MASTER THESIS
ADDING A TRACER FLUID TO THE JUPITER CODE

Cheryl Luessi
Supervisor: Prof. Dr. J. Szulágyi
ETH Zurich

April 29, 2022

Abstract

The aim of this work is to add a tracer fluid to the JUPITER code, written by F. Masset [1] and J. Szulágyi [2], who added the adiabatic treatment of the fluid by including the radiative equation, in order to study the interaction and flow between the circumstellar disk (CSD) and the circumplanetary disk (CPD) or circumplanetary envelope (CPE). This code solves the conservation equation for mass, angular momentum and energy for a fluid, as well as the radiative equation to account for heating and cooling mechanisms. We add the tracer fluid by solving only for the equation of advection, since the tracers do not exert any force on the gas fluid.
The simulations show that the during the formation of the planet, the gas on the outer side of the planet flows towards larger radii near the equatorial plane and towards larger colatitude positions (we simulate the upper hemisphere), which agrees with the expectations from our current understanding of the meridional circulation, describing the gas flow between the CSD and the CPE or CPD. Furthermore, the forming spiral wakes push the gas fluid away from the planet.

Contents

1 INTRODUCTION TO PLANET FORMATION DONE

In this Section, we start with a brief overview of the historical evolution of astrophysical observations in Section 1.1 and continue with a short summary on star formation in Section 1.2. In Section 1.3, the various mechanisms that lead to the formation of planets are listed. We focus on the formation of gaseous giant planets, see Section 1.3.2, discussing the disk-instability mechanism as well as the core nucleated accretion process, but for completeness, we describe the formation of solid planets as well, see Section 1.3.1. We list the main aspects of planet formation and the phenomena and physics in the CSD. To do so, we first introduce the governing equations of fluid motion and describe the overall particle flow in the CSD, see 1.5. A more detailed discussion describing the meridional circulation is provided in Section 1.6 as this part is closely related to the goal of this work. Finally, we take a closer look at the formation of gaps in the CSD around a planet and their profile in Section 1.7. By the end of this Section, all relevant processes in the CSD are introduced and the fluid motion in the planet's vicinity is described, such that we can introduce the methods to track the gas flow with the tracer particles. This is done in Section 3.

1.1 Historical Background DONE

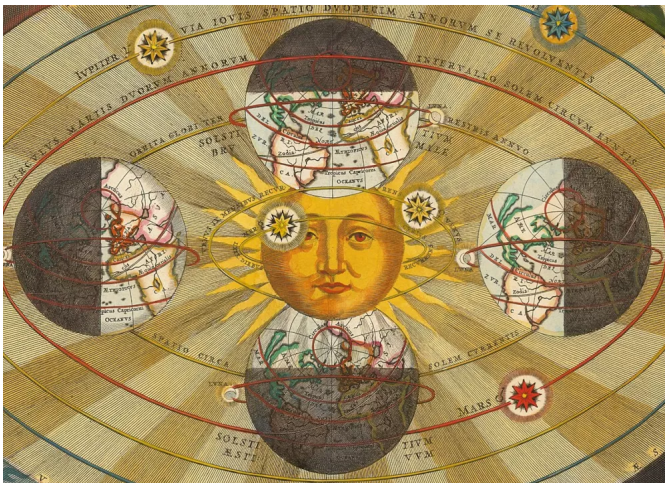

Figure 1: Illustration of the heliocentric model, as it was developed over centuries and accepted from the 18th century on, see [3].

The observation of planets, Sun, Moon and other astrophysical objects in the Earth's vicinity has been the topic of many studies for countless centuries and reaches back to ancient Greek philosophers and physicists such as Aristarchus of Samos (c. 310 – c. 230 BC), Philolaus (c. 470 – c. 385 BCE) or Hicetas (c. 400 – c. 335 BC) and even further. With ancient telescopes (see figure 2), astronomers observed the night sky, detecting the planets that orbit the Sun in our Solar System. For centuries however, this concept of a Solar System, as it is currently accepted with the Sun in the centre of mass being orbited by solid planets at smaller radii and gaseous planets at larger radii, was not known. Only in 1704, see [4], this term is recorded for the first time. From then on, the heliocentric model as illustrated in figure 1 was more and more accepted, replacing the geocentric model, in which the Earth is situated at the centre of the Universe and the Sun, Moon and planets revolve around it. In the succeeding 18th century, the formation of the Solar System has been discussed by various philosophers, the most prominent names are Immanuel Kant, figure 3 (left), in his *Universal Natural History and Theory of the Heavens*, published in 1755, see [5], and the french scholar Pierre-Simon Laplace, figure 3 (right), in his *The System of the World*, published in 1809, see [6].

With the improvement in both physics and engineering over the following centuries, larger (ground based first and later space based) telescopes, refractors and reflectors, were built, such as the Arecibo Observatory 4 (left) in Puerto Rico or the VLT 4 (right) in the Atacama Desert in Chile to name only two examples.
The Arebico Observatory was a spherically shaped reflector of 305m in diameter, which was built into a sinkhole due to its high mass and operated in the radio-wavelength regime. For many years, is was the largest ground based telescope. It has been demolished in 2020 after a part of the instrument fell on the disk and destroyed it. The VLT

Figure 2: Illustration of Johannes Hevelius, observing with one of his telescopes in 1647, [7].

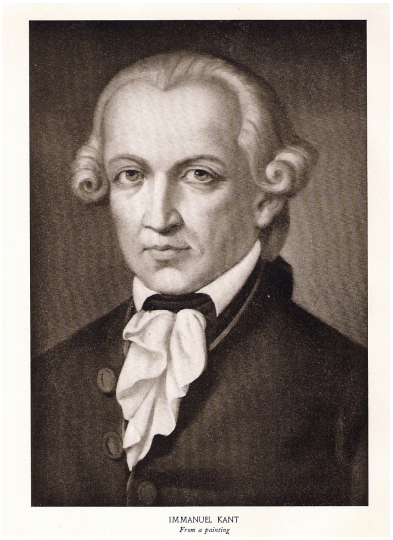

Figure 3: Left: Engraving of the German philosopher Immanuel Kant, 1724-1804, from [8]. Right: Illustration of the French scholar Simon-Pierre Laplace, 1749-1827, from [9]. Both philosophers discussed the Solar System and how it could have formed.

consists of four individual telescopes, which have primary mirrors of 8.2m diameter, and detect light in the visible and infra-red wavelengths. In order to avoid image blurring due to the Earth's atmosphere, space telescopes have been developed. Probably the most famous one is the Hubble Space Telescope, figure 5 (HST, left), which launched in 1990 and has since then made over 1.5 million observations[1], or the just recently launched[2] James Webb Space Telescope, figure 5 (JWST, right), as the successor of the HST. With telescopes such as these, astrophysicists can observe the Universe, its structures and their evolution at larger distances, greater details at higher redshifts.

