        ind_to_x = 0 + i_ghost;
                                                ind_from_x = n_ghost;// + i_ghost;

                        }

                                        if (i_boundary == 1)
                        {
                                        ind_to_x = n_x - 1 * n_ghost + i_ghost;
                                                ind_from_x = n_x - 2* n_ghost + 1; // + i_ghost;
                        }

                      }



                        if (i_dim == 1 ) // if in y direction
                        {
                                ind_to_x = i_run;
                                        ind_from_x = i_run;

                        if (i_boundary == 0)
                        {
                                        ind_to_y = 0 + i_ghost;
                                                ind_from_y = n_ghost;// + i_ghost;

                        }
                        if (i_boundary == 1)
                        {
                                        ind_to_y = n_y - 1 * n_ghost + i_ghost;
                                                ind_from_y = n_y - 2 * n_ghost + 1; // + i_ghost;
                        }


                      }

                       //apply continuous boundary conditions
                    if (bc_type[i_var][i_dim][i_boundary] == 1)
                     {

                                u0[i_var][ind_to_x][ind_to_y] = u0[i_var][ind_from_x][ind_from_y];

                          }

                    //apply global fixed boundary conditions
                    if (bc_type[i_var][i_dim][i_boundary] == 2)
```

```
                    {
                        u0[i_var][ind_to_x][ind_to_y] = ufix[i_var][ind_to_x][ind_to_y];
                    }

                    //apply global zero value boundary conditions
                    if (bc_type[i_var][i_dim][i_boundary] == 3)
                    {
                        u0[i_var][ind_to_x][ind_to_y] = 0.;
                    }


                } //is_inner


            } //i_run


        } //i_ghost



    } //i_boundary

} //i_dim


} //i_var



}
```

MAIN.CPP

```
#include <cstdlib>
#include <cmath>
#include <array>
#include <locale>
#include <iostream>

#include <string.h>
#include <fstream>
#include <sstream>

#include"constants.h"
#include"mpi.h"
#include<stdio.h>

using namespace std;


double ***f, ***res, ***u0, ***ufix;
double ****nflux;

double **x, **y, **gg;

int i, j, k, i_t;


double dtime, timep, a, d_xy, vmax, ch, cp; //a = coefficient for timestepping in Runge Kutta
double T0=0.; //ambient temperature for Hillier simulation

int numprocs, ind_cpu; //number of processors and rank


int main(int argc, char** argv) {

    //declare functions
    void alloc_mem_arr2D(double** &);
    void alloc_mem_arr3D(double*** &);
    void alloc_mem_arr4D(double**** &);
    void initial_condition(double*** , double** , double**, double**);
    void prim_to_con(double***);
    void con_to_prim(double***);
    void calc_rates(double ***);
    void apply_BC(double***, double***);

    void calc_residuals(double**, double**, double***, double***, double****);
    double timestep(double***);

    void dealloc_mem_arr2D(double**);
    void dealloc_mem_arr3D(double***);
    void dealloc_mem_arr4D(double****);

    void apply_RK4(double **, double **, double **, double ***, double ***, double ***, double ****,
double, double);
    void apply_RK1(double **, double **, double **, double ***, double ***, double ***, double ****,
double, double);
    void apply_RK2(double **, double **, double **, double ***, double ***, double ***, double ****,
double, double);

    void save_binary(int, double***, double** ,double**);
```

```
    int density_barrier(double***);
    void calc_temperature(double ***u0);

    //MPI

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs); //get the number of processors and
assign it to numprocs
    MPI_Comm_rank(MPI_COMM_WORLD, &ind_cpu); //get the number of rank (=cpu) and assign
it to myid



    if (numprocs != n_cpu_x * n_cpu_y)
 {
    std::cout<<"number of processors does not match"<<std::endl;
    abort();
  }


    alloc_mem_arr2D(x);
    alloc_mem_arr2D(y);
    alloc_mem_arr2D(gg);
    alloc_mem_arr3D(u0);
    alloc_mem_arr3D(ufix);
    alloc_mem_arr3D(res);
    alloc_mem_arr4D(nflux);
    initial_condition(u0, x, y, gg);

    i_t = 0.;


    calc_temperature(u0);
    prim_to_con(u0);
    calc_rates(u0);
    save_binary(i_t, u0, x, y);



for (k = 0; k < n_var; k++)
    {
        for (i = 0; i < n_x; i++)
        {
            for (j = 0 ; j < n_y ; j++)
            {
                    ufix[k][i][j] = u0[k][i][j];
              }
            }
     }

 apply_BC(u0, ufix);

    timep = 0.; //start of physical time


    //start solving equations
    for (int i_t=1; i_t< n_t; i_t++)
    {
```

```
        calc_temperature(u0);

        con_to_prim(u0);

        dtime = timestep(u0);

        d_xy = fmin(dx,dy);

        //for hyperbolic div cleaning
        ch = 0.5*(CFL * d_xy / dtime);
        cp = 0.18*ch;


        apply_RK1(x, y, gg, u0, ufix, res, nflux, dtime, timep);


        //density_barrier(u0);

        save_binary(i_t, u0, x, y);


        timep=timep+dtime; //physical time

            //if (timep >= 1.) //Hillier
            if (timep >= 0.48) //OT Ryu
            {
                i_t = 100000;
                save_binary(i_t, u0, x, y);
                break;
            }



    }


        dealloc_mem_arr2D(x);
        dealloc_mem_arr2D(y);
        dealloc_mem_arr2D(gg);
        dealloc_mem_arr3D(u0);
        dealloc_mem_arr3D(ufix);
        dealloc_mem_arr3D(res);
        dealloc_mem_arr4D(nflux);

    MPI_Finalize();
    return 0;
}
```

RK.CPP

```
#include<iostream>
#include<cmath>
#include "constants.h"

void calc_residuals(double**, double**, double***, double***, double****);
void apply_BC(double***, double ***);
void calc_sources(double **, double **, double**, double ***, double ***, double, double);



void apply_RK1(double **x, double **y, double** gg, double ***u, double *** ufix, double ***res,
double ****nflux, double dtime, double timep)
{
   int i, j, k;

   apply_BC(u, ufix);

   calc_residuals(x, y, u, res, nflux);
   calc_sources(x, y, gg, u, res, timep, dtime);


   for (k = 0; k < n_consVar; k++)
   {
         for (i=n_ghost; i < n_x-n_ghost+1; i++)
         {
         for (j =n_ghost; j < n_y-n_ghost+1; j++)
         {
                  u[k][i][j] = u[k][i][j] + res[k][i][j] * dtime;
         }
         }
   }



}


//---------------------------------------------------------------------------------------
//---------------------------------------------------------------------------------------
//RK4

void apply_RK4(double **x, double **y, double** gg, double ***u, double *** ufix, double ***res,
double ****nflux, double dtime, double timep)
{
   int i, j, k;
   double ***u1, ***u2;

      u1 = new double**[n_var];
   for (i = 0; i < n_var; ++i)
   {
      u1[i] = new double*[n_x];
      for (j =0; j < n_x; j++)
         u1[i][j] = new double[n_y];
   }

         u2 = new double**[n_var];
   for (i = 0; i < n_var; ++i)
```

```
{
   u2[i] = new double*[n_x];
   for (j =0; j < n_x; j++)
      u2[i][j] = new double[n_y];
}

apply_BC(u, ufix);

//k1
calc_residuals(x, y, u, res, nflux);
calc_sources(x, y, gg, u, res, timep, dtime);


for (k = 0; k < n_consVar; k++)
{
      for (i=n_ghost; i < n_x-n_ghost+1; i++)
      {
      for (j =n_ghost; j < n_y-n_ghost+1; j++)
      {

res[k][i][j] = dtime * res[k][i][j];


u1[k][i][j] = u[k][i][j] + (1. / 6.) * res[k][i][j]; // RK update u
u2[k][i][j] = u[k][i][j] + res[k][i][j] * 0.5; //RK


            }
      }
}


apply_BC(u2, ufix);

//k2
calc_residuals(x, y, u2, res, nflux);
calc_sources(x, y, gg, u2, res, timep, dtime);


   for (k=0; k<n_consVar; k++)
{
      for (i=n_ghost; i < n_x-n_ghost+1; i++)
      {
      for (j =n_ghost; j < n_y-n_ghost+1; j++)
      {

res[k][i][j] = dtime * res[k][i][j];
u1[k][i][j] = u1[k][i][j] + (1. / 3.) * res[k][i][j]; // update u RK
u2[k][i][j] = u[k][i][j] + res[k][i][j] * 0.5;
            }
      }
}


apply_BC(u2, ufix);

//k3
calc_residuals(x, y, u2, res, nflux);
calc_sources(x, y, gg, u2, res, timep, dtime);
```

```
    for (k=0; k<n_consVar; k++)
    {
          for (i=n_ghost; i < n_x-n_ghost+1; i++)
          {
          for (j =n_ghost; j < n_y-n_ghost+1; j++)
          {

    res[k][i][j] = dtime * res[k][i][j];
    u1[k][i][j] = u1[k][i][j] + (1. / 3.) * res[k][i][j]; // update u
    u2[k][i][j] = u[k][i][j] + res[k][i][j];
                      }
          }
    }


    apply_BC(u2, ufix);

