Tbilisi State University

Faculty of Exact and Natural Sciences

Department of Physics

George Miloshevich

3D modeling of Solar Coronal Closed structure formation and heating due to magneto-fluid coupling

Fundamental Physics

Master's Program Fundamental Physics

Degree: Master of Physics (Astrophysics)

Supervisor: Nana Shatashvili, Professor

Tbilisi, Georgia, 2012

Abstract

Contents

Introduction

Corona Heating Paradox

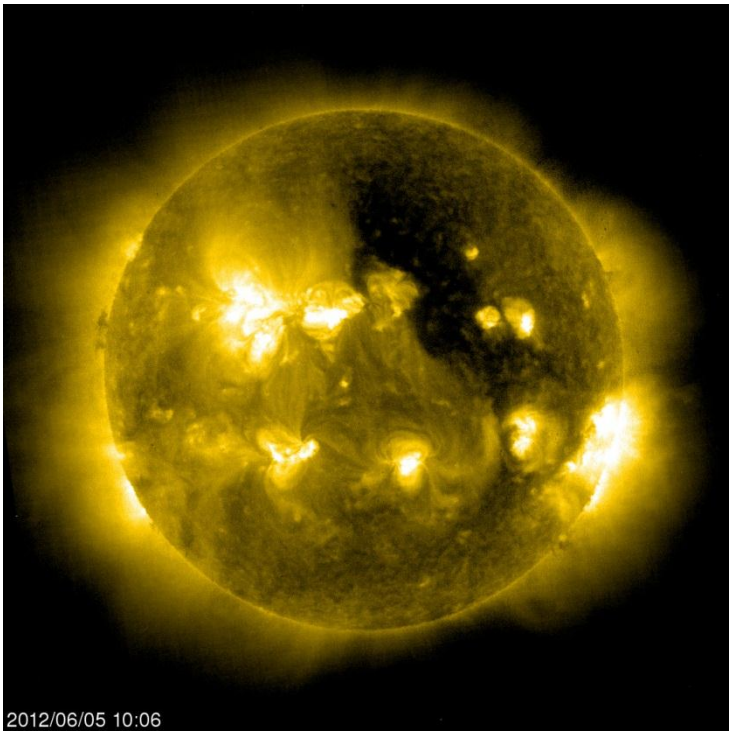

Figure 1: Full-disk image of the Sun at 284 Å wavelength from the Extreme ultraviolet Imaging Telescope (EIT) on the SOHO spacecraft

The Sun is the main sequence star with G2V spectral classification. The core of the Sun is believed to comprise a quarter of the star. It has density of 150 g/cm and a temperature approximately $15.7 \cdot 10^7\ K$. At this temperature the kinetic energy of protons is enough to overcome Coulomb potential and undergo fusion reactions. These reactions are responsible for electromagnetic radiation that creates pressure sufficient to oppose gravitational collapse of the Sun. The radiation eventually escapes from the star and plays the crucial role for biological life of the Earth. Although the core is hidden for direct observations, the evidence for nuclear reactions inside the Sun is provided by testing predictions of the theory regarding the density and temperature distributions with various methods like Solar Seismology or neutrino fluxes. The indirect measurements confirm that temperature of plasma falls gradually from tens of millions of degrees to 6000 K at the photosphere.

The photosphere is the layer of the sun below which Sun becomes opaque to visible light. The change in opacity is due to decreasing amount of H^- ions, which absorb visible light easily. Above the photosphere lies chromosphere, the transition region, that will be described more thoroughly below, and the Corona.

The solar corona is the outmost region of the Sun (Figure 1) that can be observed by naked eye during the eclipse. And indeed, the first observations of the solar corona date back to ancient eclipse observations, which have been reported from Indian, Babylonian, or Chinese sources. The element helium was discovered in the solar corona by Jules Janssen in 1868 by spectroscopic

observations. In 1942, Edlén identified forbidden lines[1] of highly ionized atoms and in this way established for the first time the million-degree temperature of the corona. The unusual spectral features of corona induced by high temperatures led some to suggest, in the 19th century, that it contained a previously unknown element, "coronium".

More contemporary observations in X-rays confirm that plasma temperature in corona is in excess of $10^6\ K$ and may sometimes locally approach $10^7 K$. This seems puzzling, since according to the second law of thermodynamics the heat flows from the hot regions to cold regions, whilst in the sun the temperature increases 200 times from photosphere to corona, in the narrow region of some 5000 kilometres called the transition region. However the density in the external part of the sun is very low (of the order of $10^{15}\ m^{-3}$), so it produces about one-millionth as much visible light, which is the reason why corona can only be observed during eclipse from the Earth. Moreover the amount of power required to heat the solar corona is assessed at about $1\ kW/m^2$ or $1/40000$ amount of light energy that escapes the Sun.

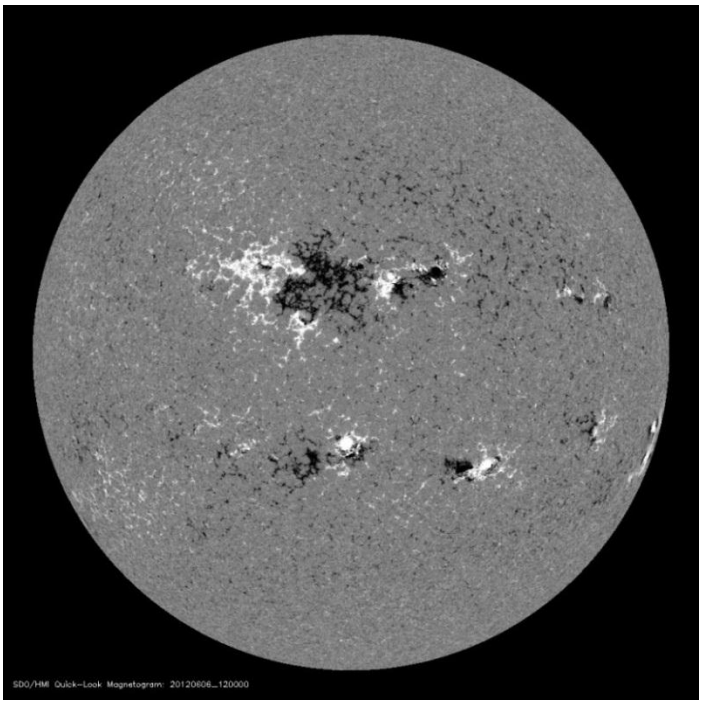

Figure 2: SDO HM1 Magnetogram Image from SOHO

Nevertheless, it is still complicated to construct the theory that explains how temperature increases with distance to the source of energy.

Dark and light regions on the Magnetogram (Figure 2) represent different polarity of the magnetic field lines extending from the Sun. When comparing it to (Figure 1) one is overwhelmed to identify the regions with strong magnetic fields to the regions where X-ray radiation is the strongest (light areas on Figure 1 where temperature is the greatest). This suggests that the magnetic field is the key player, which has been known for long. There are various candidates of anomalous heating of corona like MHD Waves, Reconnection, Turbulence or Flows; however the exact mechanism is unknown. The question of why the solar corona is so hot remains one of most exciting astronomy puzzles for the last 60 years.

[1] Forbidden lines – in astronomical spectroscopy, bright emission lines in the spectra of certain nebulae, not observed in the laboratory spectra of the same gases, because on Earth the gases cannot be rarefied sufficiently. The term forbidden is misleading; a more accurate description would be "highly improbable." (Encyclopaedia Britannica)

The regions described above (located in areas of strong magnetic field concentrations or visible as sunspot groups in optical wavelengths or magnetograms) are called Active regions. This places harbour violent processes such as flares and coronal mass ejections. The remaining areas outside of active regions were dubbed quiet (quiescent) Sun regions. However many dynamic process have been discovered all over the solar surface, like microflares, jets, transequatorial loops, explosive events, etc…

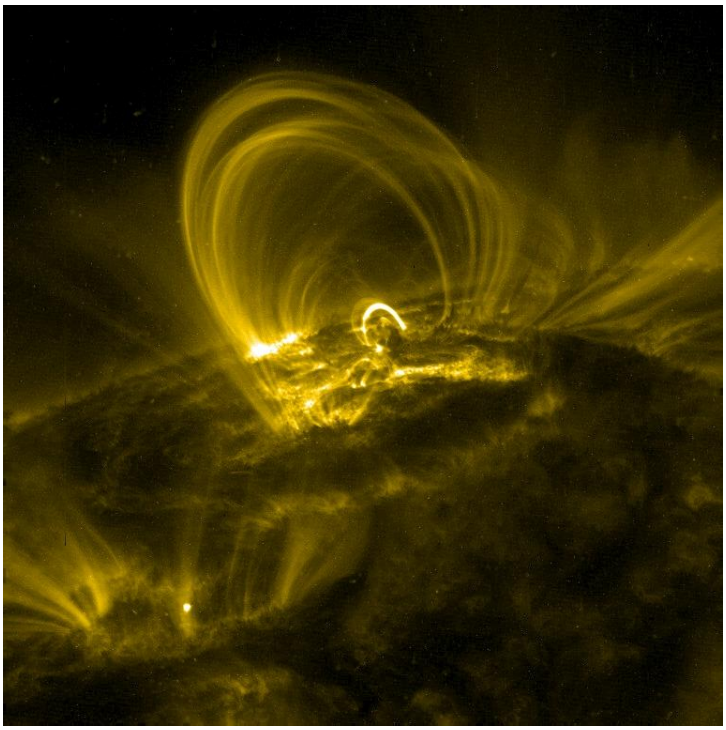

Figure 3: Typical Coronal Loops observed by TRACE

The structures that play the key role in this thesis are the coronal loops. They form the basic structure of the lower corona and transition region of the Sun. They populate both active and quite regions on the solar surface extending from the photosphere as a direct consequence of the twisted solar magnetic flux. In order to address closed fieldlines as coronal loops, they have to be filled with plasma. The evidence supporting existence of flows is overwhelming [1], [2], [3] and as recent as [4]. There are various mechanisms that attempt to explain the origin of the flows that fill the coronal loops with chromospheric plasma, like reconnection for instance, or the model where primary plasma flow (locally sub-Alfvénic) is accelerated while interacting with ambient arcade – like closed field structures [5]. The idea that the coronal heating problem is solely down to some coronal heating mechanism is misleading, since the observations of coronal upflows suggest a chromospheric source of plasma.

Popular Mechanisms

Out of popular mechanisms that are deemed responsible for anomalous heating it is notable to mention heating by waves and reconnection.

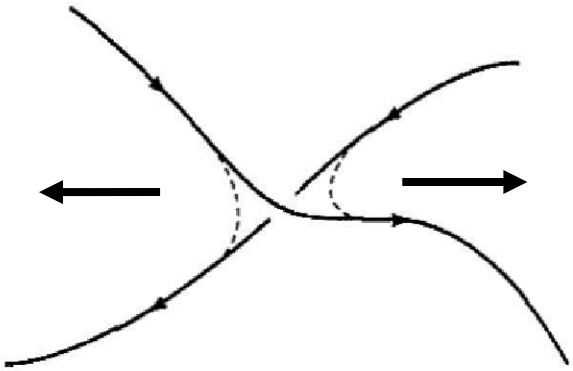

Figure 4: Magnetic Reconnection taking place for two field lines. The vertical arrows show the outflow of plasma

Magnetic reconnection is a phenomenon whereby oppositely directed magnetic field lines, carried along with a plasma "fluid" until they come close, are cut and reconnected via effect of finite resistivity into a different way. This process converts magnetic energy into kinetic energy, thermal energy and particle acceleration via inducing large electric currents. This is why it is believed to be accounted for plethora of spectacular events in space and laboratory plasmas. For instance, magnetic reconnection is hypothesized to be the mechanism behind solar flares, the largest explosions in our solar system. Using values of coronal electron temperature $T_e \sim 10^6\, K$ and magnetic diffusivity $\eta \sim 10^4 cm^2/s$ for the typical diameter of a solar flare $L \sim 10^4\, km$ one obtains that typical time-scale of the magnetic diffusion would be $\tau_\eta = L^2/\eta \sim 10^{14} s$, whereas the actual, observed time is less than $\sim 10^3 s$. In order to explain these discrepancies and somehow enhance dissipative effects, plasma theorists decided to introduce large gradients, the concept of current sheets. As the plasma "fluid" is carried along the frozen-in field it may generate steep gradients (that arise during magnetic reconnection) so that previously neglected small resistive effects are locally increased, this leads to more effective release of thermal energy. After the rearrangement of magnetic connectivity, the global magnetic configuration may relax by reducing magnetic tension and accelerating the fluid in the way indicated in Figure 4. However, the application of this mechanism to explain coronal heating problem has certain difficulties. Magnetic reconnection explains well catastrophic events like solar flares, or coronal mass ejections but falls short of providing constant and uniform mechanism occurring everywhere in corona. Hence, the idea of nanoflares was introduced by Parker [6] but according to TRACE[2] and SOHO[3] observations [7] there seem to be too few of these events to account for the energy released into the corona.

The theory that competes with reconnection is the wave heating theory, proposed by Evry Schatzman. According to the wave heating theory constant granulation caused by convection under the photosphere generates MHD waves that can carry energy from the solar interior for

[2] TRACE – Transition Region and Coronal Explorer, U.S. satellite designed to study the solar corona. (Encyclopaedia Britannica)

[3] SOHO - Solar and Heliospheric Observatory, satellite managed jointly by ESA and NASA that is equipped with a battery of novel instruments to study the Sun.

some distance through the solar atmosphere before turning into shock waves that dissipate their energy as heat. However, this approach also has some problems. Namely, magneto-acoustic waves cannot carry sufficient energy upward through the chromosphere because they tend to be reflected back to the photosphere. Alfven waves can carry enough energy, but do not dissipate that energy rapidly enough once they enter the corona. Through most of the past 50 years, neither theory has been able to account fully for the extreme coronal temperatures.

Involving flow in Heating Mechanisms

Different approach was developed in [8]. In this paper a class of mechanisms have been examined, which, through the viscous dissipation of the plasma kinetic energy, provide the primary and basic heating of the coronal structures during their very formation. Indeed observation confirm [3] that lots of thin loops that the corona is comprised of, are heated for a few tens of minutes, after which the heating ceases, or at least changes significantly in magnitude (citation). The crucial aspect of a theory is the assumption grounded in observations [2] that relatively cold chromospheric upflows exist. The flows, interacting with the ambient magnetic field anchored inside the surface of the Sun, are responsible for evolving and reorganizing coronal structures. During the process of trapping and accumulation, a part of the kinetic energy of the flow is converted to heat by viscous dissipation and the coronal structure is born hot and bright. However, fast and more efficient dissipation mechanisms are required for this.

One of the possible mechanisms by which the flow energy may be converted into heat energy is the ability of supersonic flows to create nonlinear perturbations and steepen to produce short scale structures (shocks) that can dissipate by ordinary viscosity. Supersonic flow thermalization is based on the existence of intrinsic short scales due to the shocks (primary heating) as well as two-fluid effects (mostly secondary heating) that may provide the steadier source for heating. In this model the heating occurs in two stages – primary heating and the secondary heating. Primary heating refers to a hectic dynamic period when a closed coronal structure acquires particles and energy. Then, in order to sustain the structure (against say radiative losses) secondary heating mechanisms are required.

The other possibility, which can be the source of the secondary heating, stems from property of the magnetofluid equilibria for the sub-Alfvénic flows – such flows can have a substantial, quick spatially varying velocity field component even when the magnetic field is mostly smooth. However this mechanism is not considered here because of the requirement of a much higher spatial resolution.

Such process would require the existence of particle flows at some height of solar atmosphere with reasonable amount of kinetic energy. Fortunately, such flows ($90 - 400\ km/s$) [2] have been experimentally observed by SOHO in the inner and middle corona. There were also positive observations that came from Ulysses and TRACE. In the model described here it is believed that initially cold low energy particle-flows continually emerge from sub-coronal to the coronal regions accelerated on the way (mechanisms could be several). Flow gets trapped in magnetic field structures of different strength (always present even in the quiescent Sun), thermalize and dissipate their energy.

It is noteworthy to mention [9] that the primary upflow originating from the chromosphere/transition region should be fast and cold (as opposed to hot). The heating occurs not because of the thermal energy transfer between the primary flow and corona but because supersonic flows produce shocks, as opposed to hot subsonic shocks. In fact, flows with speed $\sim 300\ km/s$ are heated to $10^6 K$ or more as they are slowed down by a passage through a shock front. Then the multi-scale picture of temperature is easy to explain – for different plasma flows, i.e. moving with different initial speed, different temperatures will be observed.