However, the theoretical part in astronomy is as relevant as collecting and analysing data, not only for the early Universe at redshifts higher than the currently possible limit. Understanding how structures form, grow and interact with each other can be studied by simulating the most relevant objects, forces and interactions. For this Master Thesis and the code we work with, we have to introduce these parts, which we do in the following sections.

[1] A collection of most astonishing images can be found in [12].
[2] The JWST launched on 25th Dec. 2022.

Figure 4: Left: Arecibo Observatory in Puerto Rico, from [10]. It was the largest radio telescope for many years. Right: the Very Large Telescope, VLT, situated in the Atacama Desert in Chile, from [11]. It detects wavelengths in the visible and infrared regime.

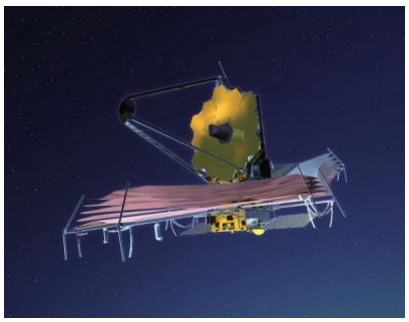

Figure 5: Left: Hubble Space Telescope, from [13]. It launched in 1990 and has since then made more than 1.5 million observations. Right: James Webb Space Telescope, from [14]. It launched on 25th Dec. 2021 as the successor of the HST. Due to its large system of mirrors, light in several wavelength regimes can be observed up to infra-red wavelengths.

1.2 Star Formation DONE

This Section is based on [15] and [16].
Most stars in the Universe form within gas-rich galaxies - typically spiral galaxies that contain large amounts of interstellar matter (ISM) - in dense molecular clouds. They compose mainly of H_2 and CO molecules at a temperature of around 30 K. They are roughly classified into three groups depending on their size, from small Bok globules ($m =\leq 100 M_{Sun}$, $r \leq 0.1 pc$) to molecular clouds ($m = 10^3 - 10^4 M_{Sun}$, $r \leq 10 pc$) to giant molecular clouds ($m = 10^5 - 10^7 M_{Sun}$, $r \leq 100 pc$). Due to gravitational instabilities, over-dense regions in such molecular clouds form and eventually collapse. These over-dense regions must satisfy the condition

$$r > r_J \simeq \frac{a_0}{(G\rho_0)^{1/2}}, \tag{1}$$

where G is the gravitational constant, ρ_0 the density of the molecular cloud and a_0 the isothermal speed of sound. This condition follows from the fact that a volume in the cloud can only collapse if its self-gravity overcomes the out-ward directed pressure of the gas (due to the molecules motion), i.e.

$$\frac{\nabla P}{\rho_0} < \frac{-GM}{r^2},$$

with mass $M \simeq r^3 \rho_0$. r_J is the so-called Jeans radius and defines the minimum radius of a volume to collapse under self-gravity. Note that this condition does not take into account other mechanisms that could increase the gas density in that volume, e.g. a supernova in the vicinity.
We can convert equation (1) to an equivalent statement,

$$M > M_J \simeq \frac{4\pi a_0^3}{3 \cdot (G^3 \rho_0)^{1/2}}, \tag{2}$$

with M_J the Jeans-mass. This description however assumes an almost homogeneous density distribution in the molecular cloud. A better estimation of the minimal mass of a cloud to collapse is given by the Virial theorem, which we derive in the Appendix 10.1:

Theorem 1 (Virial Theorem) *For a physical system in gravitational equilibrium*

$$V + 2T = 0, \tag{3}$$

where T and V denote the kinetic and potential energy, respectively. In an N-body system, we have

$$T = \sum_{i=1}^{N} T_i,$$

the sum of the kinetic energies of all members and

$$V = \frac{1}{2} \sum_{i \neq j} V_{i,j},$$

with $V_{i,j}$ the potential energy between particle i and j.

In case of a spherical collapse of mass M and radius R, we have

$$T = \frac{1}{2} M \sigma^2 \qquad V = \frac{-GM^2}{r}$$

with σ the velocity dispersion. Inserting these two expressions into the Virial Theorem (3), we find

$$\sigma^2 \simeq \frac{GM}{R},$$

or equivalent

$$M_{vir} \simeq \frac{\sigma^2 R}{G},$$

the Virial mass. If a cloud has mass $M = M_J$ it is stable and does not collapse. For $M > M_J$, the self-gravity exceeds the outwards pressure of the gas and the cloud collapses. On the other hand, a cloud with $M < M_J$ will eventually disperse, since it is not gravitationally bound.
Other mechanisms that favour the collapse of a molecular cloud are star formation feedback such as a supernova or stellar outflows which create over-pressurised regions in their surroundings, the fragmentation of the cloud, which explains the formation of multiple stars in one molecular cloud[3], or the ambipolar diffusion. Other processes, such as external heating due to stars or cosmic rays, which heat up the molecular clouds significantly, angular momentum conservation, magnetic fields or turbulence can stabilize the cloud from collapsing.
With the collapse of the molecular cloud, *Young Stellar Objects* (YSO) form. We classify these YSO according to their spectral energy distribution (SED) as shown in figure 6.

- *Class 0*: SED-peak in the far-IR or sub-mm range; no near-IR flux. This is the first stage of a YSO and describes a dense, pre-stellar core in a molecular cloud that has formed due to the previously mentioned collapsing mechanisms.

- *Class 1*: SED rises from $1\mu m$ towards longer wavelengths. Because the YSO is still in the process of collapsing, the temperature increases further but is surrounded by a gaseous shell, which absorbs most of the radiation from the YSO and re-emits it in the far-IR range.