    //k4
    calc_residuals(x, y, u2, res, nflux);
    calc_sources(x, y, gg, u2, res, timep, dtime);


    for (k=0; k<n_consVar; k++)
    {
          for (i=n_ghost; i < n_x-n_ghost+1; i++)
          {
          for (j =n_ghost; j < n_y-n_ghost+1; j++)
          {

    res[k][i][j] = dtime * res[k][i][j];
    u[k][i][j] = u1[k][i][j] + (1. / 6.) * res[k][i][j]; // update u = advanced solution

                }
          }
    }

    for (i = 0; i < n_var; ++i)
       {
          for (j =0; j < n_x; j++)
          {
             delete [] u1[i][j];
          }
          delete [] u1[i];
       }
         delete [] u1;

    for (i = 0; i < n_var; ++i)
       {
          for (j =0; j < n_x; j++)
          {
             delete [] u2[i][j];
          }
          delete [] u2[i];
       }
         delete [] u2;

    return;
}
```

CALC_RESIDUALS.CPP

```
/*
 * function to calculate residuals
 */

#include<iostream>
#include<cmath>
#include"constants.h"


void calc_residuals(double** x, double** y, double*** u0, double*** res, double**** nflux)
{

void numerical_flux(int, double*, double*, double*, double*, double*);

int i, j, k;


double ullx[n_var], ulx[n_var], uux[n_var] ,urx[n_var];
double ully[n_var], uly[n_var], uuy[n_var] ,ury[n_var];

double flux1[n_var], flux2[n_var];

for (i=n_ghost; i <n_x-n_ghost+1; i++)
{
   for (j =n_ghost; j < n_y-n_ghost+1; j++)
   {

     for (k = 0; k < n_var; k++)

         {
            //in x direction
            ullx[k]=u0[k][i-2][j] ;
            ulx[k]=u0[k][i-1][j];
            uux[k]=u0[k][i][j];
            urx[k]=u0[k][i+1][j];

            // in y direction
            ully[k]=u0[k][i][j-2] ;
            uly[k]=u0[k][i][j-1];
            uuy[k]=u0[k][i][j];
            ury[k]=u0[k][i][j+1];

         }

//call numerical flux function in x and y direction and fill array flux1 and flux2, respectively
       numerical_flux(0,ullx,ulx,uux,urx,flux1);
       numerical_flux(1,ully,uly,uuy,ury,flux2);


      for (k=0; k < n_consVar; k++)
         {

            nflux[0][k][i][j]=flux1[k];
            nflux[1][k][i][j]=flux2[k];
         }
   }
```

```
 }

//2) calculate residuals based on numerical flux and fill the array res[][]

for (k = 0; k < n_consVar; k++)
{
   for (i=n_ghost; i <n_x-n_ghost+1; i++)
   {
      for (j =n_ghost; j < n_y-n_ghost+1; j++)
      {

         res[k][i][j] = -((nflux[0][k][i+1][j]-nflux[0][k][i][j])/(x[i+1][j]-x[i][j]) // 0- x direction
                     +(nflux[1][k][i][j+1]-nflux[1][k][i][j])/(y[i][j+1]-y[i][j]));        // 1 - y direction

      }
   }
}


}
```

NUMERICAL_FLUX.CPP

```
/*here included are:
 * limiter function
 * numerical flux function
 */


#include<iostream>
#include<cmath>

#include"constants.h"


//limiter function
double limiter(double r)
    {

   //return (1.50*(r*r+r)/(r*r+r+1.0)); //ospre limiter
   return (fmax(0, fmin(1, r))); //Minmod

}



//numerical flux function
void numerical_flux(int dir, double* ull, double* ul, double* uu, double* ur, double* flux)
{
double limiter(double r);

void get_pflux(int, double*, double*);

    double ru[n_var]; //size of all variables, but further down, only loop through n_consVar
    double rl[n_var];

    double ulmh[n_var];
    double urmh[n_var];
    double fll[n_var];
    double flr[n_var];

    // limited interpolated values of the state vector at the cell centres
    double nom1, denom1, nom2, denom2;
    double pl_1, pr_1, pl_2, pr_2, csl_1, csr_1, csl_2, csr_2;
    double aaa_1, aaa1_1, aaa2_1, aaa_2, aaa1_2, aaa2_2, aaa2, result, B_tot_l, B_tot_r, va_l, va_r;

    int k;


for (k = 0; k < n_consVar; k++)
    {

        nom1 = (uu[k] - ul[k]);
        denom1 = (ur[k] - uu[k]);


         if (fabs(nom1) < 1.0e-14)
            {
            nom1 = 0.;
            denom1 = 1.0;
            }
```

```
        if ((nom1 > 1.0e-14) && (fabs(denom1) < 1.0e-14))
           {
           nom1 = 1.0e14;
           denom1 = 1.0;
           }

        if ((nom1 < -1.0e-14) && (fabs(denom1) < 1.0e-14))
           {
           nom1 = -1.0e14;
           denom1 = 1.0;
           }

        ru[k]= nom1/denom1; //ratio of the gradients


        nom2 = ul[k]-ull[k];
        denom2 = uu[k]-ul[k];

        // ; this part is just to check that we don't divide by 0
        if (fabs(nom2) < 1.0e-14)
        {
           nom2 = 0.0;
           denom2 = 1.0;
        }

        if ((nom2 > 1.0e-14) && (fabs(denom2) < 1.0e-14))
        {
           nom2 = 1.0e14;
           denom2 = 1.0;
        }


        if ((nom2 < -1.0e-14) && (fabs(denom2) < 1.0e-14))
        {
           nom2 = -1.0e14;
           denom2 = 1.0;
        }

        //checking ends here

        rl[k] = nom2 / denom2; //=ratio of the gradients

     }

   for (k = 0; k < n_consVar; k++)

   {
     //call function to calculate the limit for each variable
     ulmh[k] = ul[k] + 0.50 * limiter(rl[k]) * (uu[k] - ul[k]);
     urmh[k] = uu[k] - 0.50 * limiter(ru[k]) * (ur[k] - uu[k]);
   }



  //check
  B_tot_l = ulmh[b_x_] * ulmh[b_x_] + ulmh[b_y_] * ulmh[b_y_] + ulmh[b_z_] * ulmh[b_z_];
  B_tot_r = urmh[b_x_] * urmh[b_x_] + urmh[b_y_] * urmh[b_y_] + urmh[b_z_] * urmh[b_z_];
```

```
//pressure left and right
pl_1 = (ulmh[ene_1_] - 0.5 * ((ulmh[mom_x_1_] * ulmh[mom_x_1_]
                        + ulmh[mom_y_1_] * ulmh[mom_y_1_]
                        + ulmh[mom_z_1_] * ulmh[mom_z_1_]) / ulmh[rho_1_] + B_tot_l)) * (gam - 1.0) ; //
total ene - kin ene in x and y

pr_1 = (urmh[ene_1_] - 0.5 * ((urmh[mom_x_1_] * urmh[mom_x_1_]
                        + urmh[mom_y_1_] * urmh[mom_y_1_]
                        + urmh[mom_z_1_] * urmh[mom_z_1_]) / urmh[rho_1_] + B_tot_r)) * (gam - 1.0) ;



//pressure left and right HYDRO
pl_2 =(ulmh[ene_2_] - 0.5 * (ulmh[mom_x_2_] * ulmh[mom_x_2_]
                      + ulmh[mom_y_2_] * ulmh[mom_y_2_]
                      + ulmh[mom_z_2_] * ulmh[mom_z_2_]) / ulmh[rho_2_]) * (gam - 1.0); //total ene -
kin ene in x and y

pr_2 =(urmh[ene_2_] - 0.5 * (urmh[mom_x_2_] * urmh[mom_x_2_]
                      + urmh[mom_y_2_] * urmh[mom_y_2_]
                      + urmh[mom_z_2_] * urmh[mom_z_2_]) / urmh[rho_2_]) * (gam - 1.0); // divided
my rho because mom*mom




//speed of sound left and right MHD
csl_1 = 1.0 * sqrt(gam * pl_1 / ulmh[rho_1_]);
csr_1 = 1.0 * sqrt(gam * pr_1 / urmh[rho_1_]);

va_l =  sqrt(B_tot_l / (2. * ulmh[rho_1_]));
va_r =  sqrt(B_tot_r / (2. * urmh[rho_1_]));

csl_1 = sqrt( va_l * va_l + csl_1 * csl_1);
csr_1 = sqrt( va_r * va_r + csr_1 * csr_1);

//speed of sound left and right HYDRO
csl_2 = 1.0 * sqrt(gam * pl_2 / ulmh[rho_2_]);
csr_2 = 1.0 * sqrt(gam * pr_2 / urmh[rho_2_]);




// take the absolute values and select the max value = aaa
aaa_1  = fmax(fabs(ulmh[mom_x_1_ + dir] / ulmh[rho_1_] + csl_1), fabs(urmh[mom_x_1_ + dir] /
urmh[rho_1_] + csr_1)); //max characteristic speed- y component +1
aaa1_1 = fmax(aaa_1, fabs(ulmh[mom_x_1_ + dir] / ulmh[rho_1_] - csl_1));
aaa2_1 = fmax(aaa1_1, fabs(urmh[mom_x_1_+ dir] / urmh[rho_1_] - csr_1));