Modelling of Solar Corona

Theoretical Model

The model used below assumes that electron and proton flow velocities differ, that is, the two-fluid approximation. Outside the Debye shielding radius the quasi-neutrality holds:

$$n_e \approx n_i = n \tag{1}$$

The kinetic pressure is the sum of the partial pressures of both electrons and ions:

$$p = p_i + p_e \approx 2\, nT \tag{2}$$

In case of heavily asymmetric species by one parameter (like mass) where both constituents have different masses: $m_i \approx 1836\, m_e$, the roles of plasma species are split: ions determine inertial properties of plasma while field is frozen-in electron fluid (the so called, non-dissipative limit). Thus the fluid velocity is equal to the ion velocity in the equation of motion, while current depends on both the motion of ions and electrons, so:

$$\boldsymbol{V}_i = \boldsymbol{V}; \quad \boldsymbol{V}_e = \boldsymbol{V} - \frac{\boldsymbol{j}}{en} \tag{3}$$

From Faraday and Ohm's laws, equations of motion and (1) - (3) following two equations can be obtained by neglecting electron inertia:

$$\frac{\partial n}{\partial t} + \nabla \cdot n\boldsymbol{V} = \boldsymbol{0} \tag{4}$$

$$\frac{\partial}{\partial t} V_k + \boldsymbol{V} \cdot \nabla \ \ V_k = \frac{1}{m_i nc} (\boldsymbol{j} \times \boldsymbol{B})_k - \frac{2}{n m_i} \nabla_k (nT) + \nabla_k \left(\frac{GM_\odot}{r} \right) - \frac{1}{n m_i} \nabla_l \Pi_{i,kl} \tag{5}$$

$$\frac{\partial}{\partial t} \boldsymbol{B} - \nabla \times \left[\left(\boldsymbol{V} - \frac{\boldsymbol{j}}{en} \right) \times \boldsymbol{B} \right] = \frac{c}{e} \nabla \left(\frac{1}{n} \right) \times \nabla (nT) \tag{6}$$

$$\boldsymbol{j} = \frac{c}{4\pi} \nabla \times \boldsymbol{B} \tag{7}$$

$$\nabla \cdot \boldsymbol{B} = \boldsymbol{0} \tag{8}$$

Where m_i – the ion mass, G – the gravitational constant, $M_\odot$ - Solar mass, r – radial distance to the centre of the Sun and $\Pi_{i,kl}$ – ion viscosity tensor. Evolution of temperature is obtained from energy balance equation for a magnetized, neutral, isothermal electron – proton plasma,

$$\frac{\partial}{\partial t} \epsilon_\alpha + \nabla_k \left(\epsilon_\alpha V_{\alpha,k} + P_{\alpha,kl} V_{\alpha,l} \right) + \nabla \boldsymbol{q}_\alpha = n_\alpha \boldsymbol{f}_\alpha \cdot \boldsymbol{V}_\alpha \tag{9}$$

Here α is the index of species; the fluid energy ϵ_α and the total pressure tensor $P_{\alpha,kl}$ are given by

$$\epsilon_\alpha = n_\alpha \left(\frac{3}{2} T_\alpha + \frac{m_\alpha V_\alpha^2}{2} \right), \ \ P_{\alpha,kl} = n_\alpha T_\alpha \delta_{kl} + \Pi_{\alpha,kl} \tag{10}$$

and

$$\boldsymbol{f}_\alpha = e_\alpha E_\alpha + \frac{e_\alpha}{c} V_\alpha \times B + m_\alpha \nabla \frac{GM_\odot}{r} \tag{11}$$

is the force acting on the volume element. In equation (9), $\boldsymbol{q}_\alpha$ is the heat flux density. It can be shown that these equations together with the initial assumptions lead to the following temperature evolution equation:

$$\begin{aligned} \frac{3}{2} n \frac{d}{dt} (2T) + \nabla (q_e + q_i) & \\ = -2nT \ \ \nabla \cdot V + m_i n \, \nu_i \left[\frac{1}{2} \left(\frac{\partial V_k}{\partial x_l} + \frac{\partial V_l}{\partial x_k} \right)^2 - \frac{2}{3} (\boldsymbol{\nabla} \cdot \boldsymbol{V})^2 \right] & \\ + \frac{5}{2} n \left(\frac{\boldsymbol{j}}{en} \cdot \boldsymbol{\nabla} T \right) - \frac{\boldsymbol{j}}{en} \boldsymbol{\nabla} (nT) + E_H + E_R & \end{aligned} \tag{12}$$

where E_R is the total radiative loss, E_H - local mechanical heating function, and ν_i – ion kinematic viscosity. If primary flows are ignored various heating mechanisms like wave heating or reconnection should be invoked, and corresponding term E_H has to be added then. When

considering intrinsic mechanisms of creating short scales there is no need to invoke E_H term (local mechanical heating function).

Magnetofluid coupling – towards the unified model

The modern approach to the coronal heating often involves underlining importance of small-scale dynamics. The divergence of small-scales leads to large resistive and viscous dissipation even if resistivity and viscosity coefficients are as small as discussed before in the "Popular Mechanisms" section. Dissipation process acting on short scale could cause rapid changes that are otherwise inexplicable in terms of large scale mechanisms, like one-fluid MHD for instance. Small scales are not present in one-fluid MHD but can emerge once Hall term is taken into consideration [8], [10].

In [10] a particular solution of Hall MHD is examined that is called double Beltrami field. Within the framework of the model mechanisms for creation of short-scales are investigated: "The cooperation of nonlinearity (producing collapsed characteristics) and dispersion (unfolding singularities) underlies a robust mechanism that imparts two distinct scales (L measuring the system size, and δ_i typically of the order of the ion skin depth) to the double Beltrami states of a two-fluid plasma". In the paper the purely macroscopic (single-fluid MHD) model is referred to as the "0-model" and the more comprehensive microscopic model (Hall MHD) as the "ϵ-model". If a solution of ϵ-model converges to the solution of 0-model ($f_\epsilon \to f_0$) as $\lim \epsilon \to 0$ the perturbation introduced by ϵ-model has a negligible effect. While, if f_ϵ is singular as $\lim \epsilon \to 0$ the ϵ-model acquires much richer physical content than its macroscopic counterpart. The singular perturbation then is the harbinger of complexity.

For simplicity quasi-neutral plasma with singly charged ions is considered again as in the previous section. Neglecting the small inertia of electron, its equation of motion reads:

$$\boldsymbol{E} + \boldsymbol{V_e} \times \boldsymbol{B} + \frac{1}{en}\nabla p_e = 0 \tag{13}$$

The ion velocity (3) obeys:

$$\frac{\partial \boldsymbol{V}}{\partial t} + \boldsymbol{V} \cdot \nabla \ \ \boldsymbol{V} = \frac{e}{M}(\boldsymbol{E} + \boldsymbol{V} \times \boldsymbol{B}) - \frac{1}{Mn}\nabla p_i \tag{14}$$

Now, E and V_e can be eliminated using $\boldsymbol{V_e} = \boldsymbol{V} - \boldsymbol{j}/en$, $\boldsymbol{j} = \mu_0^{-1}\nabla \times \boldsymbol{B}$, and $\boldsymbol{E} = -\partial \boldsymbol{A}/\partial t \ - \nabla\phi$. The variables are normalized according to arbitrary length scale L_0 (typically the size of the system), representative magnetic field B_0 and density n_0:

$$\boldsymbol{x} = L_0\hat{\boldsymbol{x}}, \qquad B = B_0\hat{\boldsymbol{B}}, \qquad n = n_0\hat{n}, \quad t = (L_0/V_A)\hat{t}, \;\; p = (B_0^2/\mu_0)\hat{p}, \;\; \phi = (L_0B_0V_A)\hat{\phi}, \;\; V = V_A\hat{\boldsymbol{V}} \tag{15}$$

Here $V_A = B_0/\sqrt{\mu_0 M n_0}$ is the Alfven speed. Accordingly, equations (13) - (14) transform to

$$\frac{\partial \hat{\boldsymbol{A}}}{\partial \hat{t}} = (\hat{\boldsymbol{V}} - \frac{\epsilon}{\hat{n}}\hat{\nabla} \times \hat{\boldsymbol{B}}) \times \hat{\boldsymbol{B}} - \hat{\nabla}(\hat{\phi} - \epsilon\hat{\Pi}_e) \tag{16}$$

$$\frac{\partial}{\partial \hat{t}}\left(\epsilon\hat{\boldsymbol{V}} + \hat{\boldsymbol{A}}\right) = \hat{\boldsymbol{V}} \times (\hat{\boldsymbol{B}} - \epsilon\hat{\nabla} \times \hat{\boldsymbol{V}}) - \hat{\nabla}(\hat{\phi} + \epsilon\hat{\Pi}_i + \epsilon\hat{V}^2/2) \tag{17}$$

Here the scaling coefficient $\epsilon = \delta_i/L_0$ is the ratio of intrinsic scale, the ion skin depth to the macroscopic scale.

$$\delta_i = \frac{c}{\omega_{pi}} = \sqrt{\frac{M}{\mu_0 n e^2}} \tag{18}$$

In what follows the ^-notation is removed to simplify matters. Moreover considering incompressibility is sufficient for capturing effect. Thus $n = 1$ and $\Pi_j = p_j$ $(j = e, i)$. Subtracting (36) from (37) leads to

$$\frac{\partial \boldsymbol{V}}{\partial \hat{t}} + (\boldsymbol{V} \cdot \boldsymbol{\nabla})\mathbf{V} = (\hat{\nabla} \times \boldsymbol{B}) \times \boldsymbol{B} - \nabla p \tag{19}$$

While taking curl of (36) produces

$$\frac{\partial \boldsymbol{B}}{\partial \hat{t}} = \nabla \times (\boldsymbol{V} - \epsilon\nabla \times \boldsymbol{B}) \times \boldsymbol{B} \tag{20}$$

So the familiar Hall MHD equations (19) and (20) are obtained. Here $p = p_e + p_i$. In the limit where $\epsilon \to 0$ Hall MHD reproduce single-fluid MHD. The "dispersive" Hall term $\epsilon(\nabla \times \boldsymbol{B}) \times \boldsymbol{B}$ is the singular perturbation included in the ϵ-model. This is similar to how dissipative viscosity term (multiplied by the reciprocal Reynolds number) provides the singular perturbation in the well-known Navier-Stokes equation.

The equivalence of (39) and (40) to the vortex transport equations is easily revealed by taking curl of both equations leading to

$$\frac{\partial \boldsymbol{\Omega_j}}{\partial \hat{t}} = \nabla \times \left(\boldsymbol{U_j} \times \boldsymbol{\Omega_j}\right) \qquad (j = 1,2) \tag{21}$$

The pair of "vorticities" and their corresponding flows is defined as:

$$\boldsymbol{\Omega} = \begin{pmatrix} \boldsymbol{B} \\ \boldsymbol{B} + \epsilon\boldsymbol{\nabla} \times \boldsymbol{V} \end{pmatrix}; \qquad \boldsymbol{U} = \begin{pmatrix} \boldsymbol{V} - \epsilon\boldsymbol{\nabla} \times \boldsymbol{B} \\ \boldsymbol{V} \end{pmatrix} \tag{22}$$

The simplest stationary solution to (42) is given by the "Beltrami conditions" that imply the alignment of vorticities and corresponding flows

$$\boldsymbol{U_j} = \mu_j \boldsymbol{\Omega_j} \qquad (j = 1,2) \tag{23}$$

The inverse coefficients that are assumed constant are introduced $a = \mu_1^{-1}, b = \mu_2^{-1}$. Then (23) reads as follows,

$$\boldsymbol{B} = a(\boldsymbol{V} - \epsilon \boldsymbol{\nabla} \times \boldsymbol{B}) \tag{24}$$

$$\boldsymbol{B} + \epsilon \boldsymbol{\nabla} \times \boldsymbol{V} = b\boldsymbol{V} \tag{25}$$

Combining (23) and (24) yields for $\boldsymbol{u} = \boldsymbol{B}$ *or* $\boldsymbol{V}$

$$\epsilon^2 \boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \boldsymbol{V}) + \epsilon(a^{-1} - b)\nabla \times \boldsymbol{u} + (1 - b/a)\boldsymbol{u} = 0 \tag{26}$$

The solution of (26) is given by

$$u = c_+ \boldsymbol{G}_+ + c_- \boldsymbol{G}_- \tag{27}$$

Here "Beltrami fields" are defined as

$$\nabla \times \boldsymbol{G}_\pm = \lambda_\pm \boldsymbol{G}_\pm \tag{28}$$

$$\lambda_\pm = \frac{1}{2\epsilon}[(b^{-1} - a) \pm \sqrt{(b^{-1} - a)^2 - 4(1 - b/a)} \tag{29}$$

In view of (24), the following expressions are obtained

$$\boldsymbol{B} = C_+ \boldsymbol{G}_+ + C_- \boldsymbol{G}_- \tag{30}$$

$$\boldsymbol{V} = (a^{-1} + \epsilon\lambda_+)C_+ \boldsymbol{G}_+ + (a^{-1} + \epsilon\lambda_-)C_- \boldsymbol{G}_- \tag{31}$$

The eigenvalue of the curl operator, that is the parameter $\lambda_\pm$, characterizes the reciprocal of the length scale on which $\boldsymbol{G}_\pm$ changes significantly.

To view the solution and the associated scale lengths as explicit functions of the small parameter ϵ, let $|\lambda_-| = O(1)$ so that $\boldsymbol{G}_-$ varies on the system size. The terms of order of ϵ in (24) - (25) must then be negligible for the $\boldsymbol{G}_-$ - parts of $\boldsymbol{B}$ and $\boldsymbol{V}$ dictating $a \approx b$ to have a significant large-scale component in the solution. Consequently, the inverse of the second scale $\lambda_+ \approx (a - a^{-1})/\epsilon$. Barring the case $a \approx b \approx 1$ (Alfvénic flows – the normalized flow speed of order unity), it is clear that $\lim_{\epsilon \to 0} |\lambda_+| = \infty$, i.e. the short scale shrinks to zero. $|\lambda_-| = O(1)$ occurs only when $a \approx b \approx 1$, and generally one of $|\lambda_\pm|$ becomes very large when ϵ is small [11], [12]. Writing $b/a = 1 + \delta$ [$\delta = O(\epsilon)$, the following approximation is achieved:

$$\lambda_- \approx \frac{\delta}{\epsilon}(a^{-1} - a)^{-1}, \quad \lambda_+ \approx -\frac{1}{\epsilon}(a^{-1} - a) \tag{32}$$

These relations show how the singular part of double Beltrami field is produced, which can lead to large resistive and viscous dissipations [8].

Computation Fluid Dynamics

Although the approach in the last subsection provides reader with a flavour of how the short scales (deemed responsible for large dissipations in corona) originate, the equations (19) - (20) cannot be analytically solved (let alone the system (4) - (8), (12)). So the authors in [10] had to rely on the specific stationary solution of (19) - (20).

In order to investigate more general behaviour of the system numerical simulations are required (See Appendix I: Numerical Schemes for more information on scheme used in the thesis). However, before constructing Magnetohydrodynamic (MHD) simulation it is important to understand how hydrodynamic simulations behave. Moreover plasma flows play crucial role in the problem that we want to ultimately consider. Usually, system of hydrodynamic equations is formulated two ways: in terms of primitive variables and in terms of conservation variables.

To illustrate the first way consider 2D Hydrodynamics. The underlying variables constitute density (ρ), velocity (u, v) and pressure (p). The computational community prefers to write the governing Euler equations in the matrix form,

$$\left(\frac{\partial}{\partial t} + \mathbb{A}\frac{\partial}{\partial x} + \mathbb{B}\frac{\partial}{\partial y}\right)\boldsymbol{V} = 0 \tag{33}$$

Here,

$$\boldsymbol{V} = \begin{pmatrix} \rho \\ u \\ v \\ p \end{pmatrix}, \qquad \mathbb{A} = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & \rho^{-1} \\ 0 & 0 & u & 0 \\ 0 & \rho c_s^2 & 0 & u \end{pmatrix}, \qquad \mathbb{B} = \begin{pmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & \rho^{-1} \\ 0 & 0 & \rho c_s^2 & v \end{pmatrix} \tag{34}$$

With $c_s^2 = \gamma p/\rho$ – speed of sound and $\gamma = c_P/c_V$ – specific heat ratio. However, according to various textbooks this approach is not well suited for shockwave modelling [13]. This is why it is more popular to formulate computational hydrodynamics in terms of conservation variables: mass (ρ), momentum ($\rho u, \rho v$) and energy (ρE) densities.

$$\frac{\partial \boldsymbol{V}}{\partial t} + \frac{\partial \boldsymbol{E}}{\partial x} + \frac{\partial \boldsymbol{F}}{\partial y} = 0 \tag{35}$$

Where,

$$\boldsymbol{V} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \qquad \boldsymbol{E} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho E u + p u \end{pmatrix}, \qquad \boldsymbol{F} = \begin{pmatrix} \rho v \\ \rho v u \\ \rho v^2 + p \\ \rho E v + p v \end{pmatrix}, \qquad p = (\gamma - 1)\left(\rho E - \frac{\rho(u^2 + v^2)}{2}\right) \tag{36}$$

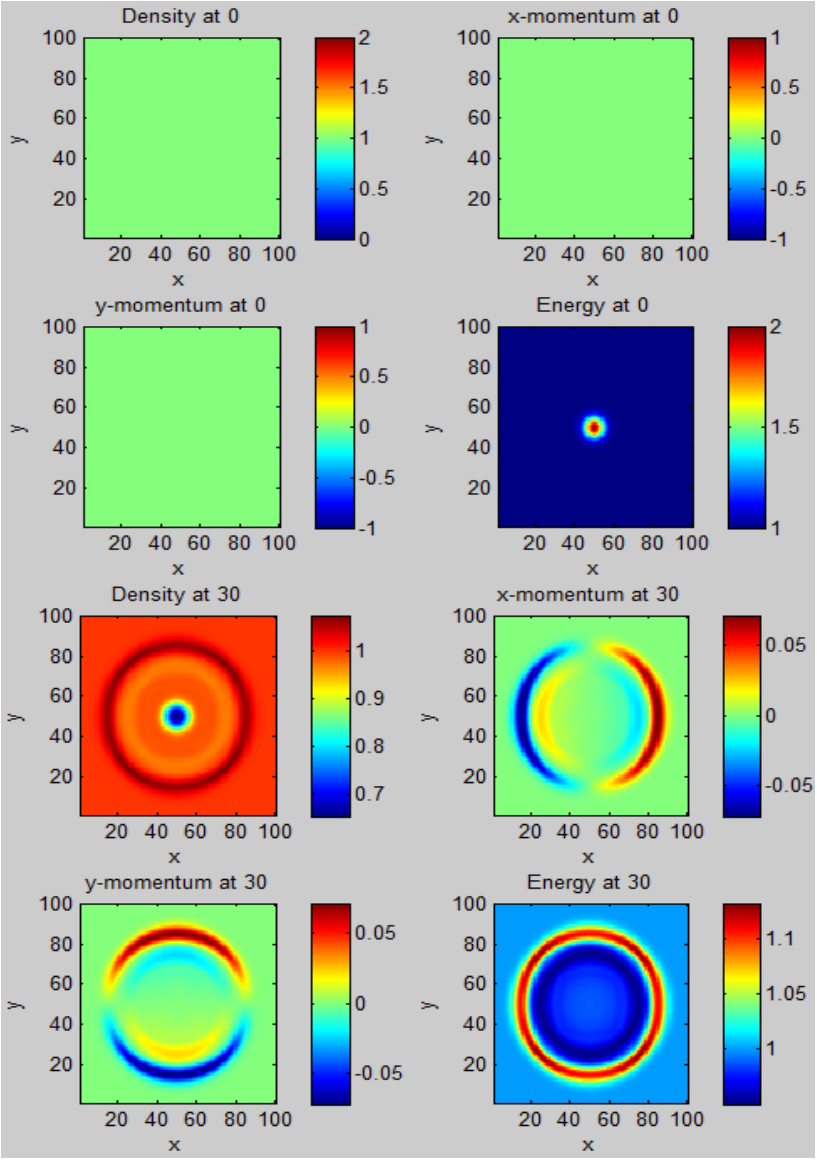

Figure 5: 2D Hydrodynamic simulation carried out using MacCormack scheme on a rectangular grid 100x100. The horizontal and vertical axes represent x and y respectively. The color maps value of 4 fields: density, momentum (x, y) and energy. Red corresponds to maximal values while blue to minimal. Such plots will be used extensively in this thesis. The top 4 plots represent the initial condition while the bottom 4 plots correspond to the final configuration.