- *Class 2*: SED falls towards mid-IR wavelengths and shows strong UV lines. These are for example T-Tauri stars. The UV-excess is due to the strong accretion of material onto the YSO. Due to angular momentum conservation, the gas shell has now formed a circumstellar disk.

[3]Depending on the size of the molecular cloud, a few stars up to hundreds of stars can form in one molecular cloud.

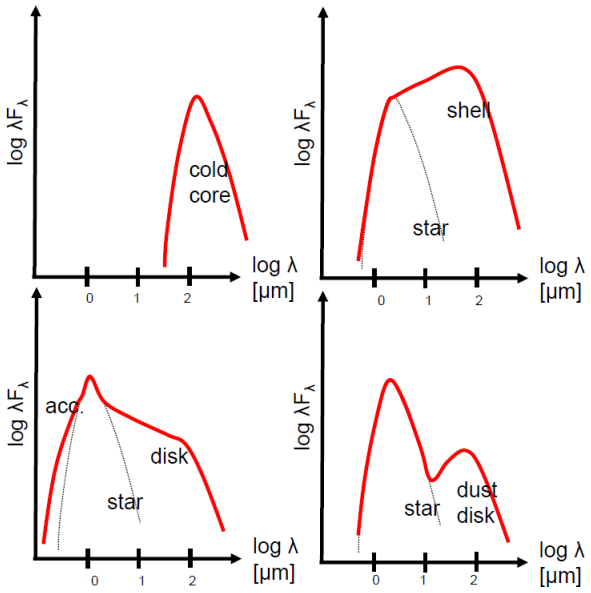

Figure 6: Classification of *Young Stellar Objects* according to their spectral energy distribution (SED), [16].

- *Class 3*: SED shows emission lines due to the ongoing gas accretion, however weaker than in the previous stage. These are pre-main sequence stars and lie in the HR-diagram[4] above the main sequence.

The code that is used for this work simulates such a circumstellar disk of a forming star.

1.3 Planet Formation DONE

This Section is dedicated to give e brief introduction on the planet formation in general, without going too much into details. We discuss various models and highlight the differences between the formation of solid planets and gaseous planets, starting with the former one.

1.3.1 The formation of solid planets DONE

Hereafter, we give a brief summary of the formation of terrestrial planets, a more thorough explanation can be found in many textbooks or papers on planet formation, for example [17] or [18].
The most accepted model for the formation of terrestrial planets is by coagulation of solids that are present in the CSD. In regions where the temperature falls below a certain threshold value, dust and other solid particles condensate out of the fluid and successively build up increasingly larger objects through aggregation[5]. These massive objects of various sizes collide, increasing their sizes further. With increasing mass, the gravitational influence of the so-called planetesimals on the surrounding objects becomes more significant and smaller particles are attracted. Further information on the formation of planetesimals is provided in e.g. [19] and [20]. In this context, we define the Hill-sphere of a planetesimal, which is the surrounding region where the gravitational attraction exerted by the planetesimal is larger than the star's gravitational force. We highlight that the feeding zone of a planetesimal is larger than the Hill-sphere, roughly by a factor of 2.3. Once all mass within the feeding zone has accreted onto the planetesimal, the planet reaches its so-called isolation mass. In the inner parts of the CSD, this mass corresponds to $\sim 0.01 - 0.1\ M_{Earth}$. The Hill sphere is defined by Hill-radius,

$$R_H = a(1-\varepsilon)\cdot\left(\frac{m}{3M}\right)^{1/3}, \tag{4}$$

[4]The Hertzsprung-Russell diagram (HR-diagram) shows the luminosity of a star as function of the temperature and is one of the main tools to show the evolution of a star or a cluster of stars.

[5]With increasing size, these objects are called *dust particles* (sub-μm to cm), *chrondules*, *chondrites*, *planetesimals* ($(100-1000)km$) and *planetary embryo* (few $10^3 km$; masses from $10^{22} kg$ to $10^{23} kg$)

where ε is the eccentricity of the smaller object's orbit with semi-major axis a, m the mass of that small object and M the mass of the central object.

Note, that the formation of terrestrial planets with Pluto- to super-Earth sizes depends on the collisional evolution of rings of solids[6] within the CSD. The evolution of these rings of solids in turn depends on various parameters, such as the efficiency of planetesimal formation f and the initial mass of the solids M_0, and different evolution histories are to be expected. A thorough study is provided in [21].

1.3.2 The formation of gaseous planets DONE

The following discussion is based on [22] and we briefly summarize the main statements regarding the two models of formation of a giant gas planet: via disk instability and via core nucleated accretion. The discussion of the formation of giant planets is key to understand the evolution of the whole planetary system, since they contain most mass in the CSD[7]. We briefly mention that the formation of giant planets usually does not take place in the innermost regions of the CSD for the following reasons: (see [23])

- 1) The temperature in the CSD at a radius of $\simeq 0.05AU$ from the central star is too high for solid particle, which are the main building block for the core, to condense out of the fluid.

- 2) The innermost region of the CSD usually does not contain enough mass for a giant gas planet to grow. This point can however be overcome by inward migration of other solid objects located at greater distances from the star. This migration can be caused by the normal (viscous) evolution of the CSD or the gas drag.

- 3) Tidal forces between the forming planet and the CSD would cause the planet to rapidly migrate towards the central star and eventually fall into it.

1.3.2.1 Formation by Disk-Instability DONE In a differentially rotating disks, gravitational instabilities may arise when the destabilization effect of gas self-gravity locally overcomes the stabilizing effect of gas rotation and pressure. In a thin Keplerian disk, this condition is given by

$$Q := \frac{c_g \Omega}{\pi G \Sigma_g} < Q_{crit},$$

where Q is the Toomre stability parameter, Q_{crit} the corresponding critical value, e.g. $Q_{crit} = 1$ in the case of instabilities driven by axisymmetric perturbations. Several studies have shown that a disk is gravitationally unstable to develop non-axisymmetric structures for an initial value $Q_i \in [1.5,\ 1.7]$, e.g. [24]. Because $c_g^2 \propto T_g$ with T_g the gas temperature, and $\Sigma_g \propto M_{loc}$, the local disk mass, the above condition implies that cold and massive regions of a disk are more susceptible to gravitational instabilities. If the condition is satisfied, a gravitational instability can grow on a time-scale τ, given by

$$\tau = \frac{2\pi}{\Omega}.$$

The evolution of gravitational instabilities in disks in predominantly defined by the gas's ability to cool and the corresponding time-scale. On one hand, if the local cooling time-scale is longer than τ, the internal energy produced by the instability balances out the heat loss nearly completely. On the other hand, if the cooling time-scale is shorter than τ, this balance is broken, i.e. the instability produces more internal energy than heat is lost, and thus self-gravitating clumps start to fragment out.