// take the absolute values and select the max value = aaa HYDRO
aaa_2  = fmax(fabs(ulmh[mom_x_2_ + dir ] / ulmh[rho_2_] + csl_2), fabs(urmh[mom_x_2_ + dir] /
urmh[rho_2_] + csr_2)); //max characteristic speed- y component +1
aaa1_2 = fmax(aaa_2, fabs(ulmh[mom_x_2_ + dir] / ulmh[rho_2_] - csl_2));
aaa2_2 = fmax(aaa1_2, fabs(urmh[mom_x_2_ + dir] / urmh[rho_2_] - csr_2));
```

```
aaa2 = fmax(aaa2_1, aaa2_2);


//calculate physical flux left and right
get_pflux(dir,urmh,flr);
get_pflux(dir,ulmh,fll);

for (k = 0; k < n_consVar; k++)
   {
      flux[k] = 0.5 * (flr[k] + fll[k] - aaa2 * (urmh[k] - ulmh[k]));
   }

}
```

GET_PFLUX.CPP

```
/*
 * calculate physical flux
 */

#include<iostream>
#include<cmath>
#include"constants.h"



void get_pflux(int direction, double* u0, double* fl)
{
  double v_x_1, v_y_1, v_z_1, B_x, B_y, B_z, B_tot, pre_1, vdotB;
  double v_x_2, v_y_2, v_z_2, pre_2;

   v_x_1 = u0[mom_x_1_] / u0[rho_1_];
   v_y_1 = u0[mom_y_1_] / u0[rho_1_];
   v_z_1 = u0[mom_z_1_] / u0[rho_1_];


   B_x = u0[b_x_];
   B_y = u0[b_y_];
   B_z = u0[b_z_];

   //hydro
   v_x_2 = u0[mom_x_2_] / u0[rho_2_];
   v_y_2 = u0[mom_y_2_] / u0[rho_2_];
   v_z_2 = u0[mom_z_2_] / u0[rho_2_];


   B_tot = B_x * B_x + B_y * B_y + B_z * B_z; //squared mag field vector = magnitude

   pre_1 = (u0[ene_1_] - 0.5 * ((u0[mom_x_1_] * v_x_1
                        + u0[mom_y_1_] * v_y_1
                        + u0[mom_z_1_] * v_z_1) + B_tot )) * (gam - 1.) + 0.5 * B_tot; //total e - ekin +
emag


   pre_2 =  (u0[ene_2_] - 0.5 * (u0[mom_x_2_] * v_x_2
                        + u0[mom_y_2_] * v_y_2
                        + u0[mom_z_2_] * v_z_2)) * (gam - 1.) ;

   vdotB = v_x_1 * B_x + v_y_1 * B_y + v_z_1 * B_z;


   if (direction == 0) //x-direction
   {

   fl[rho_1_] = u0[mom_x_1_];
   fl[mom_x_1_] = u0[mom_x_1_] * v_x_1 - B_x * B_x + pre_1;
   fl[mom_y_1_] = u0[mom_y_1_] * v_x_1 - B_x * B_y;
   fl[mom_z_1_] = u0[mom_z_1_] * v_x_1 - B_x * B_z;
   fl[ene_1_] = v_x_1 * (u0[ene_1_]+ pre_1) - B_x * vdotB;
   fl[b_x_] = 0. + u0[psi_];
   fl[b_y_] = (v_x_1 * B_y - v_y_1 * B_x) + u0[psi_];
   fl[b_z_] = -(v_z_1 * B_x - v_x_1 * B_z) + u0[psi_];
   fl[psi_] = (ch*ch) * B_x; //for hyperbolic divergence cleaning Dedner2002
```

```
//hydro
fl[rho_2_] = u0[mom_x_2_];
fl[mom_x_2_] = u0[mom_x_2_] * v_x_2 + pre_2;
fl[mom_y_2_] = u0[mom_y_2_] * v_x_2;
fl[mom_z_2_] = u0[mom_z_2_] * v_x_2;
//from energy equation
fl[ene_2_] = v_x_2 * (u0[ene_2_] + pre_2);


}



if (direction == 1) //y-direction
{
fl[rho_1_] = u0[mom_y_1_];

fl[mom_x_1_] = u0[mom_x_1_] * v_y_1 - B_y * B_x;
fl[mom_y_1_] = u0[mom_y_1_] * v_y_1 - B_y * B_y + pre_1 ;
fl[mom_z_1_] = u0[mom_z_1_] * v_y_1 - B_y * B_z;
fl[ene_1_] = v_y_1 *(u0[ene_1_] + pre_1) - B_y * vdotB;
fl[b_x_] = -(v_x_1 * B_y - v_y_1 * B_x) + u0[psi_];
fl[b_y_] = 0. + u0[psi_];
fl[b_z_] = (v_y_1 * B_z - v_z_1 * B_y) + u0[psi_];
fl[psi_] = (ch*ch) * B_y;

//hydro

fl[rho_2_] = u0[mom_y_2_];
fl[mom_x_2_] = u0[mom_x_2_] * v_y_2;
fl[mom_y_2_] = u0[mom_y_2_] * v_y_2 + pre_2;
fl[mom_z_2_] = u0[mom_z_2_] * v_y_2;
fl[ene_2_] = v_y_2 * (u0[ene_2_] + pre_2);

}




if (direction == 2) //z-direction
{
fl[rho_1_] = u0[mom_z_1_];

fl[mom_x_1_] = u0[mom_x_1_] * v_z_1 - B_z * B_x;
fl[mom_y_1_] = u0[mom_y_1_] * v_z_1 - B_z * B_y;
fl[mom_z_1_] = u0[mom_z_1_] * v_z_1 - B_z * B_z + pre_1 ;
fl[ene_1_] = v_z_1 *(u0[ene_1_] + pre_1 ) - B_z * vdotB;
fl[b_x_] = (v_z_1 * B_x - v_x_1 * B_z) + u0[psi_];
fl[b_y_] = -(v_y_1 * B_z - v_z_1 * B_y) + u0[psi_];
fl[b_z_] = 0. + u0[psi_];
fl[psi_] = (ch*ch) * B_z;

//hydro

fl[rho_2_] = u0[mom_y_2_];
fl[mom_x_2_] = u0[mom_x_2_] * v_z_2;
```

```
    fl[mom_y_2_] = u0[mom_y_2_] * v_z_2;
    fl[mom_z_2_] = u0[mom_z_2_] * v_z_2 + pre_2;
    fl[ene_2_] = v_z_2 * (u0[ene_2_] + pre_2);


    }




}
```

CALC_SOURCES

```
#include<iostream>
#include<cmath>
#include"constants.h"

void calc_rates(double ***);

void calc_temperature(double ***);

void calc_sources(double **x, double **y, double** gg, double ***u, double ***res, double timep,
double dtime)
{

    int i, j;
    double rho_change_1, rho_change_2, mom_x_change, mom_y_change, mom_z_change,
ene_change, amplitude, period, radius;
    double rho_change;
    double v2_tot_1, v2_tot_2;
    double v_x_1, v_y_1, v_z_1;
    double v_x_2, v_y_2, v_z_2;
    double t0, x0, a_col_0, B_tot;


    amplitude = 2e3;
    period = 180.;
    t0 = 4. * period;
    x0 = 0.5 * ende_x_global;
   // y0 = 0.;

/*

//date: 02.02.2018 added gravity sources (rho*g bzw rho*g.v)

          for (i=n_ghost; i < n_x-n_ghost+1; i++)
          {
                for (j =n_ghost; j < n_y-n_ghost+1; j++)
                {
                res[mom_y_1_][i][j] = res[mom_y_1_][i][j] + u[rho_1_][i][j] * gg[i][j];
                res[ene_1_][i][j] = res[ene_1_][i][j] + u[mom_y_1_][i][j] * gg[i][j];

                res[mom_y_2_][i][j] = res[mom_y_2_][i][j] + u[rho_2_][i][j] * gg[i][j];
                res[ene_2_][i][j] = res[ene_2_][i][j] + u[mom_y_2_][i][j] * gg[i][j];
                }
          }
 */


//first calculate rates for the source terms

    calc_temperature(u);

    calc_rates(u);

          for (i=n_ghost; i < n_x-n_ghost+1; i++)
          {
                for (j =n_ghost; j < n_y-n_ghost+1; j++)
                {
```

```
v_x_1 = u[mom_x_1_][i][j] / u[rho_1_][i][j];
v_y_1 = u[mom_y_1_][i][j] / u[rho_1_][i][j];
v_z_1 = u[mom_z_1_][i][j] / u[rho_1_][i][j];


//hydro
v_x_2 = u[mom_x_2_][i][j] / u[rho_2_][i][j];
v_y_2 = u[mom_y_2_][i][j] / u[rho_2_][i][j];
v_z_2 = u[mom_z_2_][i][j] / u[rho_2_][i][j];



v2_tot_1 = v_x_1 * v_x_1 + v_y_1 * v_y_1 + v_z_1 * v_z_1;
v2_tot_2 = v_x_2 * v_x_2 + v_y_2 * v_y_2 + v_z_2 * v_z_2;