The physics behind usefulness of conservation form of the equations for shock capturing can be understood by recalling Rankine - Hugoniot Relations. The thing is that across the shock primitive variables like density or velocity experience discontinuities, otherwise there is no shock. These discontinuities make computations complicated and lead to unsatisfactory spatial oscillations upstream and downstream. On the other hand, the continuity equation for a normal shock wave dictates:

$$\rho_1 u_1 = \rho_2 u_2 \qquad (37)$$

That is, the mass flux (ρu) is constant across the shock wave. Moreover equations of the type (35) are straightforward generalizations of (64) and hence it is easier to apply a standard procedure like (65). This is basically the usefulness behind the conservation form in a nutshell.

Here (Figure 5) is the example of the simulation where the initial condition is set to 0 except for the energy, which implies a perturbation in the pressure. Euler equation supports one wave – sound wave which is generated by this initial condition. Everywhere in this thesis γ is set to $5/3$ as in monoatomic ideal gas. This means that in Figure 5 environment dimensionless pressure equals to $p = 2/3$, according to the last equation in (36). This leads in turn to $c_s \approx 1$ which is in agreement with the 4 bottom plots in Figure 5. In addition to the sound wave the region where the initial perturbation was is now missing more than 30% of the material which

can be explained by the fact that it was carried out by the initial violent compressional wave and could not be compensated since the fluid spreads out in all directions.

This is very simple simulation however. Even though initial condition includes somewhat strong perturbation – as strong as the environment value, the energy of the perturbation is quickly spread over greater circumference and the process becomes linear. It would be awkward if the code failed to simulate such processes.

Indeed, it is problematic to model supersonic flows of several Mach numbers and more using this approach alone. In our problem we will need about 12 [2]. Numerical viscosity (63) which is implicit in the MacCormack scheme (65) - (66) is not enough to deal with such flows and so additional explicit artificial viscosity is added to the code. It might seem artificial indeed to add something to the system of equations. Even though the real viscosity is present in corona it is known to be very small and this creates the problem of explaining observed time scales in the first place. So how can we add the viscosity by hand and claim to explain the riddle?

As turns out we can. Artificial viscosity was first introduced by von Neumann and Richtmyer [14] just to deal with shockwave problems. The crucial aspect of von Neumann's idea is that these additional terms should simulate the effects of the physical viscosity, on the scale of the mesh, locally around the discontinuities and be negligible, that is of any order equal or higher than the truncation error, in smooth regions. The original form of the von Neumann artificial viscosity is usually applied to the Lax-Wendroff scheme (61) - (62), while for the MacCormack scheme (65) - (66) it is better to add third-order artificial viscosity first applied by MacCormack and Baldwin [15] both to corrector and predictor:

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}(\boldsymbol{U})}{\partial x} = \boldsymbol{Q}(\boldsymbol{U}) \tag{38}$$

$$\overline{\boldsymbol{U}}_{\boldsymbol{j}} = \boldsymbol{U}_{\boldsymbol{j}}^{\boldsymbol{n}} - \mathrm{C}\left(\boldsymbol{F}_{\boldsymbol{j+1}}^{\boldsymbol{n}} - \boldsymbol{F}_{\boldsymbol{j}}^{\boldsymbol{n}}\right) + \Delta t\, \boldsymbol{Q}_{\boldsymbol{j}}^{\boldsymbol{n}} + \Delta \boldsymbol{D}_{\boldsymbol{j+1}}^{\boldsymbol{n}}\left(\boldsymbol{U}_{\boldsymbol{j+1}}^{\boldsymbol{n}} - \boldsymbol{U}_{\boldsymbol{j}}^{\boldsymbol{n}}\right) - \Delta \boldsymbol{D}_{\boldsymbol{j}}^{\boldsymbol{n}}\left(\boldsymbol{U}_{\boldsymbol{j}}^{\boldsymbol{n}} - \boldsymbol{U}_{\boldsymbol{j-1}}^{\boldsymbol{n}}\right) \tag{39}$$

$$\overline{\overline{\boldsymbol{U}}}_{\boldsymbol{j}} = \boldsymbol{U}_{\boldsymbol{j}}^{\boldsymbol{n}} - \mathrm{C}\left(\overline{\boldsymbol{F}_{\boldsymbol{j}}^{\boldsymbol{n}}} - \overline{\boldsymbol{F}_{\boldsymbol{j-1}}^{\boldsymbol{n}}}\right) + \Delta t\, \overline{\boldsymbol{Q}_{\boldsymbol{j}}^{\boldsymbol{n}}} + \Delta \overline{\boldsymbol{D}_{\boldsymbol{j}}^{\boldsymbol{n}}}\left(\overline{\boldsymbol{U}_{\boldsymbol{j+1}}^{\boldsymbol{n}}} - \overline{\boldsymbol{U}_{\boldsymbol{j}}^{\boldsymbol{n}}}\right) - \Delta \overline{\boldsymbol{D}_{\boldsymbol{j-1}}^{\boldsymbol{n}}}\left(\overline{\boldsymbol{U}_{\boldsymbol{j}}^{\boldsymbol{n}}} - \overline{\boldsymbol{U}_{\boldsymbol{j-1}}^{\boldsymbol{n}}}\right) \tag{40}$$

$$U_j = \frac{\overline{U}_j + \overline{\overline{U}}_j}{2} \tag{41}$$

$$D_j^n = \varepsilon\left(\left|u_j\right| + a_j\right)\frac{\left|p_{j+1} - 2p_j + p_{j-1}\right|}{p_{j+1} + 2p_j + p_{j-1}} \tag{42}$$

Here Q is the so called source term, that is, anything that cannot be written in conservation form. The artificial viscosity coefficient D depends on speed of the flow $\left|u_j\right|$ and the speed of sound a_j. For more advanced readers though the expression $\left|u_j\right| + a_j$ is easily identified with one of the

characteristics of Euler equations. Notice that artificial viscosity depends on the pressure variations. It is chosen so because one of the reasons why the original MacCormack scheme could not deal with hypersonic flows was that spurious oscillations were generated downstream where the pressure is low and at some point pressure became negative which blew up the solution. Once again, (42) is there just to capture the shock properly, it does not alter significantly anything else, moreover the terms is of the third order. In order to maintain stability the following time step is suggested, which is derived via von Neumann analysis:

$$\Delta t \leq \frac{\Delta x}{|u| + a} \tag{43}$$

The textbooks also suggest choosing $0 \leq \varepsilon \leq 0.5$ for stability. However in a hypersonic regime I had to increase ϵ even more (otherwise the scheme would collapse) and I found that it was possible when Δt was decreased. So when $\varepsilon \geq 0.5$ the criterion (43) is violated. The empirical stability criterion like (43) could not be derived so in the following simulations Δt was chosen empirically small enough for stability to hold. This choice depends on initial and boundary conditions. Scheme (38) - (42) is easily generalized in two dimensions. Last component of each vector in (36) should be modified, though, when the method is generalized on MHD. Namely, instead of simulating energy equation we are simulating equation of state, writing it so that it resembles more the conservation form:

$$\frac{\partial p}{\partial t} + \partial_x(pv_x) + \partial_y(pv_y) + (\gamma - 1)\, p\, (\partial_x v_x + \partial_y v_y) = 0 \tag{44}$$

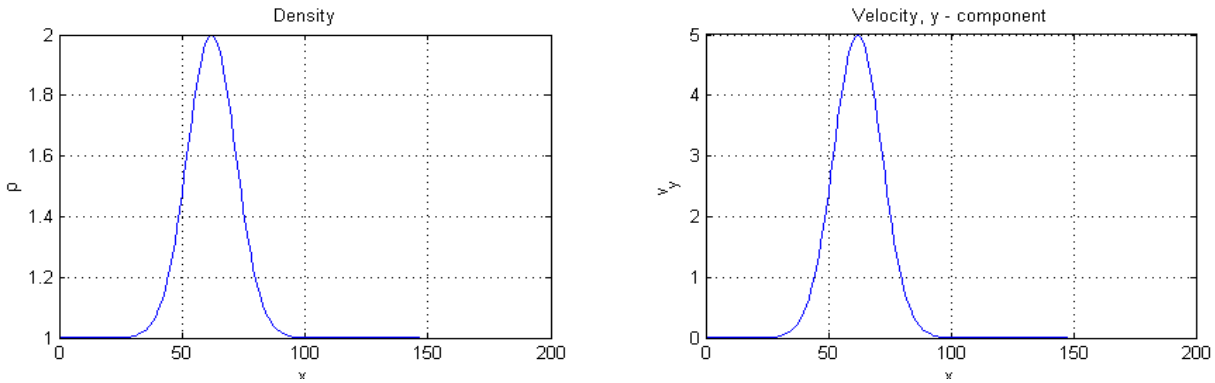

Figure 6: The boundary conditions at the bottom edge.

According to some provisional work that I do not present because I believe it is not as important, I was lead to the conclusion that dimension splitting and differencing sequence does not lead to better results in terms of efficiency of the code in this problem, so I am not using them.

To illustrate the behaviour of shocks in ideal hydrodynamics and also test the behaviour of the code we present here an example. The initial conditions are that of static hydrodynamic equilibrium: $\rho(x,y)=1,\ v(x,y)=0,\ T(x,y)=p/\rho=0.1$. In these units $m_i=1, k_B=1$. The boundary conditions on the other hand are such that the supersonic jet is injected into the fluid (with Gaussian distribution in density and velocity depicted on Figure 6.

The temperature is the same as the environment and the velocity in this example is normalized to 1 and not to 5). The artificial viscosity coefficient $\epsilon=0.5$. The speed of 1 corresponds approximately to 2.5 Mach numbers. The results of the simulation are in Figure 7.

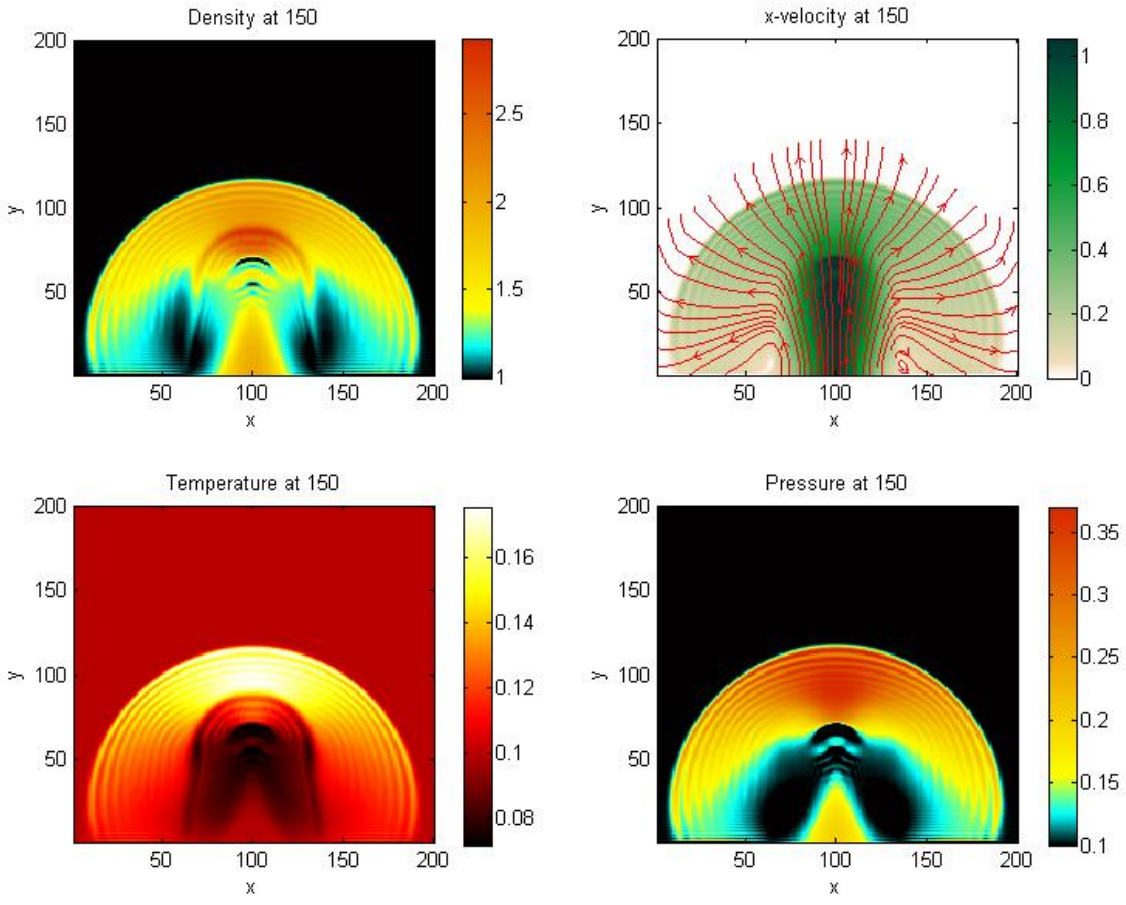

Figure 7: 2D Hydrodynamic simulation carried out using MacCormack scheme on a rectangular grid 200x200. The artificial viscosity coefficient is set to 0.5. The flow emerges from the bottom with the speed of 1. The horizontal and vertical axes represent x and y respectively. The color maps value of 4 fields: density, speed, temperature and pressure. Different colormaps are used for each of the fields. The plot tagged Velocity maps speed of the flow with color and shows the streamlines (red) of the vector field.

In order to investigate the validity of increasing the artificial viscosity (important for capturing hypersonic shocks in strong magnetic fields) the similar simulation is done (Figure 8) where parameter $\epsilon = 20$. The utility of this value of the parameter is justified by the comparison of Figure 8

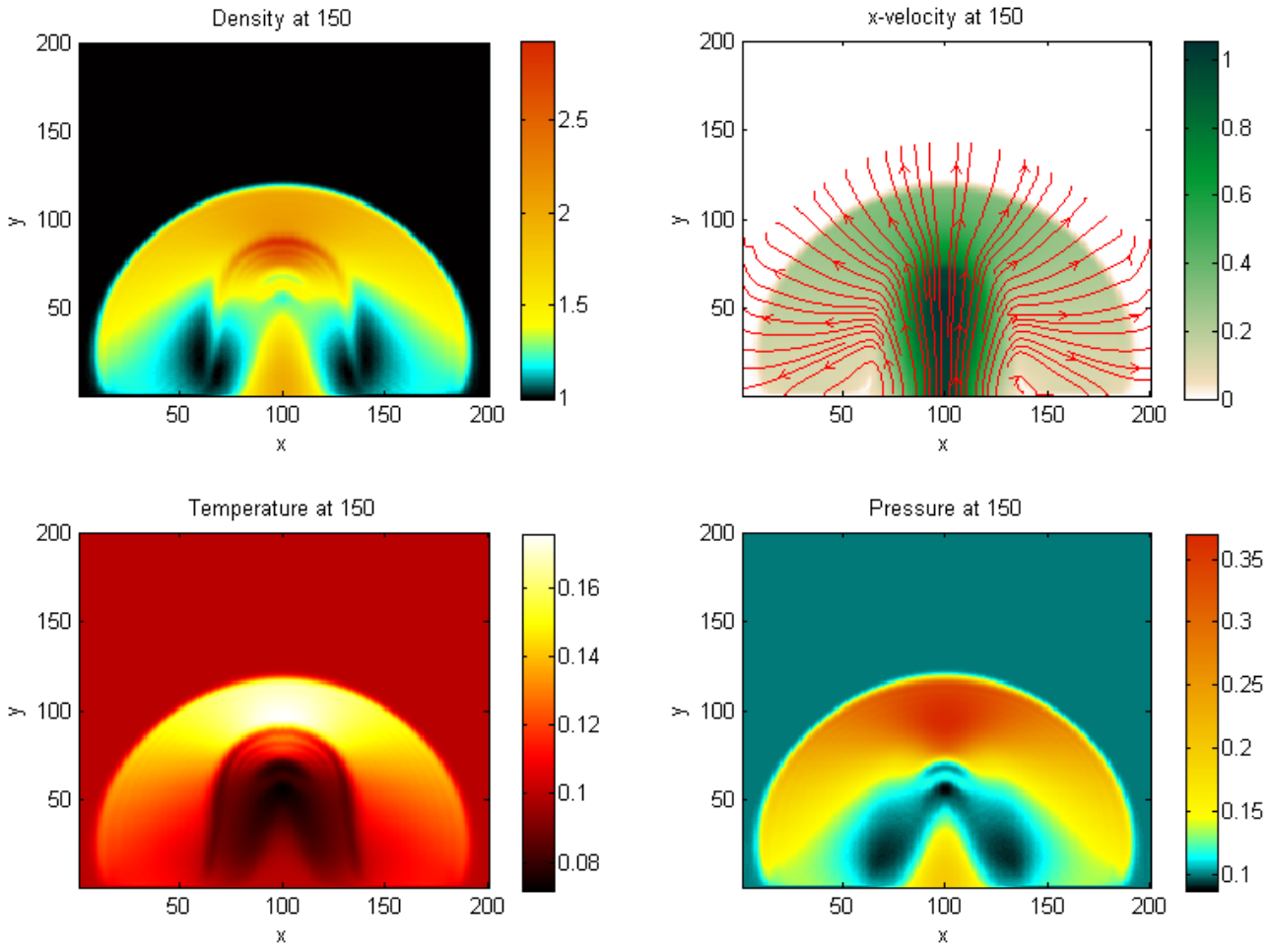

Figure 8: 2D Hydrodynamic simulation carried out using MacCormack scheme on a rectangular grid 200x200. The artificial viscosity coefficient is set to 20. The flow emerges from the bottom with the speed of 1. The horizontal and vertical axes represent x and y respectively. The color maps value of 4 fields: density, speed, temperature and pressure. Different colormaps are used for each of the fields. The plot tagged Velocity maps speed of the flow with color and shows the streamlines (red) of the vector field.

and Figure 7. The basic structure of the flow remained the same. This proves that artificial viscosity does not alter macroscopic physics. However small oscillations downwind the shock front in Figure 7 are absent in Figure 8. In fact, these oscillations are spurious (introduced by numerics [13]) and removing them is the advantage and not the drawback. By the way these oscillations in pressure always lead to violation of stability for hypersonic flows. They become so strong that pressure becomes negative (numerically) in some places and this blows up the solution.