Analytical and numerical arguments have shown that due to the inefficient cooling of gas, the possibilities of fragmenting a disk inside several tens of AU appear remote under reasonable circumstances. The radial extent of a disk region subject to gravitational instability is $\sim \frac{2\pi H_h}{Q}$. If fragmentation continues, the number of clumps in the CSD increases. In a massive CSD, their masses may be up to a few or even tens of Jupiter's masses at hundreds of AU. The survival time of a clump depends on its ability to cool (because the clumps only contract once they are

[6] A ring of solids is a structure in the CSD, composed of pebbles and planetesimals.
[7] excluding the star's mass

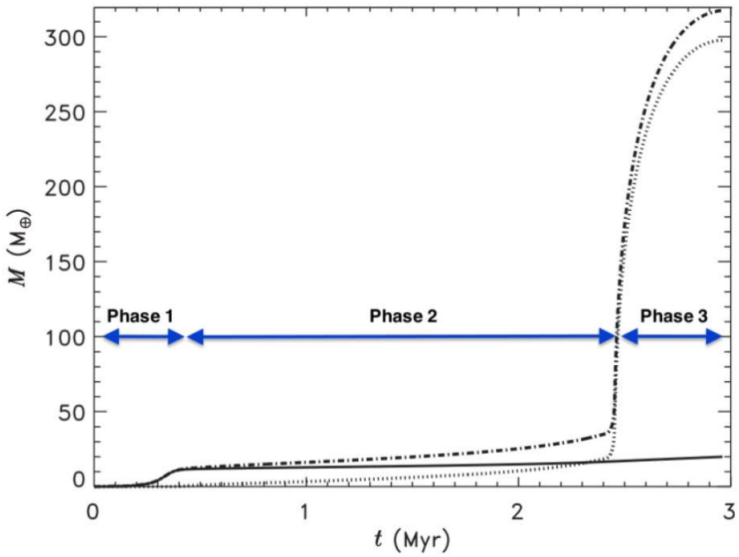

Figure 7: Illustration of the three phases in the core nucleated accretion model, aiming to describe the formation of giant planets, [22]: *Phase 1*: higher accretion rate of solids onto the core than gas accretion rate. *Phase 2*: envelope mass increases faster than condensed core's mass. *Phase 3*: envelope contracts; further gas accretion causes rapid growth in mass. *Solid line*: mass of heavy elements in the condensed core. *Dotted line*: envelope's H/He mass. *Dash-dotted line*: total mass of the planet.

cold enough) and contract before it is disrupted by interactions with other clumps (e.g. collisions that may disintegrate them) or other density perturbations. Furthermore, this method can only explain the formation of gas giants at large distances from the central star. In contrast to the formation by core nucleated accretion, giant planets that form via disk instability were long thought to have a high luminosity and entropy. It has been found later, that this idea is oversimplified, see [25]. Parameters such as the planet mass and whether the accreting gas passes through a supercritical shock front or not affect the both the entropy and the luminosity of the planet. Furthermore, faint planets are more difficult to detect, resulting in a biased observation sample. The formation of giant gas planets by disk instability has been assumed in several research papers. The entropy of a planet has been further studied in [26], in which the authors considering nebular gas that flows via the poles towards the planet and passes through such a shock front before it is being accreted. They have found that the initially high-entropy gas by passing though that shock is reduced, further confirming that the evolution of entropy of a planet that formed via disk instability is a complex process.
The major advantage of this method is that not only that it allows to explain planet formation on a short time-scale (between 10^4 and 10^5 years) but it can explain the formation of giants in the outer CSD regions as well, where the *Core Nucleated Accretion model* fails, see paragraph 1.3.2.1.
We note that the evolution of gravitationally unstable disks often leads to the formation of shock fronts in the inner CSD regions, which are driven by spiral arms (see figure 11 for spiral arms) and clumps at greater distances form the centre, [27].

1.3.2.2 Formation by Core Nucleated Accretion DONE Giant planet formation according to the core nucleated accretion model assumes already existing planetary embryo in the CSD, a condensed core that contains metals as well. In the classical model, this planetary embryo grows out of approximately $1 - 100km$ size bodies, referred to as planetesimals. Dust grains carried by the gas in a proto-planetary disk coagulate into larger particles, eventually forming planetesimals. The formation of planetary embryos is described in Section 1.3.1. Collisions among these planetesimals lead to the growth of a planetary embryo, which again accretes smaller bodies that lie around its orbit, until planetary core forms. When the gravitational energy at the surface becomes larger in magnitude than the thermal energy of the nearby gas, a core can accrete an atmosphere. The accretion rate of a H/He envelope distinguishes gas-rich planets from terrestrial-type, condensed planets. Planets, that become gas-rich in their early stage of the proto-planetary disk's evolution can become giant planets, which keep accreting gas as long as it is made available to them. Gas giant planets are mostly H and He by mass. In this model, the formation of giant planets follows three phases. Figure 7 schematically pictures them.

- *Phase 1*: The accretion rate of solid material onto the forming core exceeds that of gas.

- *Phase 2*: The mass of the envelope increases at a rate larger than that of the condensed core. This phase lasts until the envelope's mass becomes larger than the core's mass.

- *Phase 3*: The envelope contracts rapidly and ensuing gas accretion leads to a fast growth in mass.