//add hyperbolic Div cleaning Dedner2002

double divB, gradPsi_x, gradPsi_y, vdotB;
double v_x, v_y, v_z;
double rho_1_divB, mom_x_1_divB, mom_y_1_divB, mom_z_1_divB, b_x_1_divB,
b_y_1_divB, b_z_1_divB, ene_1_divB, psi_divB;
divB = 0.5 * ((u[b_x_][i+1][j] - u[b_x_][i-1][j])/(dx) + (u[b_y_][i][j+1] - u[b_y_][i][j-1])/(dy));
// divB = ((u[b_x_][i+1][j] + u[b_x_][i][j])/2.- (u[b_x_][i-1][j] +u[b_x_][i][j])/2.)/dx + ((u[b_y_][i]
[j+1] + u[b_y_][i][j])/2.- (u[b_y_][i][j-1] + u[b_y_][i][j])/2.)/dy;

gradPsi_x = 0.5 * ((u[psi_][i+1][j] - u[psi_][i-1][j])/(dx));
gradPsi_y = 0.5 * ((u[psi_][i][j+1] - u[psi_][i][j-1])/(dy));

v_x = u[mom_x_1_][i][j] / u[rho_1_][i][j];
v_y = u[mom_y_1_][i][j] / u[rho_1_][i][j];
v_z = u[mom_z_1_][i][j] / u[rho_1_][i][j];

vdotB = v_x * u[b_x_][i][j] + v_y * u[b_y_][i][j] + v_z * u[b_z_][i][j];
rho_1_divB = 0.;
mom_x_1_divB = -divB * u[b_x_][i][j];
mom_y_1_divB = -divB * u[b_y_][i][j];
mom_z_1_divB = -divB * u[b_z_][i][j];
b_x_1_divB = 0.;
b_y_1_divB = 0.;
b_z_1_divB = 0.;
ene_1_divB = (-u[b_x_][i][j] * gradPsi_x) + (-u[b_y_][i][j] * gradPsi_y);
psi_divB = -(ch*ch / cp*cp) * u[psi_][i][j];




//source terms

/*

rho_change = 0.;
mom_x_change = 0.;
 mom_y_change = 0.;
```

```
 mom_z_change = 0.;
 ene_change = 0.;
 */



rho_change =   u[a_ion_][i][j] * u[rho_2_][i][j] - u[a_rec_][i][j] * u[rho_1_][i][j];



 mom_x_change = u[f_col_][i][j] * u[rho_1_][i][j] * u[rho_2_][i][j] * (v_x_2 - v_x_1)
                                                                    - u[a_rec_][i][j] * u[mom_x_1_][i][j]
                                                                    + u[a_ion_][i][j] * u[mom_x_2_][i][j];
 mom_y_change = u[f_col_][i][j] * u[rho_1_][i][j] * u[rho_2_][i][j] * (v_y_2 - v_y_1)
                                                                    - u[a_rec_][i][j] * u[mom_y_1_][i][j]
                                                                    + u[a_ion_][i][j] * u[mom_y_2_][i][j];
 mom_z_change = u[f_col_][i][j] * u[rho_1_][i][j] * u[rho_2_][i][j] * (v_z_2 - v_z_1)
                                                                    - u[a_rec_][i][j] * u[mom_z_1_][i][j]
                                                                    + u[a_ion_][i][j] * u[mom_z_2_][i][j];

 ene_change = u[f_col_][i][j] * u[rho_1_][i][j] * u[rho_2_][i][j]
        * (0.5 * (v2_tot_2 - v2_tot_1) + 3 * Rg * (u[tem_2_][i][j]-u[tem_1_][i][j]))
            - 0.5 * u[a_rec_][i][j] * u[rho_1_][i][j] * v2_tot_1
            + 0.5 * u[a_ion_][i][j] * u[rho_2_][i][j] * v2_tot_2;









 /*
 //add an acoustic source

  radius = sqrt((x[i][j] - x0) * (x[i][j] - x0) + (y[i][j] - y0) * (y[i][j] - y0));


  mom_x_change = 0.;
  mom_y_change = 0.;
  mom_z_change = 0.;
  ene_change = amplitude * sin((timep * 2. * pi) / period) * exp(-((radius * radius) /
sigma2));
             * exp(-((timep - t0) * (timep-t0)) / ((1.*period) * (1.*period)));

  */



 // residuals plus source terms


 res[rho_1_][i][j] = res[rho_1_][i][j] + rho_change + rho_1_divB;
 res[rho_2_][i][j] = res[rho_2_][i][j] - rho_change;
```

```
                res[mom_x_1_][i][j] = res[mom_x_1_][i][j] + mom_x_change + mom_x_1_divB;
                res[mom_x_2_][i][j] = res[mom_x_2_][i][j] - mom_x_change;

                res[mom_y_1_][i][j] = res[mom_y_1_][i][j] + mom_y_change + mom_y_1_divB;
                res[mom_y_2_][i][j] = res[mom_y_2_][i][j] - mom_y_change;

                res[mom_z_1_][i][j] = res[mom_z_1_][i][j] + mom_z_change + mom_z_1_divB;
                res[mom_z_2_][i][j] = res[mom_z_2_][i][j] - mom_z_change;

                res[ene_1_][i][j] = res[ene_1_][i][j] + ene_change + ene_1_divB;
                res[ene_2_][i][j] = res[ene_2_][i][j] - ene_change;

                res[psi_][i][j] = res[psi_][i][j] + psi_divB;

                }
        }



}



void calc_rates(double ***u)
{

   /*
   double T_in, beta, Ionis_coeff, Rec_coeff;

   for (i=n_ghost; i < n_x-n_ghost+1; i++)
     {
         for (j =n_ghost; j < n_y-n_ghost+1; j++)

      {

            //new rates from Moore & Fung 1972
            T_in = (u[tem_1_][i][j] + u[tem_2_][i][j]) * 0.5;
            beta = 158000. / T_in;

            Ionis_coeff = 2.34e-8 / sqrt(beta) * exp(-beta);
            Rec_coeff = 5.2e-14 / sqrt(beta) * (0.4288 + 0.5 * log(beta) + 0.4698 /cbrt(beta));

            u[f_col_][i][j] = 1. / (m_1 + m_2) * Sig_in * sqrt(8 * k_B * T_in / pi * m_in);
            u[a_ion_][i][j] = u[rho_1_][i][j] / m_2 * Ionis_coeff;
            u[a_rec_][i][j] = u[rho_1_][i][j] / m_1 * Rec_coeff;

   */

//Hillier
   int i, j;
   double a_col_0, T_max;
   a_col_0 = 1.;
   T_max = 1.2; //OT
```

```
    calc_temperature(u);

    /*
       for (i = n_ghost; i < n_x -n_ghost+1; i++)
      {
          for (j = n_ghost; j < n_y -n_ghost+1; j++)

          {
             //u[f_col_][i][j] = 0.;
             u[f_col_][i][j] = a_col_0 * sqrt((u[tem_2_][i][j] + u[tem_1_][i][j])/ (2. * T0));
                   //std::cout << u[f_col_][i][j] << std::endl;
             //u[a_ion_][i][j] = 0.;
             //u[a_rec_][i][j] = 0.;

             //add differently calculated rates

              //Hillier
             //u[a_ion_][i][j] = 0.5 * (tanh(u[tem_2_][i][j] - 0.3) / 2. + 0.5);
             //u[a_rec_][i][j] = 0.5 * (1 - u[a_ion_][i][j]);

             //Tmax from OT plots:
             u[a_ion_][i][j] = 0.5 * (tanh(u[tem_2_][i][j] - T_max/2.) / 2. + 0.5);
            // u[a_rec_][i][j] = 0.2 * (tanh((1./u[tem_1_][i][j]) - T_max/2.) / 2. + 0.5);
                   u[a_rec_][i][j] = 0.5 * (1 - u[a_ion_][i][j]);
          }
      }
    */


}
```

SAVE_BIN.CPP

```
#include<iostream>
#include<cmath>

#include"constants.h"
#include"mpi.h"

#include <string.h>
#include <cstring>
#include <fstream>
#include <sstream>

void save_binary(int i_t, double*** u0, double** x, double** y )
{

    MPI_File outfile;
    MPI_Status status;
    MPI_Offset offset;
    MPI_Datatype bufd;
    MPI_Offset filesize;


    int len, pos;

    int i_cpu;

    double *obuffer;

    int i,j,k;
    int bufsize;

    int glodim[2];
    int locdim[2];
    int start[2];




if ((i_t % i_t_save == 0) || (i_t == 100000))
  {

          str1 = std::to_string(i_t);

          len = str1.length();

          num = std::string(9, '0');

          pos = 9. - len;

          timestep= num.replace(pos, len, str1);


           filename = "./DATA/data_OT"+timestep;


       //create pointer to the char that allocates memory and
   char * cfilename = new char [filename.length()+1]; //+1 because of the null termination
```

```
        //char * strcpy ( char * destination, const char * source );
   std::strcpy(cfilename, filename.c_str());
        //cfilename now contains a c-string copy of filename

   MPI_Comm_rank(MPI_COMM_WORLD, &i_cpu);

   glodim[0] = n_x_global;
   glodim[1] = n_y_global;
   locdim[0] = n_x_global/n_cpu_x;
   locdim[1] = n_y_global/n_cpu_y;
   start[0]  = locdim[0] * (i_cpu % n_cpu_x);
   start[1]  = locdim[1] * (i_cpu / n_cpu_x);

   bufsize = (n_x - 2 * n_ghost) * (n_y - 2 * n_ghost);
   obuffer = new double[bufsize];