Finally let's increase the velocity of the inlet flow so that it matches more the demands of our problem. Now the boundary condition is exactly that of Figure 6. That is, the velocity at the inlet is 5 which with other parameters present leads to Mach number of approximately 12. The results may be viewed on Figure 9. It readily shows the development of the shock: the flow is slowed down at the passage of the shock which leads to paramount heating.

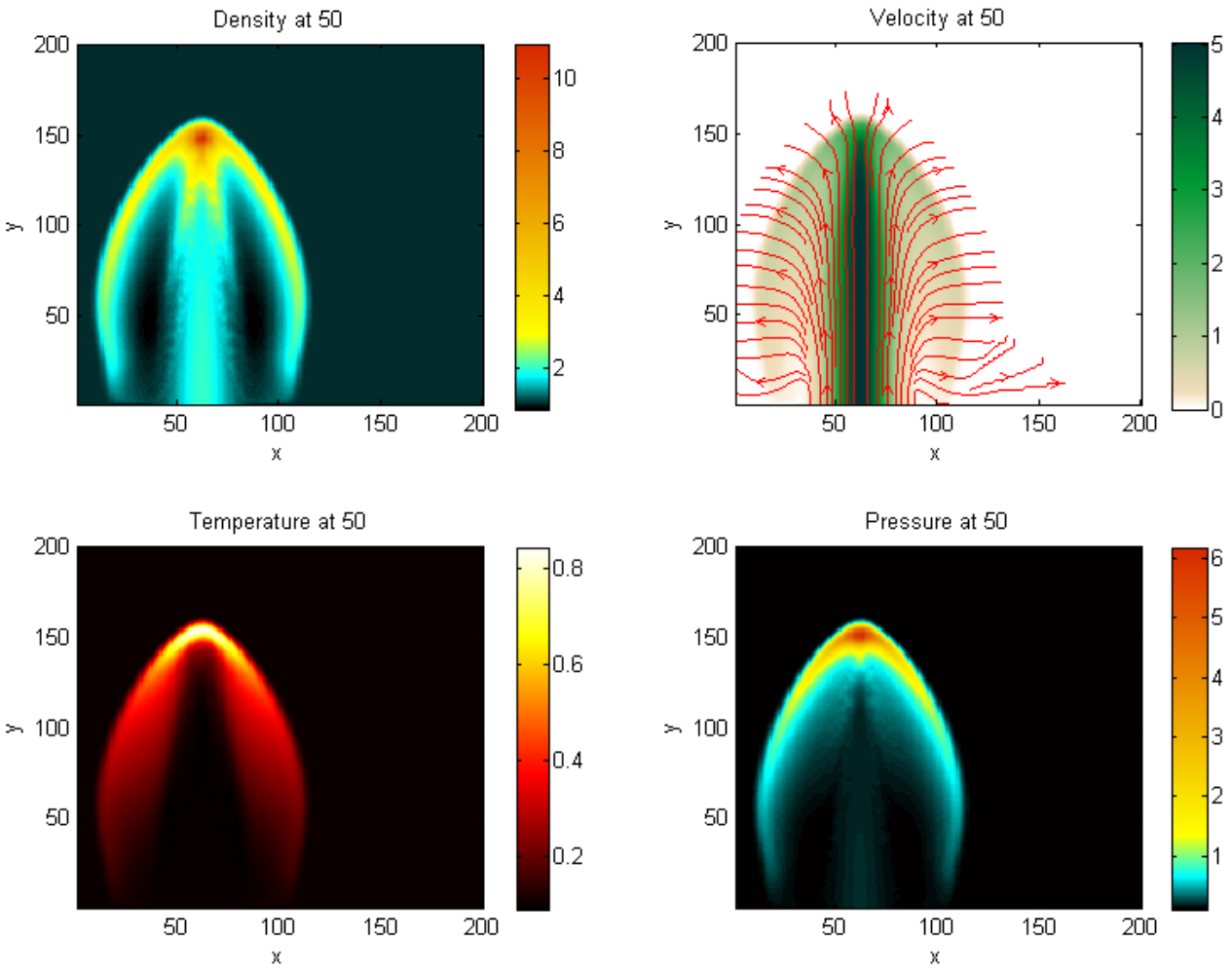

Figure 9: 2D Hydrodynamic simulation carried out using MacCormack scheme on a rectangular grid 200x200. The flow emerges from the bottom with the speed of 5. The horizontal and vertical axes represent x and y respectively. The color maps value of 4 fields: density, speed, temperature and pressure. Different colormaps are used for each of the fields. The plot tagged Velocity maps speed of the flow with color and shows the streamlines (red) of the vector field.

2D MHD

The description, carried out in last chapter, is not full without the addition of magnetic field, since it is present in corona. We seek to reproduce the injection of flows in the magnetic field anchored

in corona. It is fairly difficult to simulate (4) - (8), (12) system so to get the flavour first consider a simplified 2D version where $\alpha \to 0, v_i \to 0$. This can be validly considered as first approximation since these terms play lesser role in the primary heating (the secondary heating is not a matter of investigation of this thesis). The real viscosity plays most prominent role near the shock and its effects are modelled by artificial viscosity (otherwise very fine grid has to be used). The gravity is also neglected for simplicity. The following system is obtained then,

$$\frac{\partial n}{\partial t} + \partial_j(nv_j) = 0 \tag{45}$$

$$\frac{\partial}{\partial t}(nv_i) + \partial_j\left(\mathrm{n}v_iv_j + \left(p + \frac{B^2}{2}\right)\delta_{ij} - B_iB_j\right) = 0 \tag{46}$$

$$\frac{\partial B_i}{\partial t} + \partial_j(v_jB_i - B_jv_i) = 0 \tag{47}$$

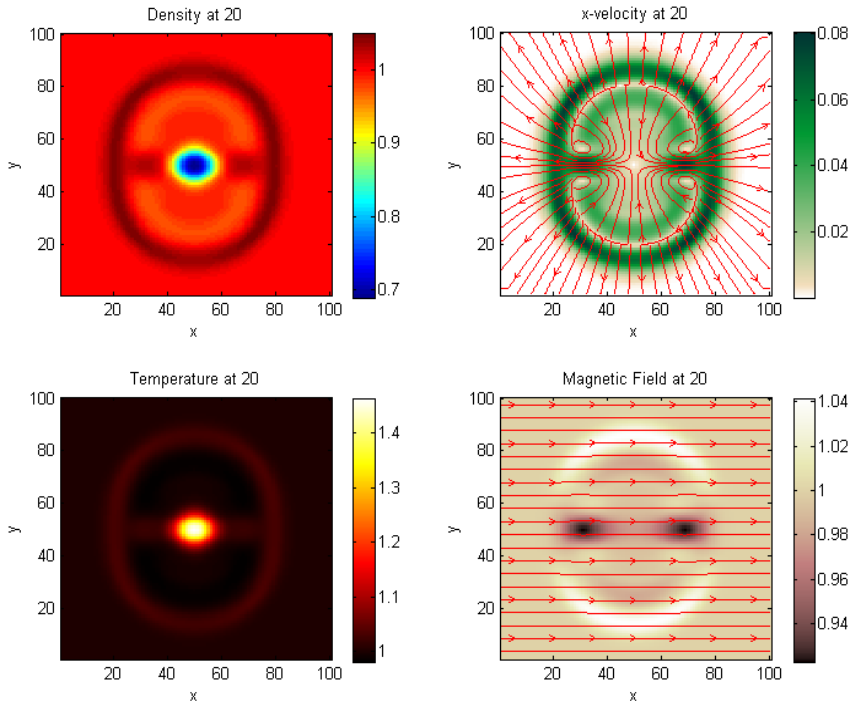

Figure 10: Ideal MHD simulation (Grid size 100x100) with aid of MacCormack method. The initial Gaussian perturbation in pressure spreads as a MHD wave. The phase velocity in agreement with the dispersion relation derived in linear MHD.

Together with (44), these equations (corresponds to 0-model of (19) - (20)) constitute a set of ideal MHD equations. Notice that they are written so that terms including magnetic field are also in conservation form. This helps guarantee that most of the time $\nabla \cdot \boldsymbol{B} = 0$. Artificial viscosity is not applied to magnetic component. It can be but it leads to results that are little bit worse.

Suppose that we want to make an experiment like on Figure 5 but with magnetic field pointing, say, in x direction and to a newly constructed code that deals with (44) - (47). Magnetohydrodynamics supports MHD waves and since the initial condition leads to compression it is natural that compressional waves should be exited. Errors in magnetic divergence conservation are negligible ($\lesssim 10^{-4}$). The results of such simulation are in Figure 10. The initial condition is the same as in Figure 5 but leads to different results. It seems that because sound and Alfven speed are of the same order (relatively

close) the initial sound wave (during nonlinear stage) excites MHD modes so that the dispersion relation of MHD waves is satisfied (Figure 5).

Finally, we must concern ourselves with injecting a flow in the vicinity of the loop. In order to make sure that strong fields under discretization do not produce a lot of spurious acceleration the MacCormack viscosity is slightly modified with addition of Alfven velocity:

$$D_j^n = \varepsilon\left(|u_j| + a_j + V_A\right)\frac{|p_{j+1} - 2p_j + p_{j-1}|}{p_{j+1} + 2p_j + p_{j-1}} \tag{48}$$

Moreover, the bottom 5 lines of grid are kept at constant density, velocity and pressure, the magnetic field is allowed to change. This deals with the problem.

The transformation from real to dimensionless variables is defined as:

$$\begin{gathered}\boldsymbol{r} \to \boldsymbol{r}R\,; \quad t \to t\,\frac{R}{V_A}; \quad \boldsymbol{B} \to \boldsymbol{B}B_\odot; \quad T \to T\beta^{-1}T_\odot; \\ n \to nn_\odot; \quad \boldsymbol{V} \to \boldsymbol{V}V_A;\end{gathered} \tag{49}$$

The parameters are defined as R – special scale; $B_\odot = 20\ Gauss$ – the magnetic field at the bottom of the arcade, $n_\odot = 4 \cdot 10^8 cm^{-3}$, $T_\odot = 3\ eV = 4.8 \cdot 10^{-12} ergs$ – concentration and temperature of the plasma flow emerging from the chromosphere and the rest are:

$$c_s^2 = \frac{2T_\odot}{m_i}; \quad V_A^2 = \frac{B_\odot^2}{4\pi n_\odot m_i}; \quad \beta = \frac{c_s^2}{V_A^2}; \tag{50}$$

Here ion mass is assumed to coincide with proton mass (completely ionized hydrogen plasma): $m_i = m_p = 1.67 \cdot 10^{-24} g$. This leads to $c_s = 2.4 \cdot 10^6\ cm/s$ and $V_A = 2.2 \cdot 10^8 cm/s$ .The usefulness behind the dimensionless variables is, first of all, that the results obtained in one simulation can be re-scaled in a straightforward manner and describe the processes on larger/smaller scale. For example, suppose all parameters but R are left unchanged and R increases twice. This means that the same results apply for this greater scale but will take twice longer - t scales proportional to R. Secondly, it is easier to program the code this way: R is chosen so that special steps $\Delta x = \Delta y = \Delta z = 1$.

The initial condition for the magnetic field is that of a conducting current running parallel to the plane of the page (Figure 11). This configuration is quite common in solar corona. The results are on Figure 12. The magnetofluid coupling occurs so the flow is trapped.

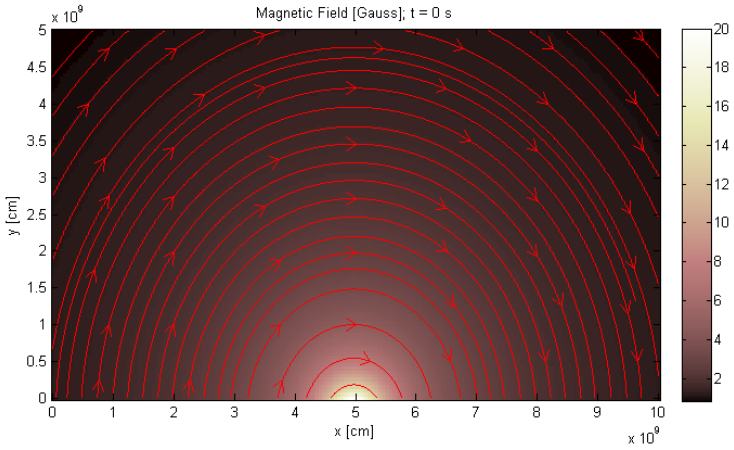

Figure 11: Streamline plot of the Magnetic Field combined with color plot representing the strength of the magnetic field. The field has maximum $B_{max} = 20\ Gauss$ at $x = 5 \cdot 10^9 cm; y = 0\ cm$.

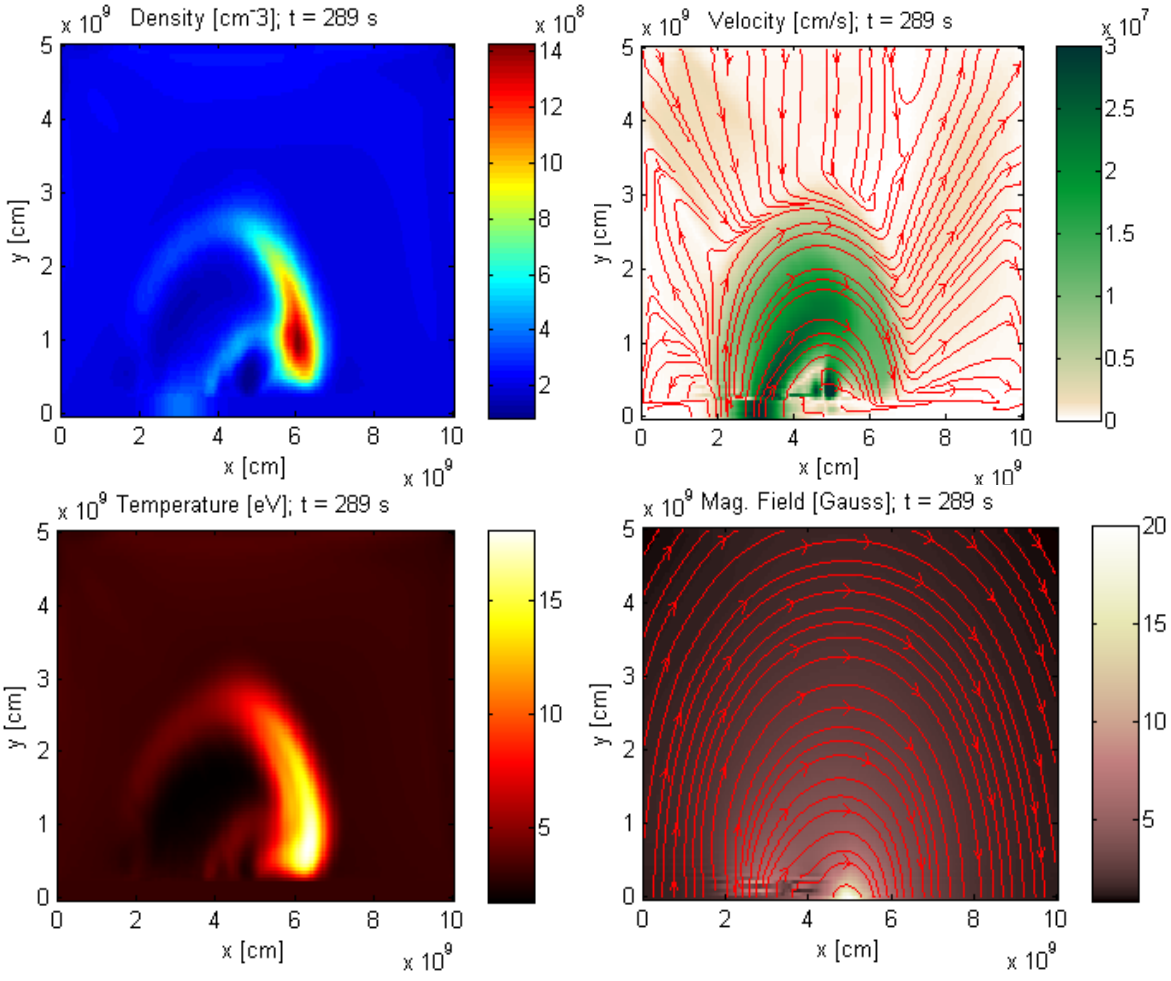

Figure 12: The Coronal structure is born. The four plots correspond to Density (concentration), Velocity, Temperature and Magnetic field. The simulations are performed via MacCormack scheme in 2D. Grid size in this simulation was 200x100. Simulations with greater grid like 400x200 produce the same results, with greater resolution. Such resolution in 3D would be too heavy, so because we seek to compare 2D and 3D we do not include 400x200 in the thesis.