After the disk's gas in the planet's neighbourhood disperses, the giant planet undergoes an isolation phase, during which it slowly contracts as it cools, possibly losing some of its gaseous component via evaporation driven by stellar irradiation if orbiting very close to the star.
The main difference between a gas-rich planet and a gaseous giant planet is that the growth of the former one is interrupted during Phase 1 or 2, because of intervening gas dispersal or gas starvation by other means. In the case of giant planets, gas starvation occurs during Phase 3.
A thorough analysis of the *in situ* formation of 47UMa, ρ CrB and 51 Peg have been performed by [23] by simulating the aforementioned process. They found that under certain circumstances, the *in situ* formation of these three giant planets is possible, but that orbital migration likely played a significant role in the evolution too.
Lissauer et al. have simulated (in 3D) the growth of Jupiter according to this model, see [28] by taking hydrodynamical processes between the planet and the CSD into account. They found that only gas within a radius of $\simeq 0.25 R_H$ remains bounded to the planet, while gas further away participates in the shear flow and that their simulated envelop could not capture the various satellites. As mentioned in the previous paragraph, planets that form via the core nucleated accretion model were thought to have both a low luminosity and entropy.
Further studies based on the core nucleated accretion model include for example [29], who aimed to derive estimates for the required mass of the protoplanetary disk M_D in order to produce a core of a planet of mass M_P within a lifetime τ. They have found that for $M_P \simeq 10 M_\otimes$ and $\tau \leq 10$ *Myrs*, which corresponds to the lifetime of nebular gas, a massive protoplanetary disk is required ($\sim$ 10 times the minimum-mass). Their simulations have further revealed, that the gas accretion onto the growing core stalls earlier than predicted by the theoretical model, such that the resulting core or protoplanet is of lower mass. Their conclusion is that this model only gives an upper limit for the efficiency of gas accretion. [30] discusses the accretion of solids and gas onto the planet. The authors found that accreting solids cause additional interactions with the disk of planetesimals during different phases of the planet's growth and evolution (enumerating the phases from 1 to 4). They have studied the accretion rate and its evolution during the different phases as well as the resulting luminosity, for example a tenuous atmosphere has an enhanced accretion rate of solids in the first phase, which decreases once the planet enters the second phase of evolution. We mention at this point, that the meridional circulation, and thus the accretion of material onto the planet, in the JUPITER code is traced only once the planet has grown to its full size. We discuss this further in 7. In [31], the authors simulated[8] the CPD in order to find intrinsic differences between the core-accretion (1) and the disk-instability (2) model regarding the mass and temperature of the CPD. Their findings include the conclusion that the CPD-mass scales linearly with the CSD-mass, thus, the CPD-mass in model (2) can be a factor of eight higher than in model (1), whereas the temperature differs by more than an order of magnitude. Temperatures of the CPD in case (1) are roughly $> 1000K$; in case (2) only $< 100K$. They concluded that the difference in temperature originates from the different potential wells and opacities of the CPD in the two models.

1.4 Formation of Planetesimals

TODO In this Section, we briefly introduce the main aspects of the formation and evolution of planetesimals in the CSD, from [32]. For detailed insight on the computations, we refer the reader to this paper and further literature in there.
If region in the CSD cool below a certain temperature, small dust particles can condense out and in the following fall towards the equatorial plane due to conservation of angular momentum. During this time interval, the particles accrete further material. The growth rate is given by

$$\frac{dr}{dt} = \frac{\alpha v_T \rho_{gas}}{\rho_{particle}},$$

[8] The simulations have been performed using the JUPITER code.

where α is the mass fraction of material in terrestrial planets, v_T the thermal velocity of the molecules in the gaseous phase and r the radius of the particle. It has been found that the particle's mass is bound by an upper limit when reaching the equatorial plane. Comparing the gas drag force F_D

$$F_D \sim \pi r^2 \rho_{gas} c v_z,$$

with c the speed of sound and v_z the z-component of the particle's velocity, with the gravitational force of the star, we find

$$v_z \sim \Omega^2 \cdot \frac{\rho_{particle} z \cdot r}{\rho_{gas} c},$$

with Ω the Keplerian angular velocity. The upper limit for the size of a planetesimal when reachign the equatorial plane is the given by

$$R \sim \frac{\alpha^{1/2} \sigma_{gas}}{\mu^{1/4} \rho_{gas}},$$

with μ the mean molecular weight. To obtain an upper estimate for the limiting mass, we simply use

$$M \sim \frac{4\pi}{3} R^3 \cdot \rho_{particle}.$$

The resulting disk of small-sized solid grains is gravitationally unstable and the planetesimals group together to form clusters with up to 10^4 members. These cluster rotate and the particles have non-trivial random velocities. The surrounding gas fluid exerts a drag force onto these particles, damping their motion and thus causing the cluster to further collapse. Note that as described in more detail in Section 1.6, small particles move essentially with the gas, while the gas drag force has a less strong effect on the motion of more massive planetesimals. The contraction and thus reduction of volume of the cluster increases the number of collisions between two planetesimals. Colliding particles then merge together, successively forming larger and more massive objects.

1.5 Particle Flow DONE

Hereafter, we discuss the particle flow in the CSD in general. A more detailed look at the meridional circulation is provided in Section 1.6. This Section is based on [2].
The overall trajectories of a particles / motion of the fluid in the CSD are governed by the Euler equations, that is, the mass continuity equation together with the conservation equation for angular momentum and mass,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{5}$$

$$\frac{\partial (\rho \vec{v})}{\partial t} + \nabla \cdot (\rho \vec{v} \otimes \vec{v} + P \cdot id) = \text{source terms} \tag{6}$$

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot ((\rho E + P) \vec{v}) = \text{source terms}, \tag{7}$$

where ρ, $\vec{v}$ are the gas's density and velocity respectively, E the total energy density and P the pressure, id denotes the unit tensor. These equations are again discussed in Section 2.2, where the JUPITER code is introduced and its solving schemes.
Together with the equation of motion,

$$P = \rho c_s^2, \tag{8}$$

with c_s the sound speed, in the isothermal case, several research groups in the past have solved the Euler equations (4)-(6), aiming to study and simulate different aspects and processes of and in an isothermal CSD, including the motion of fluids, planet formation, the planet's gap opening and various others; see e.g. [33], [34], [35] and [36]. However, it has been found that including various adiabatic processes, such as adiabatic compression, viscous heating, stellar irradiation, adiabatic expansion and radiative diffusion, and thus taking changes in temperature in the CSD and radiative effects due to the central star into account can yield different results. Thus, the adiabatic equation of motion,