   MPI_Type_create_subarray(n_dim, glodim, locdim, start, MPI_ORDER_FORTRAN,
MPI_DOUBLE, &bufd);
   MPI_Type_commit(&bufd);
   MPI_File_open(MPI_COMM_WORLD, cfilename, MPI_MODE_CREATE | MPI_MODE_WRONLY,
MPI_INFO_NULL, &outfile);
  // MPI_File_open(MPI_COMM_WORLD, filename.c_str(), MPI_MODE_CREATE |
MPI_MODE_WRONLY, MPI_INFO_NULL, &outfile);



   for (j = n_ghost; j < n_x - n_ghost; j++)
      {
         for (k = n_ghost; k < n_y - n_ghost; k++) obuffer[(j - n_ghost) + (n_x - 2 * n_ghost) * (k -
n_ghost)] = x[j][k];
      }

   MPI_File_get_size(outfile, &filesize);
   MPI_File_set_view(outfile, filesize, MPI_DOUBLE, bufd, "native", MPI_INFO_NULL);
   MPI_Barrier(MPI_COMM_WORLD);
   MPI_File_write_all(outfile, obuffer, bufsize, MPI_DOUBLE, &status);
   MPI_Barrier(MPI_COMM_WORLD);



   for (j = n_ghost; j < n_x - n_ghost; j++)
      {
         for (k = n_ghost; k < n_y - n_ghost; k++) obuffer[(j - n_ghost) + (n_x - 2 * n_ghost) * (k -
n_ghost)] = y[j][k];
      }

   MPI_File_get_size(outfile, &filesize);
   MPI_File_set_view(outfile, filesize, MPI_DOUBLE, bufd, "native", MPI_INFO_NULL);
   MPI_Barrier(MPI_COMM_WORLD);
   MPI_File_write_all(outfile, obuffer, bufsize, MPI_DOUBLE, &status);
   MPI_Barrier(MPI_COMM_WORLD);



   for (i = 0; i < n_var; i++)
   {
   for (j = n_ghost; j < n_x - n_ghost; j++)
      {
```

```
          for (k = n_ghost; k < n_y - n_ghost; k++) obuffer[(j - n_ghost) + (n_x - 2 * n_ghost) * (k -
n_ghost)] = u0[i][j][k];
       }

   MPI_File_get_size(outfile, &filesize);
   MPI_File_set_view(outfile, filesize, MPI_DOUBLE, bufd, "native", MPI_INFO_NULL);
   MPI_Barrier(MPI_COMM_WORLD);
   MPI_File_write_all(outfile, obuffer, bufsize, MPI_DOUBLE, &status);
   MPI_Barrier(MPI_COMM_WORLD);

   }

   MPI_File_close(&outfile);
   MPI_Type_free(&bufd);

   delete [] obuffer;

   }


//return 0;
}
```

CALC_TEMPERATURE.CPP

```
#include<iostream>
#include<cmath>
#include"constants.h"


void calc_temperature(double ***u0)
{
 int i,j;

for (i = n_ghost; i < n_x-n_ghost+1; i++)
  {
  for (j = n_ghost; j < n_y-n_ghost+1; j++)
     {
     //u0[tem_1_][i][j] = (u0[pre_1_][i][j] * m_1) / (k_B * u0[rho_1_][i][j]); //temperature MHD / charged
fluid
     //u0[tem_2_][i][j] = (u0[pre_2_][i][j] * m_2) / (k_B * u0[rho_2_][i][j]); //temperature HD / neutral
fluid

     u0[tem_1_][i][j] = u0[pre_1_][i][j]/ (2. * Rg * u0[rho_1_][i][j]); //temperature MHD / charged fluid
     u0[tem_2_][i][j] = u0[pre_2_][i][j] / (Rg * u0[rho_2_][i][j]); //temperature HD / neutral fluid
     }
  }

}
```

PRIM_TO_CON.CPP

```
/*
 * obtain conservative variables (eg momentum, energy) from primitive variables
 * (eg velocity, pressure)
 * convert_primitive_to_conservative
 */

#include<iostream>
#include<cmath>
#include"constants.h"

   void prim_to_con(double*** u0)
   {
      int i, j;
      double B_tot;

      for (i =0; i < n_x; i++)
      {
         for (j = 0; j<n_y; j++)
         {
             B_tot = u0[b_x_][i][j] * u0[b_x_][i][j] + u0[b_y_][i][j] * u0[b_y_][i][j] + u0[b_z_][i][j] * u0[b_z_][i]
[j];
            //velocity to momentum
            u0[mom_x_1_][i][j] = u0[v_x_1_][i][j] * u0[rho_1_][i][j];
            u0[mom_y_1_][i][j] = u0[v_y_1_][i][j] * u0[rho_1_][i][j];
            u0[mom_z_1_][i][j] = u0[v_z_1_][i][j] * u0[rho_1_][i][j];

            //pressure to energy with equation of state
            u0[ene_1_][i][j] = u0[pre_1_][i][j] / (gam - 1.) + 0.5 * (u0[mom_x_1_][i][j] * u0[v_x_1_][i][j]
                                                   + u0[mom_y_1_][i][j] * u0[v_y_1_][i][j]
                                                   + u0[mom_z_1_][i][j] * u0[v_z_1_][i][j] + B_tot);



            //hydro

            u0[mom_x_2_][i][j] = u0[v_x_2_][i][j] * u0[rho_2_][i][j];
            u0[mom_y_2_][i][j] = u0[v_y_2_][i][j] * u0[rho_2_][i][j];
            u0[mom_z_2_][i][j] = u0[v_z_2_][i][j] * u0[rho_2_][i][j];
            u0[ene_2_][i][j] = u0[pre_2_][i][j] / (gam -1.) + 0.5 * (u0[mom_x_2_][i][j] * u0[v_x_2_][i][j]
                                                   + u0[mom_y_2_][i][j] * u0[v_y_2_][i][j]
                                                   + u0[mom_z_2_][i][j] * u0[v_z_2_][i][j]);
         }
      }


   }
```

CON_TO_PRIM.CPP

```
#include<iostream>
#include<cmath>
#include"constants.h"

void con_to_prim(double*** u0)
{
   int i,j;
   double B_tot;

      // get primitive variables (p, v) from conservative variables (e, mom)
   for (i=0; i< n_x;i++)
   {
      for (j=0; j< n_y; j++)
      {

      B_tot = u0[b_x_][i][j] * u0[b_x_][i][j] + u0[b_y_][i][j] * u0[b_y_][i][j] + u0[b_z_][i][j] * u0[b_z_][i][j];

         //MHD
      u0[v_x_1_][i][j] = u0[mom_x_1_][i][j] / u0[rho_1_][i][j];
      u0[v_y_1_][i][j] = u0[mom_y_1_][i][j] / u0[rho_1_][i][j];
      u0[v_z_1_][i][j] = u0[mom_z_1_][i][j] / u0[rho_1_][i][j];

//gas pressure
      u0[pre_1_][i][j] = (u0[ene_1_][i][j] - ((u0[v_x_1_][i][j] * u0[mom_x_1_][i][j]
                                    + u0[v_y_1_][i][j] * u0[mom_y_1_][i][j]
                                    + u0[v_z_1_][i][j] * u0[mom_z_1_][i][j])  + B_tot) * 0.5 ) * (gam - 1.);

      //hydro
      u0[v_x_2_][i][j] = u0[mom_x_2_][i][j] / u0[rho_2_][i][j];
      u0[v_y_2_][i][j] = u0[mom_y_2_][i][j] / u0[rho_2_][i][j];
      u0[v_z_2_][i][j] = u0[mom_z_2_][i][j] / u0[rho_2_][i][j];
      u0[pre_2_][i][j] = (u0[ene_2_][i][j] - 0.5 * (u0[v_x_2_][i][j] * u0[mom_x_2_][i][j]
                                       + u0[v_y_2_][i][j] * u0[mom_y_2_][i][j]
                                       + u0[v_z_2_][i][j] * u0[mom_z_2_][i][j])) * (gam - 1.);

      }
   }
```

TIMESTEP.CPP

```
/*
 * calculate timestep with CFL condition
 * obtain the maximum velocity by comparing both max velocities
 */

      //calculate timestep with CFL condition dt = 0.2 * dx / v_max

      //1. calculate speed of sound
      //2. calculate maximum velocity v_x and v_y
      //3. get the maximum of both
      //dt = 0.2 * dx / v_max
#include <iostream>
#include <cmath>
#include "constants.h"
#include "mpi.h"


double timestep(double***u0)
{
   int i, j;
   double dt, dt_global, d_xy, cs_max, alfven_max, vabs_max, vmax, B_tot, B_x, B_y, B_z;
   double v_x_1, v_y_1, v_z_1, pre_1, v_x_2, v_y_2, v_z_2, pre_2;
   double cs_max_1, cs_max_2, vabs_max_1, vabs_max_2;


   int ind_cpu;

   double v_max_global, cs_1_max_global, cs_2_max_global;


   cs_max_1 = 0.0;
   cs_max_2 = 0.0;
   alfven_max = 0.0;
   vabs_max_1 = 0.0;
   vabs_max_2 = 0.0;
   vmax = 0.0;



   for (i=n_ghost; i <n_x-n_ghost+1; i++)
   {
      for (j=n_ghost; j < n_y-n_ghost+1; j++)
      {

            v_x_1 = u0[mom_x_1_][i][j] / u0[rho_1_][i][j];
            v_y_1 = u0[mom_y_1_][i][j] / u0[rho_1_][i][j];
            v_z_1 = u0[mom_z_1_][i][j] / u0[rho_1_][i][j];