3D Modelling of coronal structures

Finally, the 2D scheme is generalized in three dimensions. Again, explicit MacCormack method is used to model MHD equations (45) - (47).

$$\bar{\boldsymbol{U}}_{i,j,k} = \boldsymbol{U}^{n}_{i,j,k} - \Delta\left(\boldsymbol{E}^{*n}_{i+1,j,k} - \boldsymbol{E}^{*n}_{i,j,k} + \boldsymbol{F}^{*n}_{i,j+1,k} - \boldsymbol{F}^{*n}_{i,j,k} + \boldsymbol{G}^{*n}_{i,j,k+1} - \boldsymbol{G}^{*n}_{i,j,k}\right) + \Delta t\, \boldsymbol{Q}^{n}_{i,j,k} \tag{51}$$

$$\boldsymbol{U}^{n+1}_{i,j,k} = \left[\overline{\boldsymbol{U}^{n}}_{i,j,k} + \boldsymbol{U}^{n}_{i,j,k} - \Delta\left(\overline{\boldsymbol{E}^{*n}_{i,j,k}} - \overline{\boldsymbol{E}^{*n}_{i,j-1,k}} + \overline{\boldsymbol{F}^{*n}_{i,j,k}} - \overline{\boldsymbol{F}^{*n}_{i,j-1,k}} + \overline{\boldsymbol{G}^{*n}_{i,j,k}} - \overline{\boldsymbol{G}^{*n}_{i,j,k-1}}\right) + \Delta t\, \overline{\boldsymbol{Q}^{n}_{i,j,k}}\right]/2 \tag{52}$$

$$\boldsymbol{E}^{*n}_{i,j,k} = \boldsymbol{E}^{n}_{i,j,k} + \varepsilon\left(|u_{ijk}| + c_{s\,ijk} + V_{Aijk}\right)\frac{\left|p_{i+1,j} - 2p_{i,j,k} + p_{i-1,j,k}\right|}{p_{i+1,j,k} + 2p_{i,j,k} + p_{i-1,j,k}}\left(\boldsymbol{U}^{n}_{i,j,k} - \boldsymbol{U}^{n}_{i-1,j,k}\right) \tag{53}$$

$$\overline{\boldsymbol{E}^{*n}_{i,j,k}} = \overline{\boldsymbol{E}^{n}_{i,j,k}} + \varepsilon\left(|\overline{u_{ijk}}| + \overline{c_{s\,ijk}} + \overline{V_{Aijk}}\right)\frac{\left|\overline{p_{i+1,j}} - 2\overline{p_{i,j,k}} + \overline{p_{i-1,j,k}}\right|}{\overline{p_{i+1,j,k}} + 2\overline{p_{i,j,k}} + \overline{p_{i-1,j,k}}}\left(\overline{\boldsymbol{U}^{n}_{i+1,j,k}} - \overline{\boldsymbol{U}^{n}_{i,j,k}}\right) \tag{54}$$

Writing out y and z fluxes, correspondingly $\boldsymbol{F}$ and $\boldsymbol{G}$, is unenlightening, they coincide with (53) - (54) with a corresponding change of indices.

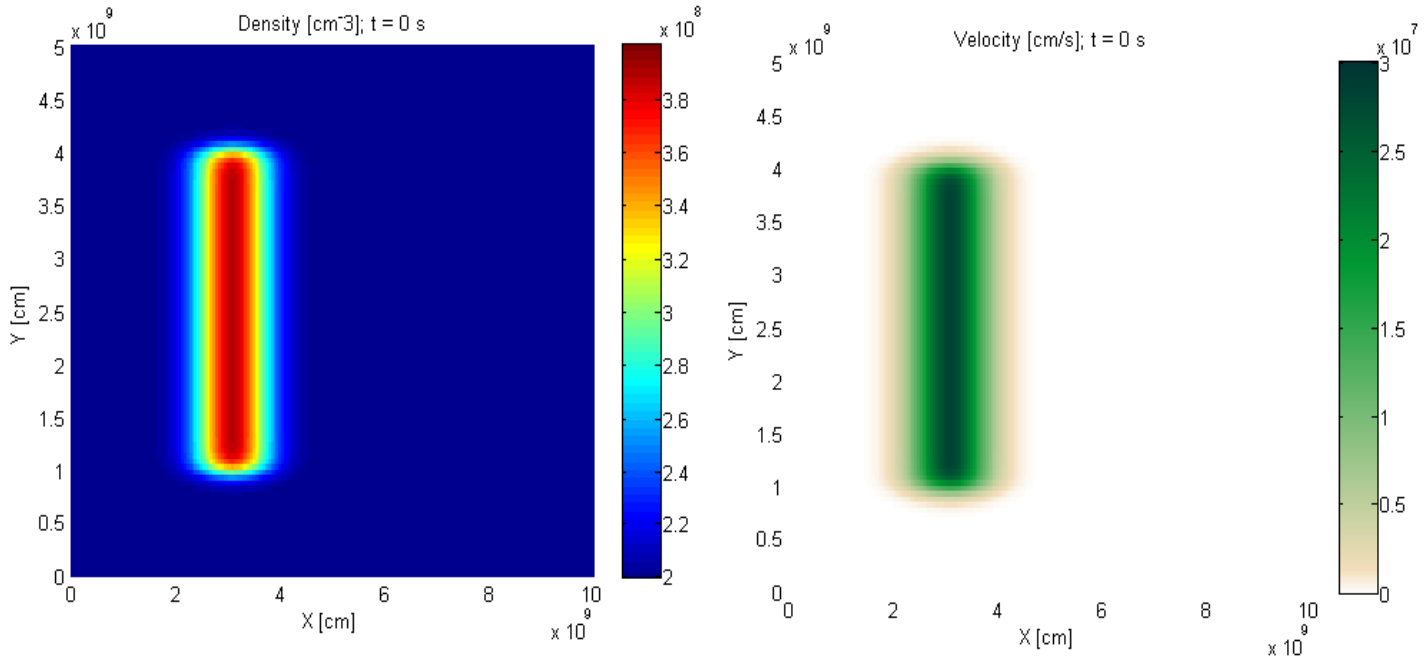

Figure 13: Boundary condition at the bottom of the grid. The flow that is Gaussian in X and uniform in Y is injected in the loop. Left – Density and Right – z – component of Velocity, when measured in Gaussian units (CGS). Components x and y are kept zero.

Now that that the method is thoroughly explained and equations are modelled it is time to apply it to our problem. First we want to make sure that 3D generalization of the algorithm produces results that are similar to that of Figure 12. So the most reasonable thing to do is to feed the

algorithm (51) - (54) the initial and boundary conditions similar to 2D, hence they have to be cylindrical in y – direction. Such boundary conditions can be viewed on Figure 13.

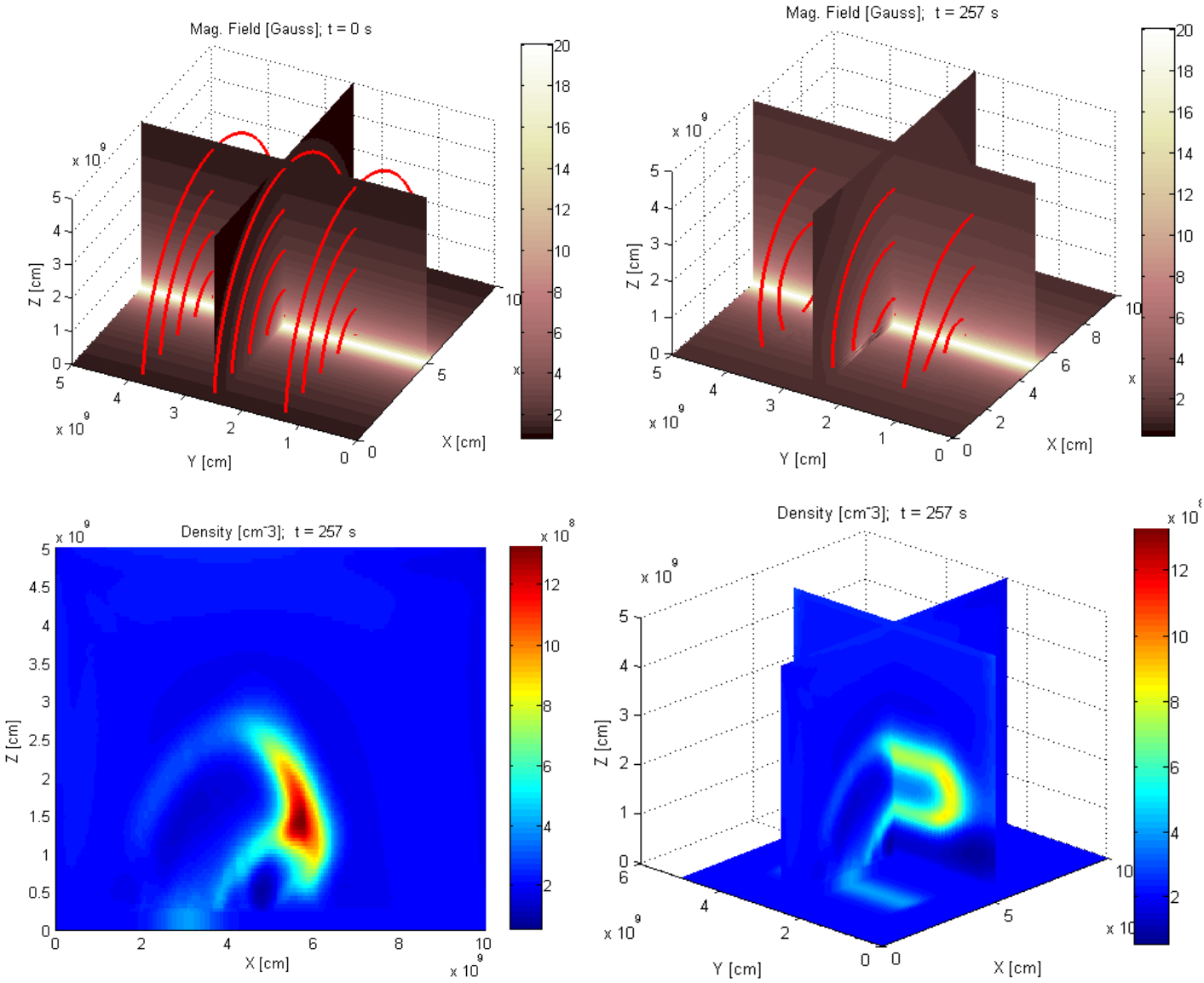

Figure 14: Upper Left plot represents the initial arcade-like configuration of the magnetic field: both the magnitude represented by color and the fieldlines represented by red curved lines. In order to better represent the 3-dimensional scalar field (magnitude of magnetic field is scalar) three cross-sections are made: first in XY plane, second in YZ plane and third in XZ plane. Geometrical location of the cross-sections is chosen to better represent the information. Upper left plot represents the state of magnetic field when the coronal structure is formed. Lower Left plot represents density at the same time in the XZ plane so that it is easier to compare it to purely 2D case. Lower Right plot shows density from a different perspective so that all three cross-sections are visible. The axes readily show what the current view is.

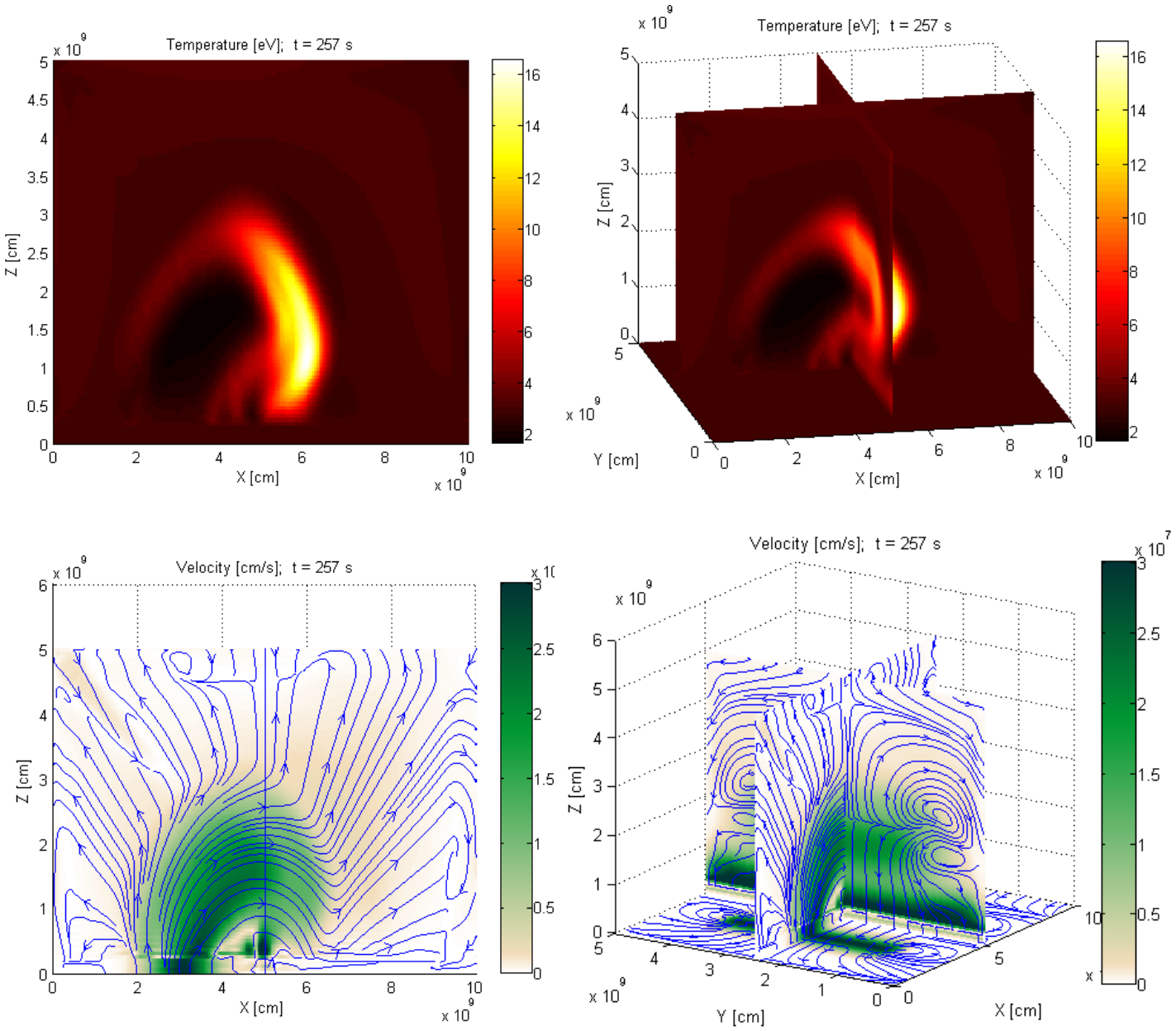

Figure 15: Upper left and upper right plots represent temperature from different perspectives; the same methodology is used to represent the results like in the previous figure. The lower left plot shows the velocity magnitude in color and the streamlines represent the vector field. The lower right plot is the view from different perspective where three cross-sections are made again. Notice that streamlines always point in the plane of the cross-sections. This means that they show only the projection of the real velocity onto the cross-section plane. Thus they do not present all information about the flow, but the information can be inferred from the symmetry of the problem.

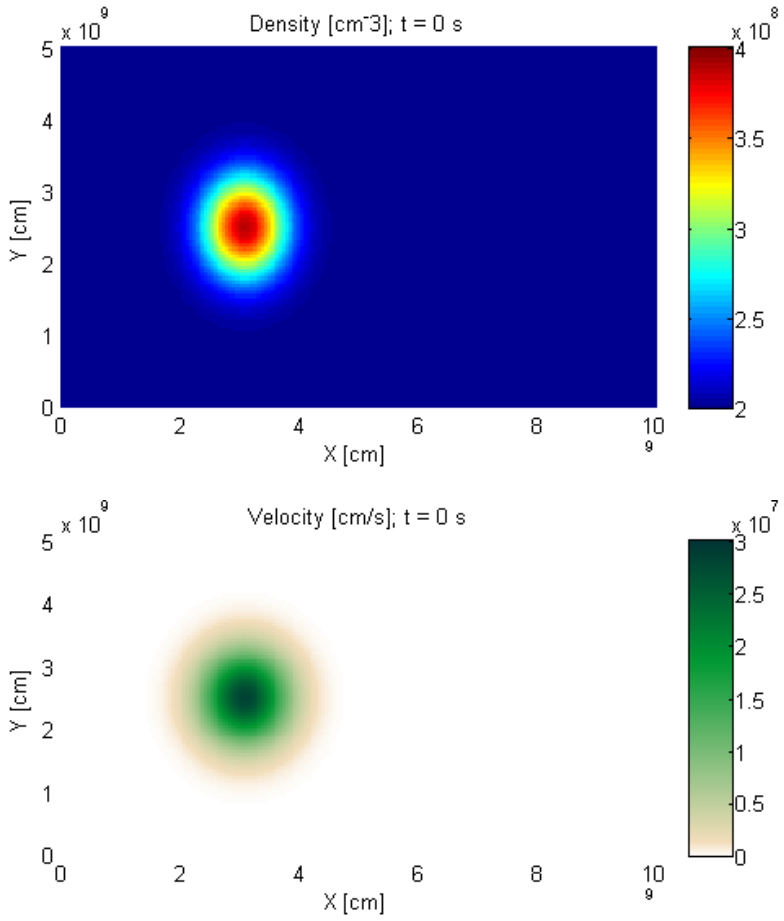

Figure 16: Boundary condition at the bottom of the grid. The flow that is Gaussian in both X and Y is injected in the loop. Up – Density and Down – z – component of Velocity, when measured in Gaussian units (CGS). Components x and y are kept zero.

The initial configuration of the magnetic field is presented in the upper left plot on Figure 14. The method we are using here does not allow to be fed with boundary conditions that are exactly constant in y, for that would mean a supersonic flow is fed in the vicinity of the boundary – such boundary conditions would lead to instability – numerical crush of the simulation. Instead conditions are chosen so that the structure injected in the loop is made as long as possible (Figure 13). As a result, after typically $5\,min$ of physical time have passed the structure presented on Figure 14 and Figure 15 is born. The results are truly consistent with Figure 12 obtained in 2D near the axis of symmetry of y – where the flow is injected. The farther away one goes the more deviations from 2D one observes, which is also reasonable. Now that we've made clear what happens in cylindrical geometry it is time to start from spherically symmetrical flow injected in the loop.

The boundary condition now is the flow with a Gaussian distribution of density and velocity both in x as well as y (Figure 16). Now more deviations from Figure 12 are expected. The magnetic field configuration is again the same (top left of Figure 14) and stays mostly constant throughout time (like in the previous example), so its evolution is not presented in this case. After 5 minutes the coronal structure is born again (Figure 17). The results for this flow are also mostly similar to the cylindrical case (Figure 14 and Figure 15), which shows the scope of application of the coronal structure formation described here.