$$P = (\gamma - 1)\varepsilon, \tag{9}$$

with γ the adiabatic index and $\varepsilon = \rho c_V T$ is used. Many studies have been performed that include adiabatic effects in order to simulate the physical nature of the CSD and its various components more accurate. A few examples

are [37], who studied the meridional circulation and delivery of solids onto the circumplanetary region, [38], who studied how efficacious the recycling hypothesis is in preventing the atmosphere from collapsing. In [39], thermal torques that are exerted by low-mass planets in a CSD are analysed and their contribution to the total torque is compared to the contribution of the Lindblad and co-rotation torque. Their simulation thus includes the thermal diffusion. In [40], the authors focussed on the migration (type 1 in isothermal CSD) of a low-mass planet and the effects of a thermal torque, which is exerted by the exchange of angular momentum between the (growing) planet and the surrounding gas. As they mention, in the isothermal case, the inward migration of the planet is relatively fast. Including non-isothermal effects, this migration is found to be slowed down or can be reversed, depending on various parameters, such as the planets luminosity. [41] studied the force that is exerted on a massive, luminous object, that moves with a constant speed through a homogeneously distributed, opaque gas. The inclusion of the luminosity of the perturber gives rise to additional radiative feedback on top of thermal diffusion in the gas. In [42], the authors investigated further on the topic of [41], focussing on the flow within the Bondi-sphere (by using a high resolution grid). They analysed the aforementioned force again as a function of mass and luminosity of the perturber, now int he subsonic regime.
To summarize, the resulting simulations show that in several situations, the adiabatic treatment is not only more accurate but necessary, for example when studying the interaction between the CPD and the CSD, as the former one is not isolated from the latter. Considering a non-constant temperature of the environment leads to more complex interactions between the fluids and particles, which we discuss further in Section 1.6.

1.6 Meridional Circulation DONE

Simplified, the meridional circulation is the motion of gas and other fluids or particles in the surrounding of a planet situated in the CSD, including the infall onto the planet via the poles and the outward spiralling motion close to the equatorial plane. Accretion occurs only when the velocity of the particles is reduced enough, for example by passing through a shock front, since only then the interaction time between the planet and the particle is long enough for the particle to be captured. Material with high velocity will thus not accrete onto the planet. In short, the inclusion of adiabatic effects as listed in Section 1.5 create spiral arms around the planet.
In this Section, we discuss the meridional circulation in more detail, focussing on the main components that are used in this work. Recent studies have shown, that the planets' surroundings - CPDs or CDEs - are not isolated from the CSD, [37], [43] and [44]. Hereafter, we give a brief introduction to the particle flow between the CSD and the CPD.

Figure 8 from [35] shows the particle flow around a planet at a distance a from the star. The yellow and green streamlines describe the inner and outer flow from the stellar disk; the blue and red ones describe the co-rotational horseshoe flow, that exchanges angular momentum with the planet, the so called horseshoe drag; and in magenta the bound streamlines between the stagnation points (black crosses) describe the atmosphere of the planet.
In contrast to this, figure 9 from [35] shows the corresponding plane of the 3 dimensional simulation. In this case, the midplane of CPD does not form a bound atmosphere around the planet, but rather an outflux of material back to the CSD. It has been shown, e.g. in [38], that all material of the CPD is recycled in a finite amount of time[9]. The outflux in the disk midplane creates a low pressure region at higher altitude, which attracts new material, see figure 10: the influx from the poles is shown in green / blue, the outflux near the disk midplane in red.

In the following, we discuss the orbits of dust particles in more detail as simulated by [33].
Small particles with size $r \leq 1m$ are strongly coupled to the gas and therefore follow its motion, as shown in figure 11. The smallest simulated particles ($r = 0.01m$) do not accrete onto the planet, while with increasing particle size ($r = 0.1m - 1m$) a larger fraction enters the Hill sphere of the planet. The reason for this is that after these particles pass through the shock surface (coming from the direction of the poles), their velocity is reduced which causes them to decouple from the gas's motion. For the $r = 10m$-sized particles, this effect is negligible[10], because their coupling to the gas is insignificant and therefore they are captured. Furthermore, we highlight that with increasing size of the particles, they approach the planet closer.

For the discussion of large particles' orbits, we introduce an impact parameter b, that describes the regime

[9] This is the *Recycling Hypothesis* or *transient horseshoe flow*, that is based on the fact that the CPD or CPE is not isolated from the CSD and dust, gas and other material is continuously exchanged. For further information, the reader is referred to [38].
[10] A good evidence for the negligibility of the gas drag / decoupling are the crossed orbits.

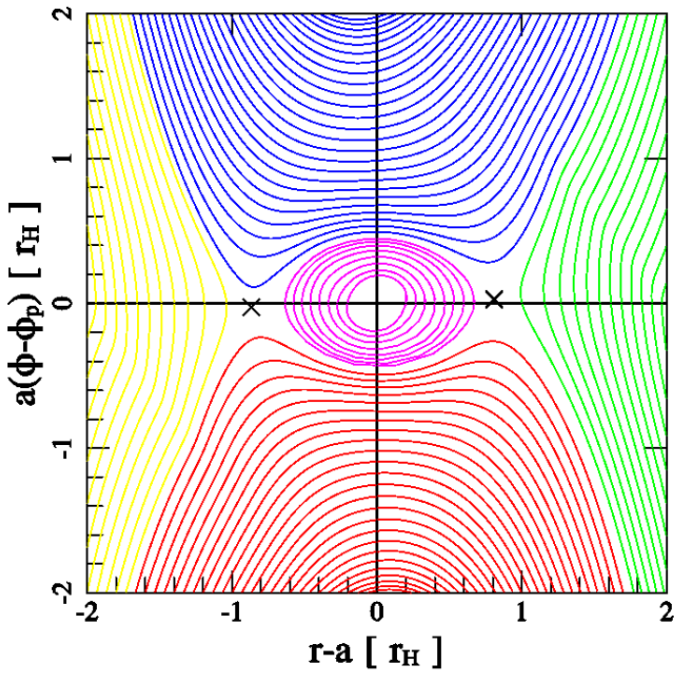

Figure 8: Particle flow in 2 dimensions around a planet at distance a from the star. The yellow and green streamlines describe the inner and outer flow from the stellar disk; the blue and red ones describe the co-rotational horseshoe flow, that exchanges angular momentum with the planet (horseshoe drag); and in magenta, the bound streamlines between the stagnation points (black crosses) describe the atmosphere of the planet, [35].