            B_x = u0[b_x_][i][j];
            B_y = u0[b_y_][i][j];
            B_z = u0[b_z_][i][j];

            //hydro
            v_x_2 = u0[mom_x_2_][i][j] / u0[rho_2_][i][j];
            v_y_2 = u0[mom_y_2_][i][j] / u0[rho_2_][i][j];
            v_z_2 = u0[mom_z_2_][i][j] / u0[rho_2_][i][j];
```

```
        B_tot = B_x * B_x + B_y * B_y + B_z * B_z;

        pre_1 = (u0[ene_1_][i][j] - 0.5 * (u0[mom_x_1_][i][j] * v_x_1
                                + u0[mom_y_1_][i][j] * v_y_1
                                + u0[mom_z_1_][i][j] * v_z_1) + B_tot ) * (gam - 1.); //total e - (ekin +
emag)


        pre_2 =  (u0[ene_2_][i][j] - 0.5 * (u0[mom_x_2_][i][j] * v_x_2
                                + u0[mom_y_2_][i][j] * v_y_2
                                + u0[mom_z_2_][i][j] * v_z_2)) * (gam - 1.) ;



        cs_max_1 = fmax(cs_max_1, sqrt(gam * pre_1 / u0[rho_1_][i][j])); //max sound speed MHD

        cs_max_2 = fmax(cs_max_2, sqrt(gam * pre_2 / u0[rho_2_][i][j])); //max sound speed
HYDRO

        vabs_max_1 = fmax(vabs_max_1,(sqrt(v_x_1 * v_x_1 +
                                v_y_1 * v_y_1 +
                                v_z_1 * v_z_1 ))); //maximum flow speed MHD

        vabs_max_2 = fmax(vabs_max_2,(sqrt(v_x_2 * v_x_2 +
                                v_y_2 * v_y_2 +
                                v_z_2 * v_z_2 ))); //maximum flow speed HYDRO

        alfven_max = fmax(alfven_max, B_tot / sqrt(2. * u0[rho_1_][i][j]));



      }
   }


cs_max_1 = sqrt(cs_max_1 * cs_max_1 + alfven_max * alfven_max);

cs_max = fmax(cs_max_1, cs_max_2);

vabs_max = fmax(vabs_max_1, vabs_max_2);

vmax = fmax(cs_max,vabs_max);


d_xy = fmin(dx,dy);
dt = CFL * d_xy / vmax; //for regular grid
// dt = 0.00001;

MPI_Allreduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MIN, MPI_COMM_WORLD);

MPI_Allreduce(&cs_max_1, &cs_1_max_global, 1, MPI_DOUBLE, MPI_MAX,
MPI_COMM_WORLD);
MPI_Allreduce(&cs_max_2, &cs_2_max_global, 1, MPI_DOUBLE, MPI_MAX,
MPI_COMM_WORLD);
MPI_Allreduce(&vabs_max, &v_max_global, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);

   MPI_Comm_rank(MPI_COMM_WORLD, &ind_cpu);
   //if (ind_cpu == 0) std::cout << "IN DT::: CS1=" << cs_1_max_global << " CS2=" <<
cs_2_max_global << " VMAX=" << v_max_global << " DT =" << dt_global << std::endl;
```

```
return dt_global;
}
```

ALLOCATE MEMORY

```
#include<iostream>
#include<cmath>
#include"constants.h"

void alloc_mem_arr2D(double** &arr2D)
{
   int i,j,k;

   arr2D = new double*[n_x];
   for (i = 0; i < n_x; i++)
   {
      arr2D[i] = new double[n_y];
   }


}

#include<iostream>
#include<cmath>
#include"constants.h"

void alloc_mem_arr3D(double*** &arr3D)
{
   int i, j, k;

   arr3D = new double**[n_var];
   for (i = 0; i < n_var; ++i)
   {
      arr3D[i] = new double*[n_x];
      for (j =0; j < n_x; j++)
         arr3D[i][j] = new double[n_y];
   }


}

void alloc_mem_arr4D(double**** &arr4D)
{
   int i,j,k;

   arr4D = new double***[n_dim];
   for (i = 0; i < n_dim; ++i)
   {
      arr4D[i] = new double**[n_var];
      for (j =0; j < n_var; j++)
         {arr4D[i][j] = new double*[n_x];
                  for (k =0; k < n_x; k++)
                           {
                           arr4D[i][j][k] = new double[n_y];}
                  }

   }

}
```

DEALLOCATE MEMORY

```
#include<iostream>
#include<cmath>
#include"constants.h"

void dealloc_mem_arr2D(double** arr2D)
{
   int i, j, k;

 for (i = 0; i < n_x; i++)
      {
           delete [] arr2D[i];
        }
        delete [] arr2D;


}



#include<iostream>
#include<cmath>
#include"constants.h"

void dealloc_mem_arr3D(double*** arr3D)
{
   int i, j, k;


        for (i = 0; i < n_var; ++i)
      {
         for (j =0; j < n_x; j++)
         {
            delete [] arr3D[i][j];
         }
         delete [] arr3D[i];
      }
        delete [] arr3D;


}



#include<iostream>
#include<cmath>
#include"constants.h"

void dealloc_mem_arr4D(double**** arr4D)
{
   int i, j, k;



   for (k = 0; k < n_dim; k++)
      {
        for (i = 0; i < n_var; ++i)
```

```
        {
            for (j =0; j < n_x; j++)
            {
                delete [] arr4D[k][i][j];
            }
            delete [] arr4D[k][i];
        }
          delete [] arr4D[k];
            }
        delete [] arr4D;