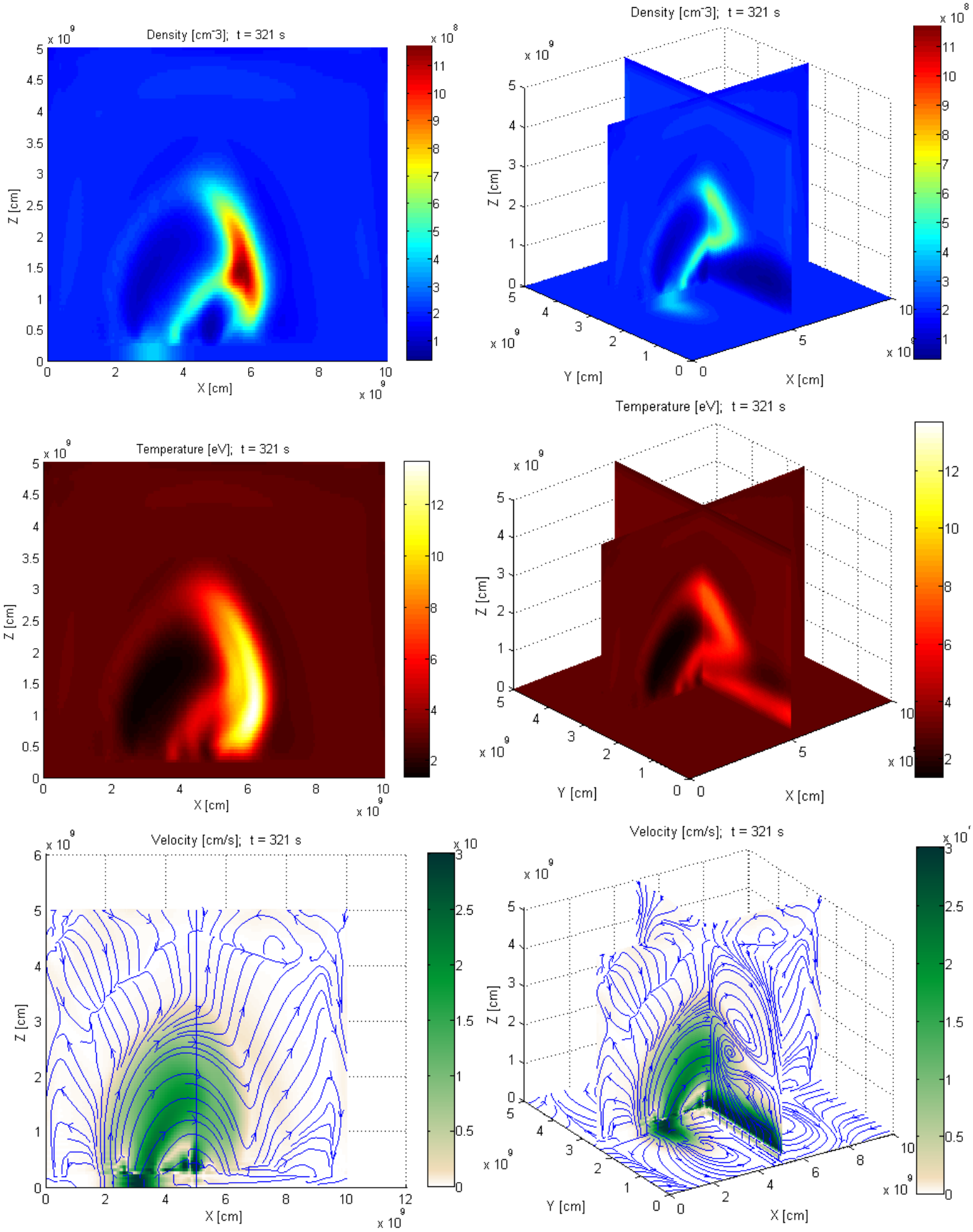

Figure 17: Coronal structure that is formed when flow that is Gaussian in X and Y is injected at the foot of the loop.

Conclusions, comparison and discussion for future

Bibliography

[1] G. Noci, E. Antonucci J. L. Kohl, *Solar Physics*, 1997.

[2] S. R. Habbal, J. L. Kohl, and G. Noci X. Li, "Astrophys. J. Lett.," 1998.

[3] A. M. Title, T. E. Berger, L. Fletcher, N. E. Hurlburt, R. W. Nightingale, R. A. Shine, T. D. Tarbell, J. Wolfson, L. Golub, J. A. Bookbinder, E. E. Deluca, R. A. McMullen, H. P. Warren, C. C. Kankelborg, B. N. Handy, and B. DePontieu C. J. Schrijver, "Solar Physics," 1999.

[4] C. Gontikakis, M.K. Georgoulis, C.E. Alissandrakis, K. Tsinganos P. Syntelis, "Study of the three-dimensional shape and dynamics of coronal loops observed by Hinode/EIS," *arXiv:1205.0126v1*, June 2012.

[5] Shatashvili N.L., Mikeladze S.V., and Sigua K.I. Mahajan S. M., "Acceleration of plasma flows in the closed magnetic fields: Simulation and analysis," vol. 13, no. 6, 2006.

[6] E. N. Parker, "The Astrophiscal Journal," 1972.

[7] S. Patsourakos and J.-C Vial, "Intermittent behavior in the transition region and the low corona of the quiet Sun," *Astronomy and Astrophysics*, 2002.

[8] R. Miklaszewski, K.I Nikol'skaya, N.L. Shatashvili S.M. Mahajan, "Formation and primary heating of the solar corona: Theory and simulation," *Physics of Plasmas*, vol. 8, April 2001.

[9] S. M. Mahajan and N. L. Shatashvili, "Comments on the "The Coronal Heating Paradox" by M.J. Aschwaden, A. Winebager, D. Tsiklauri and H. Peter," *arXiv:0709.1535v1*, Sep 2007.

[10] S. M. Mahajan, S. Ohsaki Z. Yoshida, "Scale hierarchy created in plasma flow," *Physics of Plasmas*, vol. II, no. 7, July 2004.

[11] N. L. Shatashvili, Z. Yoshida, and S. M. Mahajan S. Ohsaki, *Astrophys. J. Lett.*, 2001.

[12] N. L. Shatashvili, Z. Yoshida, and S. M. Mahajan S. Ohsaki, *Astrophys. J.*, 2002.

[13] T. J. Chung, *Computational Fluid Dynamics*. Cambridge, New York: Cambridge University Press, 2010.

[14] R. D. Richtmyer J. Von Neumann, "A method for the numerical calculations of hydrodynamical shocks," *Journal of Mathematical Physics*, 1950.

[15] B.S. Baldwin R.W. MacCormack, "A numerical method for solving the Navier-Stokes equation with application to shock-boundary layer interaction," *AIAA Paper*, 1975.

[16] S. A. Teukolsky, W. T. Vetterling, B. P Flannery W. H. Press, *Numerical Recipes in C.*: Cambridge University Press, 1992.

[17] R. W. MacCormack, "The effect of viscosity in hypervelocity impact cratering," *AIAA Paper*, 1969.

Appendix I: Numerical Schemes

The numerical solution of the system of partial differential equations (we shall use term PDE henceforth) is not as straightforward as it seems. Before we undertake such a bold mission, it is important to consider simplifications and build the scheme from bottom to top. The system of interest is nonlinear but we shall first consider methods for solving linear problems.

Linear PDEs can be subdivided into three subcategories [16] according to their peculiar behaviour:

Type	Hyperbolic	Parabolic	Elliptic
Example	$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2}$	$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D \frac{\partial u}{\partial x}\right)$	$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x.y)$

The subcategories are due to how characteristics of the PDEs propagate. Physically this means that different types of equations behave differently. Hyperbolic and parabolic equations belong to class of initial value problems that describe the time evolution of certain physical process, while elliptic equations belong to boundary value problems that describe static solutions. Each of the subcategories is treated differently in numerical simulations. Our problem is of mixed type, however it can be identified most closely with hyperbolic type.

There are several different methodologies to approach the solution of initial value problems. The first is the Finite Difference Method where the space is represented as fixed usually rectangular grid (continuous solutions are impossible numerically). Finite Elements Method allows modifying the grid shape and adjusting it to mimic the distribution of fields or the peculiar boundary conditions. Hence it is used most often in solid mechanics and structural engineering. Finally, there is a class of spectral methods; however these methods are not well suited for discontinuous problems [16].Thus we choose the Finite Difference Method as the most straightforward and simplest approach.

Discretization of originally continuous equations is rather non-trivial problem that requires special care. For instance, consider the simplest example of the hyperbolic problem – one dimensional advection equation:

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0 \tag{55}$$

It has elegant analytical solution $u = f(x - vt)$, u is transported with velocity v. Imagine one attempts to discretize time with forward Euler derivative and space with central Euler derivative – FTCS (Forward Time Centred Space) Scheme:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -v\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta \mathrm{x}} + O(\Delta t, \Delta x^2) \tag{56}$$

Although the accuracy suggests the numerics would converge with truncation error of the order $(\Delta t, \Delta x^2)$, this attempt is going to fail because FTCS scheme is unstable when applied to advection equation. The problem is that the short scales, introduced by discretization, that are not of interest exponentially grow and, as a result, large scale solution is spoiled very fast. To roughly explain why that happens consider von Neumann stability analysis. The independent solutions, or eigenmodes, of the difference equation (56) are all of the form:

$$u_j^n = \xi^n \exp(i\, kj\Delta x) \tag{57}$$

Here ξ is the complex amplitude and k the wave number. To find the amplitude substitute modes into difference equation. This leads to the following expression for ξ:

$$\xi(k) = 1 - i\frac{v\Delta t}{\Delta x}\sin k\Delta x \tag{58}$$

For all $k, \Delta t$ or Δx $|\xi(k)| > 1$ thus u_j^n will grow unlimitedly grow with time. The von Neumann analysis can be extended to nonlinear problems as well expanding $u = u_0 + \delta u$ and assuming that u_0 already satisfies the stability criterion for the linear difference equation, analysis would look for an unstable eigenmodes of δu.

Consider a variation of the scheme discussed above whereby special derivative is approximated with backward difference. Such scheme is usually referred as (FTBS) Forward Time and Backward Space or the first order upwind scheme.

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -v\frac{u_j^n - u_{j-1}^n}{\Delta \mathrm{x}} + O(\Delta t, \Delta x) \tag{59}$$

One might wonder what decreasing special accuracy by one order might improve but as experience shows computer modelling is often art more than science, so unexpected results often appear. Von Neumann analysis shows that the scheme is stable when

$$\frac{v\Delta t}{\Delta x} \leq 1 \tag{60}$$

This inequality is called Courant – Friedrich – Lewy stability criterion or simply Courant condition. There is also physical explanation why (59) scheme is superior to (56). Namely in (56) the information is allowed to propagate in both directions, whereas in (59) the information propagates in the direction of transport. Incidentally, if the sign of v is negative forward differencing (downwind scheme) is required in the right hand side of (59).

Second order upwind/downwind schemes are more complicated so we are going to consider second order central schemes that are also stable when (60) holds. They generally belong to the family of multistep, or predictor – corrector methods.

Lax – Wendroff Multistep Scheme with $C = v\Delta t/\Delta x$:

Step 1: $$u_{j+1/2}^{n+1/2} = \frac{1}{2}\left(u_{j+1}^{n} + u_{j}^{n}\right) - \frac{C}{2}\left(u_{j+1}^{n} - u_{j}^{n}\right) \tag{61}$$

Step 2: $$u_{j}^{n+1} = u_{j}^{n} - C\left(u_{j+1/2}^{n+1/2} - u_{j-1/2}^{n+1/2}\right), \qquad O(\Delta t^2, \Delta x^2) \tag{62}$$

In order to understand intuition behind choosing (61) - (62) scheme substitute (61) into (62) to get

$$u_{j}^{n+1} = u_{j}^{n} - \frac{C}{2}\left(u_{j+1}^{n} - u_{j-1}^{n}\right) + \frac{C^2}{2}\left(u_{j+1}^{n} - 2u_{j}^{n} + u_{j-1}^{n}\right) + O(\Delta t^2, \Delta x^2) \tag{63}$$

In such representation Lax-Wendroff scheme looks like (56) modified by presence of numerical viscosity. The numerical viscosity is of second order so it is there just to dissipate short scales that we are not interested it, so that they do not ruin large scales. Alternative would be to investigate stability via von Neumann analysis which leads to Courant – Lewy condition (60).

Lax-Wendroff method is well suited for nonlinear problems; however a small modification leads to even better results. It was introduced by MacCormack [17]. Let the equation of interest be

$$\frac{\partial u}{\partial t} + \frac{\partial F(u)}{\partial x} = 0 \tag{64}$$

Then the MacCormack method consists of:

Step 1: Predictor
$$u_j^* = u_j^n - \mathrm{C}\left(F_{j+1}^n - F_j^n\right) \tag{65}$$

Step 2: Corrector
$$u_j^{n+1} = \frac{1}{2}\left[u_j^n + u_j^* - \mathrm{C}\left(F_j^* - F_{j-1}^*\right)\right], \qquad O(\Delta t^2, \Delta x^2) \tag{66}$$

In order to compare efficiency of central schemes consider $F = u$ which has a known solution. Let the initial condition be a simple step from 1 to 0 in u. The advection equation tells us that this disturbance should travel unaltered forward.

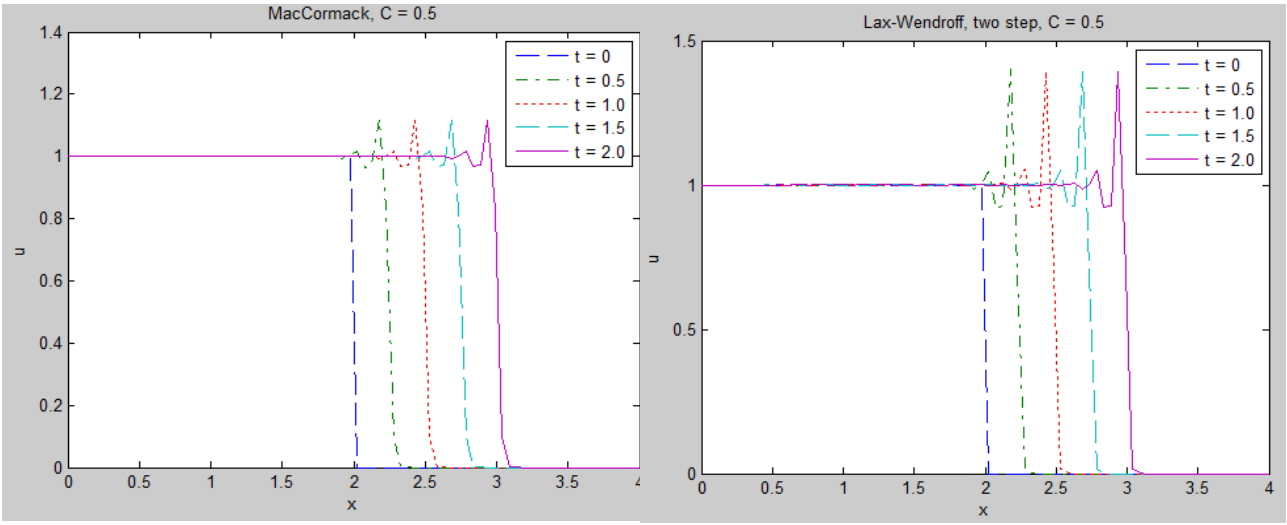

Figure 18: Numerical solutions of advection equation. MacCormack seems to be better.

Figure 18 readily shows the superiority of MacCormack algorithm.

Generally, errors in numerics are divided into truncation and round-off errors. Truncation errors decrease with the step size. The round-off errors are produced because of rounding expression to a finite number of digits in the arithmetic operations. The latter may seem innocuous, however for very fine grids, repetitive operations they may cause the solution to systematically diverge from the exact one. To exclude these we are using double precision calculations (8 bytes).

Finally the scheme and the step size have to be chosen so that stability condition is not violated. Sometimes this forces small temporal step sizes and hence very long computation. The cure to this is implementation of implicit scheme as opposed to explicit, which was already explained. With implicit schemes stability is no longer an issue, so the step size can be chosen larger. However, they

are more difficult to implement and require solution of system of equations throughout the grid, so in some situations may be more trouble than worth. Hence we are using explicit scheme.