of pro- and retrograde trajectories[11]. Starting with the prograde orbits, i.e. small and large b (see [33]), figure 12 shows, that while $0.1m$-sized particles do not enter the Hill sphere, for $r = 1m$ to $100m$, a fraction of particles does enter the Hill sphere after loosing kinetic energy by passing through the shock surface, and these particles spiral towards the planet, maintaining their initial prograde orbit. We highlight that with increasing radius r, the circularization radius R around the planet decreases, i.e. large particles approach the planet closer due to the decreasing effect the gas drag exerts on them. Thus, the $10000m$-sized particles move free from the gas drag and have therefore a very long circularization time. They orbit the planet thus on an eccentric (prograde) orbit.
Figure 13 shows the flow for retrograde orbits, where again an increased accretion onto the CPD can be observed with increasing particle size. Due to the gas drag however, particles with $r = 10m$ to $100m$ are forced into a prograde orbit when approaching the planet. Because this effect is too weak for $r = 10000m$, these latter ones remain on their initial retrograde orbit.

D'Angelo et al. (see [45]) have performed an isothermal simulation, attempting to study the interaction between the planet and the CSD with a varying planet-to-star mass ratio $q := \frac{M_P}{M_{star}}$. They found that within the Hill sphere of the planet, a CPD with spiral arms (over-dense regions; see figure 14) formed. The strength, curvature and size (length of the arms) of the spiral arms depend on the planet mass M_P and become stronger with increasing M_P. Furthermore, a detailed analysis of the total torque revealed that nearby material can cause the planet to slow down its inward migration in the CSD[12].

1.7 Gap Formation and Gas Profile DONE

The formation of gaps in the CSD is a complicated process, which depends on several parameters as well as other astrophysical phenomena in the surrounding environment. Several models have been built to simulate the forma-

[11] Prograde orbits occur when the particle forms in the disk itself and thus obeying the conservation of angular momentum. Retrograde orbits occur, when a particle is captured by the planet or migrates towards it after a planet-planet collision.

[12] Their argumentation is as follows: if the overall density distribution was symmetric, a vanishing net torque would act on the planet. But a planet within the CSD alters the overall distribution of the gas and other fluids due to accretion and possible gap formation (for gap formation, see Section 1.7). As a result, the acting torque is non-trivial and because of conservation of angular momentum, the planet's radial position changes over time, leading to a migration motion of the planet, see Section 5.3 in [45].

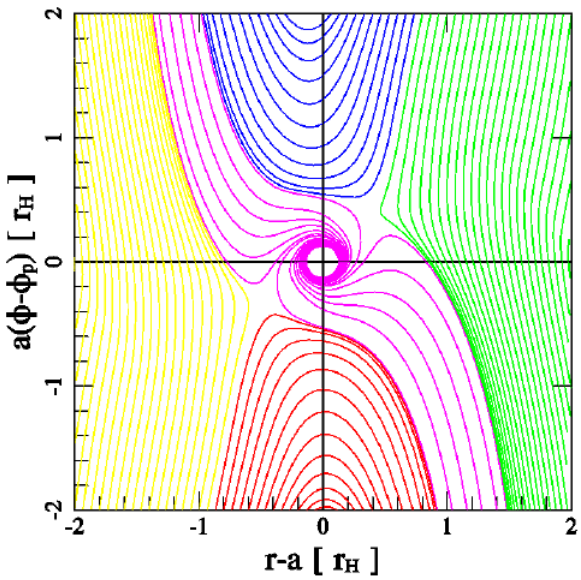

Figure 9: Particle flow in 3 dimensions, with the same colour code as in figure 8, [35]. The main difference is that in the 3D case the magenta streamlines are not bound to the planet but spiral outwards in the midplane of the CPD, i.e. interaction and material exchange between the CPD and the CSD occur.

tion of gaps by planets in two and three dimensions, as well as in isothermal and adiabatic disks, e.g. [47], [48] or [49]. In the following, we give a brief overview on the formation process of planetary gap and the findings of several studies.
Oversimplified, the formation of a gap in the CSD occurs when a planet starts to form and thus changing the total potential over time. As a result, the (forming) planet interacts in several ways with its surroundings and eventually accumulates gas, dust or even larger astrophysical objects in the vicinity of its orbit, which is then depleted of these materials. The shape (depth and width) of the gap depends for example on:

1. the mass of the planet, as more massive planets are capable of attracting more material by gravitation. In figure 15, we can see that massive planets ($M_P = 10 M_{Jup}$) create gaps that reach over a wider radius range.

2. the temperature of the planet, which can be seen in figure 15 as well. [46] have simulated the influence of the planet's temperature on the gap formation, taking various CSD-masses into account as well. One of their main conclusion was that the depth and width of the gap increase with decreasing planet temperature T_P. They have further shown, that the gap profile is not symmetric with respect to the radius.

3. several (counter-)acting torques: the gravity torque t_g, originating form the planet; the torque due to the CSD-viscosity t_ν and the pressure torque t_P, which can be interpreted as waves that carry angular momentum[13]. In case of thermal equilibrium of the CSD, these three torques cancel out. Using this condition, [50] have determined the structure of the gap: in a disk with uniformly distributed density, the pressure gradient vanishes, while in general it can be approximated as

$$t_P = -a(r) \cdot \left(\frac{d\Sigma}{\Sigma \cdot dr} \right), \quad a(r) = \left(\frac{H}{r} \right)^2 \cdot r R_H \cdot r_P \Omega_P^2 \cdot a'' \left(\frac{\Delta}{R_H} \right), \tag{10}$$

where Σ the surface density, R_H, r_P, Ω_P the planet's Hill radius, radial distance from the star and orbital frequency; $\Delta := r - r_P$. With the fit[14] the last factor in this equation, $a''\left(\frac{\Delta}{R_H}\right)$, and find that in equilibrium,

[13] This statement is non-trivial and in fact one of the findings in their analysis. Section 2.2 to 3.1 in [50] explain the physics and calculation in great detail and the reader is warmly invited to have a look at them.
[14] This fit assumes a planet-to-star mass ratio of 10^{-3} (typical Jupiter-mass planet), a disk viscosity of $10^{-5.5}$ and $H/r = 5\%$.