}
```

Bibliography

Alvarez Laguna, A., Ozak, N., Lani, A., Deconinck, H., Poedts, S.: 2018, Fully-implicit finite volume method for the ideal two-fluid plasma model. *Computer Physics Communications* **231**, 31 – 44. doi:10.1016/j.cpc.2018.05.006.

Alvarez-Laguna, A., Ozak, N., Lani, A., Mansour, N.N., Deconinck, H., Poedts, S.: 2018, A versatile numerical method for the multi-fluid plasma model in partially- and fully-ionized plasmas. *Journal of Physics: Conference Series* **1031**, 012015. doi:10.1088/1742-6596/1031/1/012015.

Arber, T.D., Haynes, M., Leake, J.E.: 2007, Emergence of a Flux Tube through a Partially Ionized Solar Atmosphere. **666**, 541 – 546. doi:10.1086/520046.

Arber, T.D., Longbottom, A.W., Gerrard, C.L., Milne, A.M.: 2001, A Staggered Grid, Lagrangian-Eulerian Remap Code for 3-D MHD Simulations. *Journal of Computational Physics* **171**(1), 151 – 181. doi:10.1006/jcph.2001.6780.

Arregui, I.: 2015, Wave heating of the solar atmosphere. *Philosophical Transactions of the Royal Society of London Series A* **373**, 20140261 – 20140261. doi:10.1098/rsta.2014.0261.

Atkinson, K.: 1989, *An introduction to numerical analysis*, Wiley, Iowa. ISBN 9780471624899.

Ballester, J.L., Alexeev, I., Collados, M., Downes, T., Pfaff, R.F., Gilbert, H., Khodachenko, M., Khomenko, E., Shaikhislamov, I.F., Soler, R., Vazquez-Semadeni, E., Zaqarashvili, T.: 2017, Partially Ionized Plasmas in Astrophysics. *ArXiv e-prints*.

Balsara, D.S.: 2004, Second-Order-accurate Schemes for Magnetohydrodynamics with Divergence-free Reconstruction. **151**, 149 – 184. doi:10.1086/381377.

Balsara, D., Kim, J.: 2003, An intercomparison between divergence-cleaning and staggered mesh formulations for numerical magnetohydrodynamics. *The Astrophysical Journal* **602**. doi:10.1086/381051.

Biberman, L.M., Vorob'ev, V.S., Yakubov, I.T.: 1969, On the Theory of Ionization and Recombination in a Low-temperature Plasma. *Soviet Journal of Experimental and Theoretical Physics* **29**, 1070.

Bittencourt, J.A.: 2004, *Fundamentals of plasma physics*, 3rd edn. Springer, New York.

Botha, G., Rucklidge, A., Hurlburt, N.: 2006, Converging and diverging convection around axisymmetric magnetic flux tubes. *Monthly Notices of the Royal Astronomical Society* **369**, 1611 – 1624. doi:10.1111/j.1365-2966.2006.10480.x.

Boyd, T., Sanderson, J.: 2003, *The physics of plasmas*, Cambridge University Press, Cambridge. ISBN 9780521459129.

Braginskii, S.I.: 1965, Transport Processes in a Plasma. *Reviews of Plasma Physics* **1**, 205.

Brandenburg, A., Dobler, W.: 2002, Hydromagnetic turbulence in computer simulations. *Computer Physics Communications* **147**(1-2), 471 – 475. doi:10.1016/S0010-4655(02)00334-X.

Brio, M., Wu, C.C.: 1988, An upwind differencing scheme for the equations of ideal magnetohydrodynamics. *Journal of Computational Physics* **75**, 400 – 422. doi:10.1016/0021-9991(88)90120-9.

Cap, F.: 1994, *Lehrbuch der plasmaphysik und magnetohydrodynamik*, Springer, Vienna. ISBN 9783709166222.

Cavalli, F., Naldi, G., Puppo, G., Semplice, M.: 2006, A comparison between relaxation and Kurganov-Tadmor schemes. *ArXiv Mathematics e-prints*.

Chandrasekhar, S.: 1961, *Hydrodynamic and hydromagnetic stability*, *Dover Books on Physics Series*, Dover Publications, New York. ISBN 9780486640716.

Chen, F.: 1984, *Introduction to plasma physics and controlled fusion*, *Introduction to Plasma Physics and Controlled Fusion*, Springer, Los Angeles. ISBN 9780306413322.

Danaila, I., Joly, P., Kaber, S., Postel, M.: 2007, *An introduction to scientific computing - twelve computational projects solved with matlab*, 2nd edn. Springer, France. ISBN 0-387-30889-X.

Davidson, P.: 2001, *An introduction to magnetohydrodynamics*, 2nd edn. Cambridge University Press, England. ISBN 0-521-79487-0.

Davis, S.F.: 1987, A simplified tvd finite difference scheme via artificial viscosity. *SIAM Journal on Scientific and Statistical Computing* **8**(1), 1 – 18.

de Pontieu, B., Haerendel, G.: 1998, Weakly damped Alfven waves as drivers for spicules. *Astronomy and Astrophysics* **338**, 729 – 736.

Dedner, A., Kemm, F., Kröner, D., Munz, C.D., Schnitzer, T., Wesenberg, M.: 2002, Hyperbolic Divergence Cleaning for the MHD Equations. *Journal of Computational Physics* **175**, 645 – 673. doi:10.1006/jcph.2001.6961.

Denner, F., Evrard, F., Serfaty, R., [van Wachem], B.G.: 2017, Artificial viscosity model to mitigate numerical artefacts at fluid interfaces with surface tension. *Computers Fluids* **143**, 59 – 72. doi:https://doi.org/10.1016/j.compfluid.2016.11.006.

Díaz, A. J., Khomenko, E., Collados, M.: 2014, Rayleigh-taylor instability in partially ionized compressible plasmas: One fluid approach. *A&A* **564**, A97. doi:10.1051/0004-6361/201322147. https://doi.org/10.1051/0004-6361/201322147.

Draine, B.T., Roberge, W.G., Dalgarno, A.: 1983, Magnetohydrodynamic shock waves in molecular clouds. **264**, 485 – 507. doi:10.1086/160617.

Felipe, T., Khomenko, E., Collados, M.: 2010, Magneto-acoustic Waves in Sunspots: First Results From a New Three-dimensional Nonlinear Magnetohydrodynamic Code. **719**(1), 357 – 377. doi:10.1088/0004-637X/719/1/357.

Goedbloed, J.P.H., Poedts, S.: 2004, *Principles of Magnetohydrodynamics: With Applications to Laboratory and Astrophysical Plasmas*, Cambridge University Press, Cambridge. doi:10.1017/CBO9780511616945.

González-Morales, P.A., Khomenko, E., Downes, T. P., de Vicente, A.: 2018, Mhdsts: a new explicit numerical scheme for simulations of partially ionised solar plasma. *A&A* **615**, A67. doi:10.1051/0004-6361/201731916. https://doi.org/10.1051/0004-6361/201731916.

Grant, I., Phillips, W.R.: 1999, *Electromagnetism*, 2nd edn. Wiley, England. ISBN 978-0-471-92712-9.

Griffiths, D.J.: 1999, *Introduction to electrodynamics*, 3rd edn. Prentice Hall, USA. ISBN 0-13-805326-X.

Gudiksen, B.V., Carlsson, M., Hansteen, V.H., Hayek, W., Leenaarts, J., Martínez-Sykora, J.: 2011, The stellar atmosphere simulation codebifrost. *Astronomy Astrophysics* **531**, A154. doi:10.1051/0004-6361/201116520. http://dx.doi.org/10.1051/0004-6361/201116520.

Guillet, T., Pakmor, R., Springel, V., Chandrashekar, P., Klingenberg, C.: 2019, High-order magnetohydrodynamics for astrophysics with an adaptive mesh refinement discontinuous Galerkin scheme. **485**(3), 4209 – 4246. doi:10.1093/mnras/stz314.

Harten, A.: 1982, High Resolution Schemes for Hyperbolic Conservation Laws. *Journal of Computational Physics* **49**, 357 – 393.

Hartley, P., Wynn-Evans, A.: 1979, *An introduction to numerical analysis*, Stanley Thornes Ltd., England. ISBN 0859504263.

Hesthaven, J.S.: 2018, *Numerical methods for conservation laws*, 1st edn. Siam, Philadelphia, USA. ISBN 978-1-611975-09-3.

Hillier, A.: 2019, Ion-neutral decoupling in the nonlinear kelvin–helmholtz instability: Case of field-aligned flow. *Physics of Plasmas* **26**, 082902. doi:10.1063/1.5103248.

Hillier, A., Takasao, S., Nakamura, N.: 2016, The formation and evolution of reconnection-driven, slow-mode shocks in a partially ionised plasma. *Astronomy and Astrophysics* **591**, A112. doi:10.1051/0004-6361/201628215.

Hillier, A.S.: 2016, On the nature of the magnetic rayleigh–taylor instability in astrophysical plasma: the case of uniform magnetic field strength. *Monthly Notices*

of the Royal Astronomical Society **462**(2), 2256 – 2265. doi:10.1093/mnras/stw1805. http://dx.doi.org/10.1093/mnras/stw1805.

Jiang, G.S., Levy, D., Lin, C., Osher, S., Tadmor, E.: 1997, High-resolution non-oscillatory central schemes with non-staggered grids for hyperbolic conservation laws. *SIAM J. Numer. Anal* **35**, 2147 – 2168.

Keppens, R.: 2007, Radiative transfer and numerical mhd. *Summer School*.

Khodachenko, M.L., Arber, T.D., Rucker, H.O., Hanslmeier, A.: 2004, Collisional and viscous damping of MHD waves in partially ionized plasmas of the solar atmosphere. **422**, 1073 – 1084. doi:10.1051/0004-6361:20034207.

Khomenko, E., Collados, M.: 2012, Heating of the Magnetized Solar Chromosphere by Partial Ionization Effects. *The Astrophysical Journal* **747**, 87. doi:10.1088/0004-637X/747/2/87.

Khomenko, E., Collados, M., Díaz, A., Vitas, N.: 2014, Fluid description of multi-component solar partially ionized plasma. *Physics of Plasmas* **21**(9), 092901. doi:10.1063/1.4894106. http://dx.doi.org/10.1063/1.4894106.

Kull, H.: 1991, Theory of the rayleigh-taylor instability. *Physics Reports* **206**(5), 197 – 325. doi:https://doi.org/10.1016/0370-1573(91)90153-D. http://www.sciencedirect.com/science/article/pii/037015739190153D.

Kumar, N., Roberts, B.: 2003, Ion–neutral collisions effect on mhd surface waves. *Solar Physics* **214**(2), 241 – 266. doi:10.1023/A:1024299029918. https://doi.org/10.1023/A:1024299029918.