Appendix II: Simulation Code

```
/* Load Additional Libraries */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

/* Declaring Functions and Procedures */

double* File2Array(char* filename, int* num);
double**** File2Array(char* filename, int X, int Y, int Z, int dim);
double**** AllocateArray(int n1, int n2, int n3, int n4);
double*** AllocateArray(int n1, int n2, int n3);
void Var2File(double, char*, const char*);
void Array2File(double*** ,char* ,int , int ,int , const char*);
void Forw_Pred(double ,int ,int ,int,double ,double , double ****,
        double ****, double ***, double ***, double ***, double ***,
        double ***, double ***);
void Back_Corr(double ,int ,int , int,  double, double, double ****,
        double ****, double ***, double ***, double ***, double ***,
        double ***, double ***);
void Prim_Calc(double, int , int , int , double ****, double ***,
        double ***, double ***);
void Bound_Cond(int , int , int , double ***, int);

/* Main Function */

int main(void)
{

       /* Initializing Variables */
       double Delta, gamma;
       double ****u0, ****U, ****U_int, ***u, ***v, ***w, ***artx, ***arty, ***artz;
       double *Instruct;
       int size_inst, T, T_num, X, Y, Z, brk, a, n;
       int i, j, k, r;
       double eps = 20.0; //artificial viscosity constant ~ 1
       char buffer[5];
       char outfile[20];
       char fileextension[] = ".dat";
       /* Reading Initial Conditions */
       Instruct = File2Array("data/Instruct.dat", &size_inst);
       Delta         = Instruct[0];                    // Step Size
       T                   = (int)Instruct[1];         // Number of iterations
       T_num         = (int)Instruct[2];         // Number of iterations outputed
```

```
   X                    = (int)Instruct[3];               // Simulation Dimensions
   Y                    = (int)Instruct[4];               // Simulation Dimensions
   Z                    = (int)Instruct[5];               // Simulation Dimensions

   gamma         = Instruct[6];                     // ratio of specific heat

   u0 = File2Array("data/u0.dat", X, Y, Z, 8);  // Extracting Initial Fields



   /* Allocating Memory for calculations */

   U = AllocateArray(8, X, Y, Z);
   U_int = AllocateArray(8, X, Y, Z);
   u = AllocateArray(X, Y, Z);
   v = AllocateArray(X, Y, Z);
   w = AllocateArray(X, Y, Z);
   artx = AllocateArray(X, Y, Z);
   arty = AllocateArray(X, Y, Z);
   artz = AllocateArray(X, Y, Z);

   /* Initialize Data. Converting input into Conservation Variables */

   for (i = 0; i < X; i++)
          for (j = 0; j < Y; j++)
                 for (k = 0; k < Z; k++) {
                        U[0][i][j][k] = u0[0][i][j][k];
                        u[i][j][k] = u0[1][i][j][k];
                        v[i][j][k] = u0[2][i][j][k];
                        w[i][j][k] = u0[3][i][j][k];
                        U[1][i][j][k] = U[0][i][j][k]*u[i][j][k];
                        U[2][i][j][k] = U[0][i][j][k]*v[i][j][k];
                        U[3][i][j][k] = U[0][i][j][k]*w[i][j][k];
                        /* Pressure and Magnetic Field kept primitive */
                        for (r = 4; r < 8; r++)
                               U[r][i][j][k] = u0[r][i][j][k];
                        artx[i][j][k] = 0;
                        arty[i][j][k] = 0;
                        artz[i][j][k] = 0;
                 }
   /* Simulating */
   a = 0;
n = 1;
brk = 0;
   while ((n <= T)&&(brk == 0))
{
     for (i = 0; i < 8; i++)
        for (j = 0; j < X; j++)
            for (k = 0; k < Y; k++)
                for (r = 0; r < Z; r++)
                {
                    //Checking whether the solution became NaN
                    if (U[i][j][k][r] != U[i][j][k][r])
                        brk = 1;
                    else
```

```
                        U_int[i][j][k][r] = U[i][j][k][r];
                    }
          //stops in case of NaN
          if (brk == 1)
              printf("Stopped on %d iteration\n",n);

/* Making Calculations: */

        //Forward Predictor
        Forw_Pred(Delta, X, Y, Z, gamma, eps, U, U_int, u, v, w, artx, arty, artz);

        //Backward Corrector
        Back_Corr(Delta, X, Y, Z, gamma, eps, U, U_int, u, v, w, artx, arty, artz);
        /* Boundary Condition: */

        for (r = 0; r < 5; r++)
            for (i = 0; i < X; i++)
                for (j = 0; j < Y; j++)
                   for (k = 1; k < 5; k++)
                        U[r][i][j][k] = U[r][i][j][0]; //Hydrodynamical Quantities should be
kept constant near the boundary

//where Magnetic Field is strong. Otherwise spurious acceleration is produced.

        Bound_Cond(X, Y, Z, U[0], 1);
        Bound_Cond(X, Y, Z, U[1], -1);
        Bound_Cond(X, Y, Z, U[2], -1);
        Bound_Cond(X, Y, Z, U[3], -1);
        Bound_Cond(X, Y, Z, U[4], 1);
        Bound_Cond(X, Y, Z, U[5], -1);
        Bound_Cond(X, Y, Z, U[6], -1);
        Bound_Cond(X, Y, Z, U[7], -1);

        //Prim_Calc(gamma, X, Y, Z, U, u, v, w);//Re-estimating Primitive Variables

              /* Generating Output */

        if ((n == ((int)((a+1)*T/T_num + 0.5)))||(brk == 1)) {
                    a++;
                    /* Creating a name for the output file: u(#number).dat */

                    memset(&outfile[0], 0, sizeof(outfile));
                    memset(&buffer[0], 0, sizeof(buffer));
                    sprintf(outfile, "data/u");
                    sprintf(buffer, "%d", a);
                    strcat(outfile, buffer);
                    strcat(outfile, fileextension);

                    /* Writing data to Disk */

                    printf("Writing output #%d of %d\n",a,T_num);
                    Array2File(U[0], outfile, X, Y, Z, "w");
                    Array2File(u, outfile, X, Y, Z, "a");
                    Array2File(v, outfile, X, Y, Z, "a");
                    Array2File(w, outfile, X, Y, Z, "a");
                    Array2File(U[4], outfile, X, Y, Z, "a");
                    Array2File(U[5], outfile, X, Y, Z, "a");
                    Array2File(U[6], outfile, X, Y, Z, "a");
```

```
                        printf("Done Writing output #%d of %d\n",a,T_num);
                        if (a == 1)
                                Var2File(Delta*n, "data/t.dat", "w");
                        else
                                Var2File(Delta*n, "data/t.dat", "a");

                }
                n++;
        }
        system("pause");
        return 0;
}

/* Program Routines */

double* File2Array(char* filename, int* num)
{
        /* This function reads the file "filename" and
                extracts the entries in the array and
                returns the number of entries in a file*/
        FILE *fr;
        double val, *parray;
        int i, size;
        /* open the file to read */
        fr = fopen(filename, "r");
        if (fr == NULL) {
                printf("File %s could not be opened for reading.\n",filename);
                exit(0);
        }
        /* establishing the number of entries */
        size = 0;
        while (fscanf(fr, "%lf", &val) == 1) {
                //printf("input: %e\n", val); // uncomment if you want to show on the screen the
results of loading
                size++;
        }
        *num = size;
        fseek(fr, 0, SEEK_SET);             // seek back to beginning of file
        /* allocating memory to store the contents */
        parray = (double*)malloc(size*sizeof(double));
        i = 0;
        while (fscanf(fr, "%lf", &val) == 1) {
                parray[i] = val;
                i++;
        }
        fclose(fr); //Closing the file

        return parray;
}

double**** File2Array(char* filename, int X, int Y, int Z, int dim)
{
        /* This function reads the file "filename" and
                extracts the entries in the array and
                returns the number of entries in a file*/
        FILE *fr;
        double val, ****parray;
        int i, j, k, r, size;
```

```
	/* open the file to read */
	fr = fopen(filename, "r");
	if (fr == NULL) {
		printf("File %s could not be opened for reading.\n",filename);
		exit(0);
	}
	/* establishing the number of entries */
	size = 0;
	while (fscanf(fr, "%lf", &val) == 1) {
		//printf("input: %e\n", val); // uncomment if you want to show on the screen the
results of loading
		size++;
	}
	if (size != (X*Y*Z*dim)) {
		printf("File %s contains too few or too much entries.\n",filename);
		exit(0);
	}
	fseek(fr, 0, SEEK_SET);		// seek back to beginning of file
	/* allocating memory to store the contents */
	parray = AllocateArray(dim, X, Y, Z);
	/* reading data */
	for (r = 0; r < dim; r++)
		for (k = 0; k < Z; k++)
			for (i = 0; i < X; i++)
				for (j = 0; j < Y; j++) {
					fscanf(fr, "%lf", &val);
					parray[r][i][j][k] = val;
				}
	fclose(fr); //Closing the file

	return parray;
}

void Array2File(double*** parray, char* filename, int X, int Y, int Z, const char* type)
{
	/* This function writes the file "filename" and
		extracts the entries from the 3-Dimensional array
		if type == "w" it overwrites the old cotents
		if type == "a" it appends the new row */
	FILE *fw;
	int i, j, k;
	fw = fopen(filename, type);
    if (fw == NULL) {
      printf("File %s could not be opened for writing.\n",filename);
      exit(0);
    }
	for (k = 0; k < Z; k++)
		for (i = 0; i < X; i++)
			for (j = 0; j < Y; j++)
				fprintf(fw, "  %.16e", parray[i][j][k]);
	fclose(fw); // Closing file for writing
}

double**** AllocateArray(int n1, int n2, int n3, int n4)
{
	/* This function allocates 2 dimensional array of pointers */
	int i, j, k;
	double**** parray;
```

```
	parray = (double****) malloc(n1*sizeof(double***));
	for (i = 0; i < n1; i++) {
		parray[i] = (double***) malloc(n2*sizeof(double**));
		for (j = 0; j < n2; j++) {
			parray[i][j] = (double**) malloc(n3*sizeof(double*));
			for (k = 0; k < n3; k++)
				parray[i][j][k] = (double*) malloc(n4*sizeof(double));
		}
	}
	return parray;
}


double*** AllocateArray(int n1, int n2, int n3)
{
	/* This function allocates 2 dimensional array of pointers */
	int i, j;
	double*** parray;
	parray = (double***) malloc(n1*sizeof(double**));
	for (i = 0; i < n1; i++) {
		parray[i] = (double**) malloc(n2*sizeof(double*));
		for (j = 0; j < n2; j++)
			parray[i][j] = (double*) malloc(n3*sizeof(double));
	}
	return parray;
}

void Var2File(double var, char* filename, const char* type)
{
	/* This function writes variable to the file "filename"
	if type == "w" it overwrites the old cotents
		if type == "a" it appends the new raw */
	FILE *fw;
	fw = fopen(filename, type);
    if (fw == NULL) {
      printf("File %s could not be opened for writing.\n",filename);
      exit(0);
    }
    fprintf(fw, "%e ", var);
	fclose(fw); // Closing file for writing
}

/* Computation Routines */

//FORWARD PREDICTOR
void Forw_Pred(double Delta, int X, int Y, int Z, double gamma, double eps,
	double ****U, double ****U_int, double ***u, double ***v,
	double ***w, double ***artx, double ***arty, double ***artz)
{
    //This procedure updates U_int, p, u and v on the basis of all
    //other inputs. It incorporates FORWARD PREDICTOR
    int j, k, r;
    //Artificial Viscosity
    for (j = 1; j < (X-1); j++)
        for (k = 1; k < (Y-1); k++)
            for (r = 1; r < (Z-1); r++)
            {
```

```
                artx[j][k][r] = -eps*(abs(u[j][k][r]) + sqrt(gamma*U[4][j][k][r]/U[0][j][k][r])
+ sqrt((U[5][j][k][r]*U[5][j][k][r] + U[6][j][k][r]*U[6][j][k][r] +
U[7][j][k][r]*U[7][j][k][r])/U[0][j][k][r]))* abs(U[4][j+1][k][r] - 2*U[4][j][k][r] + U[4][j-
1][k][r])/(U[4][j+1][k][r] + 2*U[4][j][k][r] + U[4][j-1][k][r]);
                arty[j][k][r] = -eps*(abs(v[j][k][r]) + sqrt(gamma*U[4][j][k][r]/U[0][j][k][r])
+ sqrt((U[5][j][k][r]*U[5][j][k][r] + U[6][j][k][r]*U[6][j][k][r] +
U[7][j][k][r]*U[7][j][k][r])/U[0][j][k][r]))* abs(U[4][j][k+1][r] - 2*U[4][j][k][r] +
U[4][j][k-1][r])/(U[4][j][k+1][r] + 2*U[4][j][k][r] + U[4][j][k-1][r]);
                artz[j][k][r] = -eps*(abs(w[j][k][r]) + sqrt(gamma*U[4][j][k][r]/U[0][j][k][r])
+ sqrt((U[5][j][k][r]*U[5][j][k][r] + U[6][j][k][r]*U[6][j][k][r] +
U[7][j][k][r]*U[7][j][k][r])/U[0][j][k][r]))* abs(U[4][j][k][r+1] - 2*U[4][j][k][r] +
U[4][j][k][r-1])/(U[4][j][k][r+1] + 2*U[4][j][k][r] + U[4][j][k][r-1]);
            }


    //Step:
    for (j = 1; j < (X-1); j++)
        for (k = 1; k < (Y-1); k++)
            for (r = 1; r < (Z-1); r++)
            {
                //Mass Density Equation
                U_int[0][j][k][r] = U[0][j][k][r] -
                        Delta*( U[1][j+1][k][r] - U[1][j][k][r] + U[2][j][k+1][r] -
U[2][j][k][r] + U[3][j][k][r+1] - U[3][j][k][r] +
                        //artificial viscosity:
                        artx[j+1][k][r]*(U[0][j+1][k][r] - U[0][j][k][r]) -
artx[j][k][r]*(U[0][j][k][r] - U[0][j-1][k][r]) +
                        arty[j][k+1][r]*(U[0][j][k+1][r] - U[0][j][k][r]) -
arty[j][k][r]*(U[0][j][k][r] - U[0][j][k-1][r]) +
                        artz[j][k][r+1]*(U[0][j][k][r+1] - U[0][j][k][r]) -
artz[j][k][r]*(U[0][j][k][r] - U[0][j][k][r-1]) );

                //Momentum Density Equation - x
                U_int[1][j][k][r] = U[1][j][k][r] -
                        Delta*( //x - derivative:
                        U[1][j+1][k][r]*u[j+1][k][r] - U[1][j][k][r]*u[j][k][r] +
                                        U[4][j+1][k][r] - U[4][j][k][r] +
                                        (U[6][j+1][k][r]*U[6][j+1][k][r] -
U[6][j][k][r]*U[6][j][k][r] +
                        U[7][j+1][k][r]*U[7][j+1][k][r] - U[7][j][k][r]*U[7][j][k][r] -
                        U[5][j+1][k][r]*U[5][j+1][k][r] + U[5][j][k][r]*U[5][j][k][r])/2  +
                        //y - derivative:
                        U[1][j][k+1][r]*v[j][k+1][r] - U[1][j][k][r]*v[j][k][r] -
                        U[5][j][k+1][r]*U[6][j][k+1][r] + U[5][j][k][r]*U[6][j][k][r] +
                        //z - derivative:
                        U[1][j][k][r+1]*w[j][k][r+1] - U[1][j][k][r]*w[j][k][r] -
                        U[5][j][k][r+1]*U[7][j][k][r+1] + U[5][j][k][r]*U[7][j][k][r] +
                        //artificial viscosity:
                        artx[j+1][k][r]*(U[1][j+1][k][r] - U[1][j][k][r]) -
artx[j][k][r]*(U[1][j][k][r] - U[1][j-1][k][r]) +
                        arty[j][k+1][r]*(U[1][j][k+1][r] - U[1][j][k][r]) -
arty[j][k][r]*(U[1][j][k][r] - U[1][j][k-1][r]) +
                        artz[j][k][r+1]*(U[1][j][k][r+1] - U[1][j][k][r]) -
artz[j][k][r]*(U[1][j][k][r] - U[1][j][k][r-1]));
                //Momentum Density Equation - y
                U_int[2][j][k][r] = U[2][j][k][r] -
                        Delta*( //x - derivative:
                        U[2][j+1][k][r]*u[j+1][k][r] - U[2][j][k][r]*u[j][k][r] -
```

```
                        U[6][j+1][k][r]*U[5][j+1][k][r] + U[6][j][k][r]*U[5][j][k][r]  +
                        //y - derivative:
                        U[4][j][k+1][r] - U[4][j][k][r] +
                        U[2][j][k+1][r]*v[j][k+1][r] - U[2][j][k][r]*v[j][k][r] +
                        (U[5][j][k+1][r]*U[5][j][k+1][r] - U[5][j][k][r]*U[5][j][k][r] +
                        U[7][j][k+1][r]*U[7][j][k+1][r] - U[7][j][k][r]*U[7][j][k][r] -
                        U[6][j][k+1][r]*U[6][j][k+1][r] + U[6][j][k][r]*U[6][j][k][r])/2 +
                        //z - derivative:
                        U[2][j][k][r+1]*w[j][k][r+1] - U[2][j][k][r]*w[j][k][r] -
                        U[6][j][k][r+1]*U[7][j][k][r+1] + U[6][j][k][r]*U[7][j][k][r]  +
                        //artificial viscosity:
                        artx[j+1][k][r]*(U[2][j+1][k][r] - U[2][j][k][r]) -
artx[j][k][r]*(U[2][j][k][r] - U[2][j-1][k][r]) +
                        arty[j][k+1][r]*(U[2][j][k+1][r] - U[2][j][k][r]) -
arty[j][k][r]*(U[2][j][k][r] - U[2][j][k-1][r]) +
                        artz[j][k][r+1]*(U[2][j][k][r+1] - U[2][j][k][r]) -
artz[j][k][r]*(U[2][j][k][r] - U[2][j][k][r-1]));
                //Momentum Density Equation - z
                U_int[3][j][k][r] = U[3][j][k][r] -
                        Delta*( //x - derivative:
                        U[3][j+1][k][r]*u[j+1][k][r] - U[3][j][k][r]*u[j][k][r] -
                        U[7][j+1][k][r]*U[5][j+1][k][r] + U[7][j][k][r]*U[5][j][k][r] +
                        //y - derivative:
                        U[3][j][k+1][r]*v[j][k+1][r] - U[3][j][k][r]*v[j][k][r] -
                        U[7][j][k+1][r]*U[6][j][k+1][r] + U[7][j][k][r]*U[6][j][k][r] +
                        //z - derivative:
                        U[3][j][k][r+1]*w[j][k][r+1] - U[3][j][k][r]*w[j][k][r] +
                        U[4][j][k][r+1] - U[4][j][k][r] +
                        (U[5][j][k][r+1]*U[5][j][k][r+1] - U[5][j][k][r]*U[5][j][k][r] +
                        U[6][j][k][r+1]*U[6][j][k][r+1] - U[6][j][k][r]*U[6][j][k][r] -
                        U[7][j][k][r+1]*U[7][j][k][r+1] + U[7][j][k][r]*U[7][j][k][r])/2  +
                        //artificial viscosity:
                        artx[j+1][k][r]*(U[3][j+1][k][r] - U[3][j][k][r]) -
artx[j][k][r]*(U[3][j][k][r] - U[3][j-1][k][r]) +
                        arty[j][k+1][r]*(U[3][j][k+1][r] - U[3][j][k][r]) -
arty[j][k][r]*(U[3][j][k][r] - U[3][j][k-1][r]) +
                        artz[j][k][r+1]*(U[3][j][k][r+1] - U[3][j][k][r]) -
artz[j][k][r]*(U[3][j][k][r] - U[3][j][k][r-1]));
                //Pressure Density Equation
                U_int[4][j][k][r] = U[4][j][k][r] -
                        Delta*( U[4][j+1][k][r]*u[j+1][k][r] - U[4][j][k][r]*u[j][k][r] +
                        U[4][j][k+1][r]*v[j][k+1][r] - U[4][j][k][r]*v[j][k][r] +
                        U[4][j][k][r+1]*w[j][k][r+1] - U[4][j][k][r]*w[j][k][r] +
                                        (gamma - 1)*U[4][j][k][r]*(u[j+1][k][r] - u[j][k][r] +
v[j][k+1][r] - v[j][k][r] + w[j][k][r+1] - w[j][k][r]) +
                        //artificial viscosity:
                        artx[j+1][k][r]*(U[4][j+1][k][r] - U[4][j][k][r]) -
artx[j][k][r]*(U[4][j][k][r] - U[4][j-1][k][r]) +
                        arty[j][k+1][r]*(U[4][j][k+1][r] - U[4][j][k][r]) -
arty[j][k][r]*(U[4][j][k][r] - U[4][j][k-1][r]) +
                        artz[j][k][r+1]*(U[4][j][k][r+1] - U[4][j][k][r]) -
artz[j][k][r]*(U[4][j][k][r] - U[4][j][k][r-1]));
                //Magnetic Field - x
                U_int[5][j][k][r] = U[5][j][k][r] -
                        Delta*(  //y - derivative:
                        v[j][k+1][r]*U[5][j][k+1][r] - v[j][k][r]*U[5][j][k][r] -
                        u[j][k+1][r]*U[6][j][k+1][r] + u[j][k][r]*U[6][j][k][r] +
                        //z - derivative:
```

```
                        w[j][k][r+1]*U[5][j][k][r+1] - w[j][k][r]*U[5][j][k][r] -
                        u[j][k][r+1]*U[7][j][k][r+1] + u[j][k][r]*U[7][j][k][r] );
                //Magnetic Field - y
                U_int[6][j][k][r] = U[6][j][k][r] -
                        Delta*( //x - derivative:
                        u[j+1][k][r]*U[6][j+1][k][r] - u[j][k][r]*U[6][j][k][r] -
                        v[j+1][k][r]*U[5][j+1][k][r] + v[j][k][r]*U[5][j][k][r] +
                        //z - derivative:
                        w[j][k][r+1]*U[6][j][k][r+1] - w[j][k][r]*U[6][j][k][r] -
                        v[j][k][r+1]*U[7][j][k][r+1] + v[j][k][r]*U[7][j][k][r] );
                //Magnetic Field - z
                U_int[7][j][k][r] = U[7][j][k][r] -
                        Delta*(  //x - derivative:
                        u[j+1][k][r]*U[7][j+1][k][r] - u[j][k][r]*U[7][j][k][r] -
                        w[j+1][k][r]*U[5][j+1][k][r] + w[j][k][r]*U[5][j][k][r] +
                        //y - derivative:
                        v[j][k+1][r]*U[7][j][k+1][r] - v[j][k][r]*U[7][j][k][r] -
                        w[j][k+1][r]*U[6][j][k+1][r] + w[j][k][r]*U[6][j][k][r] );
            }
        //Calculating Velocity and pressure
        Prim_Calc(gamma, X, Y, Z, U_int, u, v, w);
}
//BACKWARD CORRECTOR
void Back_Corr(double Delta, int X, int Y, int Z, double gamma, double eps,
        double ****U, double ****U_int, double ***u, double ***v,
        double ***w, double ***artx, double ***arty, double ***artz)
{
    //This procedure updates U, p, u and v on the basis of all
    //other inputs. It incorporates BACKWARD CORRECTOR
    int j, k, r;
    //Artificial viscosity
    for (j = 1; j < (X-1); j++)
        for (k = 1; k < (Y-1); k++)
            for (r = 1; r < (Z-1); r++)
            {
                artx[j][k][r] = -eps*(abs(u[j][k][r]) +
sqrt(gamma*U_int[4][j][k][r]/U_int[0][j][k][r]) + sqrt((U_int[5][j][k][r]*U_int[5][j][k][r] +
U_int[6][j][k][r]*U_int[6][j][k][r] + U_int[7][j][k][r]*
U_int[7][j][k][r])/U_int[0][j][k][r]))*abs(U_int[4][j+1][k][r] - 2*U_int[4][j][k][r] +
U_int[4][j-1][k][r])/ (U_int[4][j+1][k][r] + 2*U_int[4][j][k][r] + U_int[4][j-1][k][r]);
                arty[j][k][r] = -eps*(abs(v[j][k][r]) +
sqrt(gamma*U_int[4][j][k][r]/U_int[0][j][k][r]) + sqrt((U_int[5][j][k][r]*U_int[5][j][k][r] +
U_int[6][j][k][r]*U_int[6][j][k][r] + U_int[7][j][k][r]*
U_int[7][j][k][r])/U_int[0][j][k][r]))*abs(U_int[4][j][k+1][r] -  2*U_int[4][j][k][r] +
U_int[4][j][k-1][r])/(U_int[4][j][k+1][r] + 2*U_int[4][j][k][r] + U_int[4][j][k-1][r]);
                artz[j][k][r] = -eps*(abs(w[j][k][r]) +
sqrt(gamma*U_int[4][j][k][r]/U_int[0][j][k][r]) + sqrt((U_int[5][j][k][r]*U_int[5][j][k][r] +
U_int[6][j][k][r]*U_int[6][j][k][r] + U_int[7][j][k][r]*
U_int[7][j][k][r])/U_int[0][j][k][r]))*abs(U_int[4][j][k][r+1] -  2*U_int[4][j][k][r] +
U_int[4][j][k][r-1])/(U_int[4][j][k][r+1] + 2*U_int[4][j][k][r] + U_int[4][j][k][r-1]);
            }
    for (j = 1; j < (X-1); j++)
        for (k = 1; k < (Y-1); k++)
            for (r = 1; r < (Z-1); r++)
            {
                //Mass Density Equation
                U[0][j][k][r] = (U[0][j][k][r] + U_int[0][j][k][r] -
```

```
                        Delta*( U_int[1][j][k][r] - U_int[1][j-1][k][r] + U_int[2][j][k][r] -
U_int[2][j][k-1][r] + U_int[3][j][k][r] - U_int[3][j][k][r-1] +
                        //artificial viscosity:
                        artx[j][k][r]*(U_int[0][j+1][k][r] - U_int[0][j][k][r]) - artx[j-
1][k][r]*(U_int[0][j][k][r] - U_int[0][j-1][k][r]) +
                        arty[j][k][r]*(U_int[0][j][k+1][r] - U_int[0][j][k][r]) - arty[j][k-
1][r]*(U_int[0][j][k][r] - U_int[0][j][k-1][r]) +
                        artz[j][k][r]*(U_int[0][j][k][r+1] - U_int[0][j][k][r]) - artz[j][k][r-
1]*(U_int[0][j][k][r] - U_int[0][j][k][r-1])
                        ))/2;
                //Momentum Density Equation - x
                U[1][j][k][r] = (U[1][j][k][r] + U_int[1][j][k][r] -
                        Delta*( //x - derivative:
                        U_int[1][j][k][r]*u[j][k][r] - U_int[1][j-1][k][r]*u[j-1][k][r] +
                        U_int[4][j][k][r] - U_int[4][j-1][k][r] +
                        (U_int[6][j][k][r]*U_int[6][j][k][r] - U_int[6][j-1][k][r]*U_int[6][j-
1][k][r] + U_int[7][j][k][r]*U_int[7][j][k][r] - U_int[7][j-1][k][r]*U_int[7][j-1][k][r] -
                        U_int[5][j][k][r]*U_int[5][j][k][r] + U_int[5][j-1][k][r]*U_int[5][j-
1][k][r])/2 +
                        //y - derivative:
                        U_int[1][j][k][r]*v[j][k][r] - U_int[1][j][k-1][r]*v[j][k-1][r]  -
                        U_int[5][j][k][r]*U_int[6][j][k][r] + U_int[5][j][k-
1][r]*U_int[6][j][k-1][r] +
                        // z -derivative:
                        U_int[1][j][k][r]*w[j][k][r] - U_int[1][j][k][r-1]*w[j][k][r-1]  -
                        U_int[5][j][k][r]*U_int[7][j][k][r] + U_int[5][j][k][r-
1]*U_int[7][j][k][r-1] +
                        //artificial viscosity:
                        artx[j][k][r]*(U_int[1][j+1][k][r] - U_int[1][j][k][r]) - artx[j-
1][k][r]*(U_int[1][j][k][r] - U_int[1][j-1][k][r]) +
                        arty[j][k][r]*(U_int[1][j][k+1][r] - U_int[1][j][k][r]) - arty[j][k-
1][r]*(U_int[1][j][k][r] - U_int[1][j][k-1][r]) +
                        artz[j][k][r]*(U_int[1][j][k][r+1] - U_int[1][j][k][r]) - artz[j][k][r-
1]*(U_int[1][j][k][r] - U_int[1][j][k][r-1])
                        ))/2;
                //Momentum Density Equation - y
                U[2][j][k][r] = (U[2][j][k][r] + U_int[2][j][k][r] -
                        Delta*( //x - derivative:
                        U_int[2][j][k][r]*u[j][k][r] - U_int[2][j-1][k][r]*u[j-1][k][r] -
                        U_int[6][j][k][r]*U_int[5][j][k][r] + U_int[6][j-1][k][r]*U_int[5][j-
1][k][r] +
                        //y - derivative:
                        U_int[4][j][k][r] - U_int[4][j][k-1][r] +
                        U_int[2][j][k][r]*v[j][k][r] - U_int[2][j][k-1][r]*v[j][k-1][r] +
                        (U_int[5][j][k][r]*U_int[5][j][k][r] - U_int[5][j][k-
1][r]*U_int[5][j][k-1][r] +
                        U_int[7][j][k][r]*U_int[7][j][k][r] - U_int[7][j][k-
1][r]*U_int[7][j][k-1][r] -
                        U_int[6][j][k][r]*U_int[6][j][k][r] + U_int[6][j][k-
1][r]*U_int[6][j][k-1][r])/2 +
                        //z - derivative:
                        U_int[2][j][k][r]*w[j][k][r] - U_int[2][j][k][r-1]*w[j][k][r-1] -
                        U_int[6][j][k][r]*U_int[7][j][k][r] + U_int[6][j][k][r-
1]*U_int[7][j][k][r-1] +
                        //artificial viscosity:
                        artx[j][k][r]*(U_int[2][j+1][k][r] - U_int[2][j][k][r]) - artx[j-
1][k][r]*(U_int[2][j][k][r] - U_int[2][j-1][k][r]) +
```

```
                        arty[j][k][r]*(U_int[2][j][k+1][r] - U_int[2][j][k][r]) - arty[j][k-
1][r]*(U_int[2][j][k][r] - U_int[2][j][k-1][r]) +
                        artz[j][k][r]*(U_int[2][j][k][r+1] - U_int[2][j][k][r]) - artz[j][k][r-
1]*(U_int[2][j][k][r] - U_int[2][j][k][r-1])
                        ))/2;
                //Momentum Density Equation - z
                U[3][j][k][r] = (U[3][j][k][r] + U_int[3][j][k][r] -
                        Delta*( //x - derivative:
                        U_int[3][j][k][r]*u[j][k][r] - U_int[3][j-1][k][r]*u[j-1][k][r] -
                        U_int[7][j][k][r]*U_int[5][j][k][r] + U_int[7][j-1][k][r]*U_int[5][j-
1][k][r] +
                        //y - derivative:
                        U_int[3][j][k][r]*v[j][k][r] - U_int[3][j][k-1][r]*v[j][k-1][r] -
                        U_int[7][j][k][r]*U_int[6][j][k][r] + U_int[7][j][k-
1][r]*U_int[6][j][k-1][r] +
                        //z - derivative:
                        U_int[4][j][k][r] - U_int[4][j][k][r-1] +
                        U_int[3][j][k][r]*w[j][k][r] - U_int[3][j][k][r-1]*w[j][k][r-1] +
                        (U_int[5][j][k][r]*U_int[5][j][k][r] - U_int[5][j][k][r-
1]*U_int[5][j][k][r-1] +
                        U_int[6][j][k][r]*U_int[6][j][k][r] - U_int[6][j][k][r-
1]*U_int[6][j][k][r-1] -
                        U_int[7][j][k][r]*U_int[7][j][k][r] + U_int[7][j][k][r-
1]*U_int[7][j][k][r-1])/2 +
                        //artificial viscosity:
                        artx[j][k][r]*(U_int[3][j+1][k][r] - U_int[3][j][k][r]) - artx[j-
1][k][r]*(U_int[3][j][k][r] - U_int[3][j-1][k][r]) +
                        arty[j][k][r]*(U_int[3][j][k+1][r] - U_int[3][j][k][r]) - arty[j][k-
1][r]*(U_int[3][j][k][r] - U_int[3][j][k-1][r]) +
                        artz[j][k][r]*(U_int[3][j][k][r+1] - U_int[3][j][k][r]) - artz[j][k][r-
1]*(U_int[3][j][k][r] - U_int[3][j][k][r-1])
                        ))/2;
                //Pressure Density Equation
                 U[4][j][k][r] = (U[4][j][k][r] + U_int[4][j][k][r] -
                        Delta*( U_int[4][j][k][r]*u[j][k][r] - U_int[4][j-1][k][r]*u[j-1][k][r]
+
                        U_int[4][j][k][r]*v[j][k][r] - U_int[4][j][k-1][r]*v[j][k-1][r] +
                        U_int[4][j][k][r]*w[j][k][r] - U_int[4][j][k][r-1]*w[j][k][r-1] +
                        (gamma - 1)*U_int[4][j][k][r]*(u[j][k][r] - u[j-1][k][r] + v[j][k][r] -
v[j][k-1][r] + w[j][k][r] - w[j][k][r-1]) +
                        //artificial viscosity:
                        artx[j][k][r]*(U_int[4][j+1][k][r] - U_int[4][j][k][r]) - artx[j-
1][k][r]*(U_int[4][j][k][r] - U_int[4][j-1][k][r]) +
                        arty[j][k][r]*(U_int[4][j][k+1][r] - U_int[4][j][k][r]) - arty[j][k-
1][r]*(U_int[4][j][k][r] - U_int[4][j][k-1][r]) +
                        artz[j][k][r]*(U_int[4][j][k][r+1] - U_int[4][j][k][r]) - artz[j][k][r-
1]*(U_int[4][j][k][r] - U_int[4][j][k][r-1])
                        ))/2;
                //Magnetic Field - x
                U[5][j][k][r] = (U[5][j][k][r] + U_int[5][j][k][r] -
                        Delta*(  //y - derivative:
                        v[j][k][r]*U_int[5][j][k][r] - v[j][k-1][r]*U_int[5][j][k-1][r] -
                        u[j][k][r]*U_int[6][j][k][r] + u[j][k-1][r]*U_int[6][j][k-1][r] +
                        //z - derivative:
                        w[j][k][r]*U_int[5][j][k][r] - w[j][k][r-1]*U_int[5][j][k][r-1] -
                        u[j][k][r]*U_int[7][j][k][r] + u[j][k][r-1]*U_int[7][j][k][r-1]
                        ))/2;
                //Magnetic Field - y
```

```
                U[6][j][k][r] = (U[6][j][k][r] + U_int[6][j][k][r] -
                        Delta*( //x - derivative:
                        u[j][k][r]*U_int[6][j][k][r] - u[j-1][k][r]*U_int[6][j-1][k][r] -
                        v[j][k][r]*U_int[5][j][k][r] + v[j-1][k][r]*U_int[5][j-1][k][r] +
                        //z - derivative:
                        w[j][k][r]*U_int[6][j][k][r] - w[j][k][r-1]*U_int[6][j][k][r-1] -
                        v[j][k][r]*U_int[7][j][k][r] + v[j][k][r-1]*U_int[7][j][k][r-1]
                        ))/2;
                //Magnetic Field - z
                U[7][j][k][r] = (U[7][j][k][r] + U_int[7][j][k][r] -
                        Delta*( //x - derivative:
                        u[j][k][r]*U_int[7][j][k][r] - u[j-1][k][r]*U_int[7][j-1][k][r] -
                        w[j][k][r]*U_int[5][j][k][r] + w[j-1][k][r]*U_int[5][j-1][k][r] +
                        //y - derivative:
                        v[j][k][r]*U_int[7][j][k][r] - v[j][k-1][r]*U_int[7][j][k-1][r] -
                        w[j][k][r]*U_int[6][j][k][r] + w[j][k-1][r]*U_int[6][j][k-1][r]
                        ))/2;
            }
        //Calculating Velocity and pressure
        Prim_Calc(gamma, X, Y, Z, U, u, v, w);
}

void Prim_Calc(double gamma, int X, int Y, int Z, double ****U, double ***u,
        double ***v, double ***w)
{
    int j, k, r;
    //Calculating Velocity and pressure
        for (j = 0; j < X; j++)
            for (k = 0; k < Y; k++)
                for (r = 0; r < Z; r++)
                {
                    if (U[0][j][k][r] <= 0)
                    {
                        printf("Density Negative or Zero!\n");
                        printf("x = %d; y = %d; z = %d!\n",j,k,r);
                                    return;
                    }
                    u[j][k][r] = U[1][j][k][r]/U[0][j][k][r];
                    v[j][k][r] = U[2][j][k][r]/U[0][j][k][r];
                    w[j][k][r] = U[3][j][k][r]/U[0][j][k][r];
                }
}

void Bound_Cond(int X, int Y, int Z, double ***U, int sign)
{
    int j, k, r;

    for (j = 0; j < X; j++)
        for (k = 0; k < Y; k++)
        {
            U[j][k][Z-1] = sign*U[j][k][Z-2];
        }
    for (k = 0; k < Y; k++)
        for (r = 0; r < Z; r++)
        {
            U[0][k][r] = sign*U[1][k][r];
            U[X-1][k][r] = sign*U[X-2][k][r];
        }
```

```
    for (j = 0; j < X; j++)
        for (r = 0; r < Z; r++)
        {
            U[j][0][r] = sign*U[j][1][r];
            U[j][Y-1][r] = sign*U[j][Y-2][r];
        }
}
```