Kurganov, A., Tadmor, E.: 2000, New high-resolution central schemes for nonlinear conservation laws and convection—diffusion equations. *J. Comput. Phys.* **160**(1), 241 – 282. doi:10.1006/jcph.2000.6459. http://dx.doi.org/10.1006/jcph.2000.6459.

Kuzmin, D.: 2010, *A guide to numerical methods for transport equations*, 1st edn., Nuernberg, Germany.

Leake, J.E., Arber, T.D.: 2006, The emergence of magnetic flux through a partially ionised solar atmosphere. *A&A* **450**(2), 805 – 818. doi:10.1051/0004-6361:20054099.

Leake, J.E., Lukin, V.S., Linton, M.G., Meier, E.T.: 2012, Multi-fluid Simulations of Chromospheric Magnetic Reconnection in a Weakly Ionized Reacting Plasma. **760**(2), 109. doi:10.1088/0004-637X/760/2/109.

LeVeque, R.J.: 2002, *Finite-volume methods for hyperbolic problems*, Cambridge University Press, Cambridge.

Londrillo, P., Zanna, L.D.: 2000, High-order upwind schemes for multidimensional magnetohydrodynamics. *The Astrophysical Journal* **530**(1), 508. http://stacks.iop.org/0004-637X/530/i=1/a=508.

Lora-Clavijo, F.D., Cruz-Perez, J.P., Guzman, F.S., Gonzalez, J.A.: 2013, Exact solution of the 1D Riemann problem in Newtonian and relativistic hydrodynamics. *ArXiv e-prints*.

Maneva, Y.G., Laguna, A.A., Lani, A., Poedts, S.: 2017, Multi-fluid modeling of magnetosonic wave propagation in the solar chromosphere: Effects of impact ionization and radiative recombination. *The Astrophysical Journal* **836**(2), 197. doi:10.3847/1538-4357/aa5b83. http://dx.doi.org/10.3847/1538-4357/aa5b83.

Martínez-Gómez, D., Soler, R., Terradas, J.: 2017, Multi-fluid Approach to High-frequency Waves in Plasmas. II. Small-amplitude Regime in Partially Ionized Media. **837**(1), 80. doi:10.3847/1538-4357/aa5eab.

Martínez Sykora, J., De Pontieu, B., Hansteen, V., Carlsson, M.: 2015, The role of partial ionisation effects in the chromosphere. *Philosophical Transactions of the Royal Society A* **373**. doi:https://doi.org/10.1098/rsta.2014.0268.

Martínez-Sykora, J., De Pontieu, B., Hansteen, V.H., Rouppe van der Voort, L., Carlsson, M., Pereira, T.M.D.: 2017, On the generation of solar spicules and alfvénic waves. *Science* **356**(6344), 1269 – 1272. doi:10.1126/science.aah5412. https://science.sciencemag.org/content/356/6344/1269.

Mathers, C.D., Cramer, N.F.: 1978, The effect of ionization and recombination on the resistivity of a partially ionized plasma in a magnetic field. *Australian Journal of Physics* **31**, 171. doi:10.1071/PH780171.

Mattis, D.C.: 1965, *The theory of magnetism. an introduction to the study of cooperative phenomena.* **149**, American Association for the Advancement of Science, Washington, DC, 411 – 412. doi:10.1126/science.149.3682.411-a.

McNally, C.P., Lyra, W., Passy, J.C.: 2012, A Well-posed Kelvin-Helmholtz Instability Test and Comparison. **201**, 18. doi:10.1088/0067-0049/201/2/18.

Meier, E.T., Shumlak, U.: 2012, A general nonlinear fluid model for reacting plasma-neutral mixtures. *Physics of Plasmas* **19**(7), 072508. doi:10.1063/1.4736975.

Norman, M.L., Winkler, K.: 1985, Supersonic Jets. *Los Almos Science*.

Orszag, S.A., Tang, C.M.: 1979, Small-scale structure of two-dimensional magnetohydrodynamic turbulence. *Journal of Fluid Mechanics* **90**, 129 – 143. doi:10.1017/S002211207900210X.

Pandey, B.P., Wardle, M.: 2008, Hall magnetohydrodynamics of partially ionized plasmas. *Monthly Notices of the Royal Astronomical Society* **385**(4), 2269 – 2278. doi:10.1111/j.1365-2966.2008.12998.x. https://doi.org/10.1111/j.1365-2966.2008.12998.x.

Peiro, J., Sherwin, S.: 2005, *Finite Difference, Finite Element and Finite Volume Methods for Partial Differential Equations*, 2415 – 2446. doi:10.1007/978-1-4020-3286-8$_1$27.

Popescu Braileanu, B., Lukin, V., Khomenko, E., De Vicente, A.: 2019, Two-fluid simulations of waves in the solar chromosphere. ii. propagation and damping of fast magneto-acoustic waves and shock. *Astronomy Astrophysics*. doi:10.1051/0004-6361/201935844.

Powell, K.G.: 1994, An approximate riemann solver for magnetohydrodynamics.

Priest, E.: 2014, *Magnetohydrodynamics of the Sun*, Cambridge University Press, Cambridge, UK.

Quarteroni, A., Sacco, R., Saleri, F.: 2006, *Numerical mathematics (texts in applied mathematics)*, Springer, Berlin, Heidelberg. ISBN 3540346589.

Raboonik, A., Cally, P.S.: 2019, Hall-coupling of Slow and Alfvén Waves at Low Frequencies in the Lower Solar Atmosphere. **294**(10), 147. doi:10.1007/s11207-019-1544-1.

Roe, P.: 1986, Characteristic-based schemes for the euler equations. *Annual Review of Fluid Mechanics* **18**, 337 – 365.

Rozhansky, V., Tsendin, L.: 2001, *Transport phenomena in partially ionized plasma*, Taylor & Francis, St Petersburg.

Ryu, D., Miniati, F., Jones, T.W., Frank, A.: 1998, A Divergence-free Upwind Code for Multidimensional Magnetohydrodynamic Flows. **509**, 244 – 255. doi:10.1086/306481.

Scheid, F.: 1968, *Schaum's outline of theory and problems of numerical analysis*, *Schaum's outline series*, McGraw-Hill, New York.

Schnack, D.D.: 2009, *Lectures in Magnetohydrodynamics: With an Appendix on Extended MHD*, *Lecture Notes in Physics*, Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-00688-3.

Shyy, W.: 2006, *Computational modeling for fluid flow and interfacial transport (dover books on engineering)*, Dover Publications, Incorporated, New York. ISBN 0486453030.

Snow, B., Hillier, A.: 2019, Intermediate shock sub-structures within a slow-mode shock occurring in partially ionised plasma. **626**, A46. doi:10.1051/0004-6361/201935326.

Sod, G.A.: 1978, A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws. *Journal of Computational Physics* **27**(1), 1 – 31. doi:10.1016/0021-9991(78)90023-2.

Soler, R., Oliver, R., Ballester, J.L.: 2009, Magnetohydrodynamic waves in a partially ionized filament thread. *The Astrophysical Journal* **699**(2), 1553 – 1562. doi:10.1088/0004-637x/699/2/1553.

Soler, R., Carbonell, M., Ballester, J.L., Terradas, J.: 2013, Alfvén waves in a partially ionized two-fluid plasma. *The Astrophysical Journal* **767**(2), 171. doi:10.1088/0004-637x/767/2/171. http://dx.doi.org/10.1088/0004-637X/767/2/171.

Soler, R., Terradas, J., Oliver, R., Ballester, J.L.: 2017, Propagation of torsional alfvén waves from the photosphere to the corona: Reflection, transmission, and heating in expanding flux tubes. *The Astrophysical Journal* **840**(1), 20. doi:10.3847/1538-4357/aa6d7f.

Spruit, H.C.: 2017, *Essential magnetohydrodynamics for astrophysics*, Garching.

Stone, J.M., Hawley, J.F., Evans, C.R., Norman, M.L.: 1992, A test suite for magnetohydrodynamical simulations. **388**, 415 – 437. doi:10.1086/171164.

Sweby, P.K.: 1984, High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws. *SIAM Journal on Numerical Analysis* **21**, 995 – 1011. doi:10.1137/0721062.

Tipler, P.A., Mosca, G.: 2009, *Physik für Wissenschaftler und Ingenieure*, 2nd edn. Spektrum Akademischer Verlag, Heidelberg. ISBN 978-3-8274-1945-3.

Toth, G.: 1994, Numerical Study of Two-Fluid C-Type Shock Waves. **425**, 171. doi:10.1086/173973.

Tsap, Y.T., Stepanov, A.V., Kopylova, Y.G.: 2011, Energy Flux of Alfvén Waves in Weakly Ionized Plasma and Coronal Heating of the Sun. **270**(1), 205 – 211. doi:10.1007/s11207-011-9727-4.

Versteeg, H., Malalasekera, W.: 2007, *An introduction to computational fluid dynamics: The finite volume method*, Pearson Education Limited, Edinburgh. ISBN 978-0-13-127498-3.

Vögler, A., Shelyag, S., Schüssler, M., Cattaneo, F., Emonet, T., Linde, T.: 2005, Simulations of magneto-convection in the solar photosphere. Equations, methods, and results of the MURaM code. *Astronomy and Astrophysics* **429**, 335 – 351. doi:10.1051/0004-6361:20041507.

Vranjes, J., Poedts, S., Pandey, B.P., de Pontieu, B.: 2008, Energy flux of Alfvén waves in weakly ionized plasma. *Astronomy and Astrophysics* **478**, 553 – 558. doi:10.1051/0004-6361:20078274.

Waterson, N., Deconinck, H.: 2007, Design principles for bounded higher-order convection schemes a unified approach. *Journal of Computational Physics* **224**, 182 – 207.

Wendt, J., Anderson Jr., J.D., Degroote, J., Degrez, G., Dick, E., R., G., J., V.: 2009, *Computational fluid dynamics*, 3rd edn. Springer, Berlin Heidelberg. ISBN 978-3-540-85055-7.

Wilmot-Smith, A.L., Priest, E.R., Hornig, G.: 2005, Magnetic diffusion and the motion of field lines. *Geophysical and Astrophysical Fluid Dynamics* **99**, 177 – 197. doi:10.1080/03091920500044808.

Yunus A. Cengel, Y.C.: 2002, *Heat transfer: a practical approach*, 2nd edn. McGraw-Hill Science/Engineering/Math, New York. ISBN 0072826207 9780072826203.

Zaqarashvili, T.V., Khodachenko, M.L., Rucker, H.O.: 2011, Magnetohydrodynamic waves in solar partially ionized plasmas: two-fluid approach. **529**, A82. doi:10.1051/0004-6361/201016326.

Zweibel, E.G.: 2015, In: Lazarian, A., de Gouveia Dal Pino, E.M., Melioli, C. (eds.) *Ambipolar Diffusion*, *Astrophysics and Space Science Library* **407**, 285. doi:10.1007/978-3-662-44625-6$_1$1.