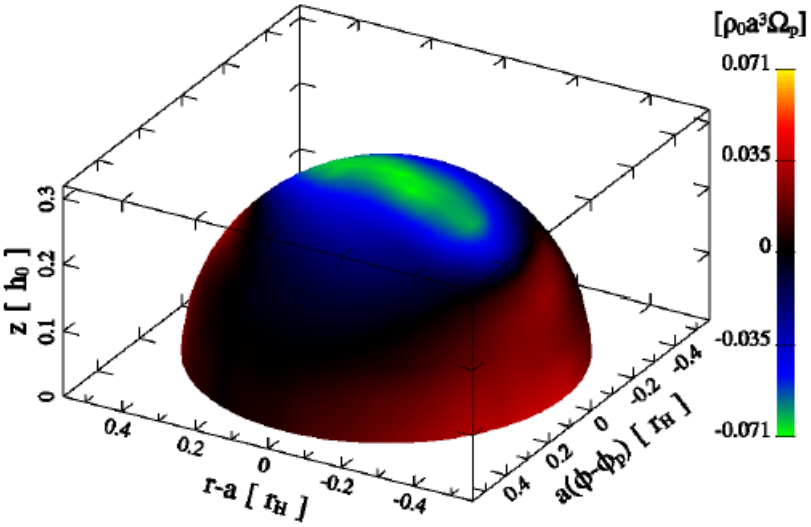

Figure 10: The mass flux for positive altitudes, assuming symmetry with respect to the midplane. The material influx from the poles is shown in green / blue, the outflux in the disk midplane is shown in red, [35].

the gap profile then reads:

$$\left(\frac{R_H}{\Sigma}\frac{d\Sigma}{dr}\right) = \frac{t_g - \frac{3}{4}\nu\Omega}{\left(\frac{H}{R}\right)^2 r r_P \Omega_P^2 a'' + \frac{3}{2}\nu\frac{r}{R_H}\Omega}. \tag{11}$$

The interested reader is referred to figure 8 in [50], that shows a number of gap density profiles for varying ν, q and H/r. We highlight, that formula (10) depends on the planet-to-star mass ratio via t_g (see equation (11) in [50]) and thus on the planet's mass and torques, as expected. For increasing q, the gap becomes more prominent.

4. the migration movement of the planet within the CSD alters the gap as well due to the changing potential. We distinguish between the following types of migration:

   - *Type 1:* This migration occurs, when the planet's mass is not large enough to change the local density of the CSD enough, and the resulting motion is directed towards the central star with a velocity, that is proportional to the planet's mass.

   - *Type 2:* This migration occurs, when the planet's mass is so large, that a considerable gap forms. The direction of the migration depends on the viscosity ν of the CSD, as discussed above.

   - *Type 3:* This migration occurs, when the mass of the planet is in intermediate range and instead of a gap, a dip in the surface density around the planet's orbit can be observed. In this case, a runaway migration can occur, where the drift rate increases exponentially.

The direction of a planet's movement depends on the acting torques and the planet mass, and can be directed in- or outward, see [50]. In the case of a CSD with two embedded planets, simulations have shown that their migration can be either convergent or divergent, see e.g. [51], who have studied the divergent migration in more detail, with a special focus on what combination of planet masses and CSD mass would allow for a divergent migration and the formation of a gap. Such a pair of planets can form a gap in the planetesimal disk, of roughly 0.1% of the minimum mass solar nebula (MMSN). The migration for a three-planet system has been simulated by [52]. Note, that both these studies have concluded that the presence of a second or even third planet can cause significantly changes to the overall behaviour of the migration and gap formation.

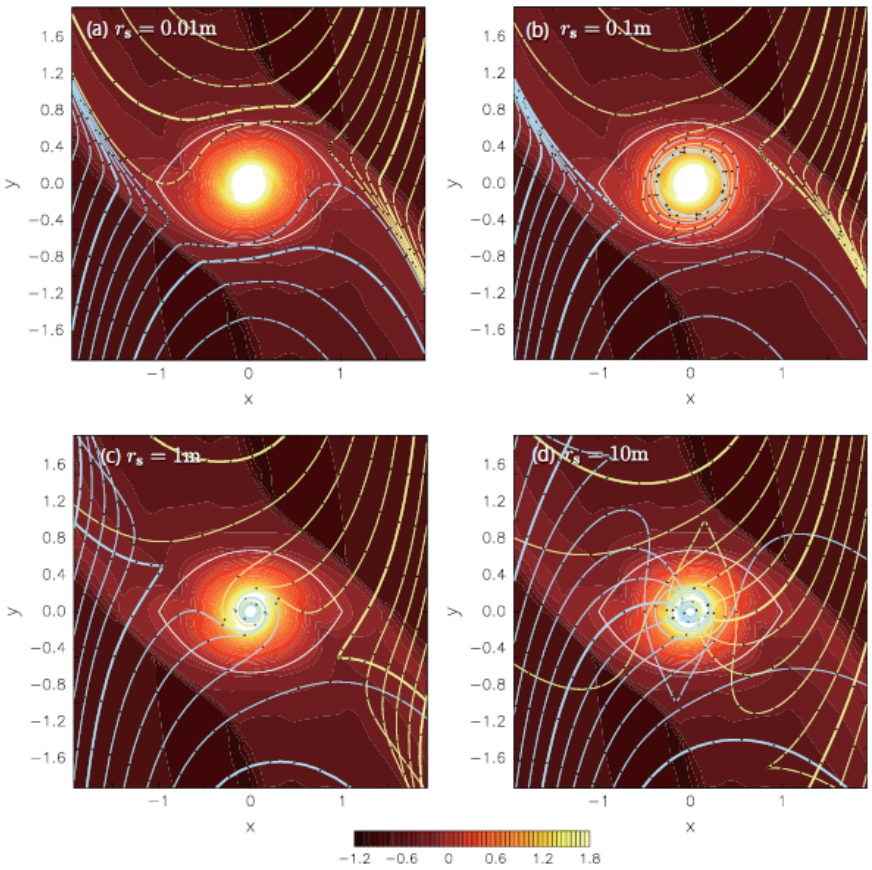

Figure 11: Trajectories for small particle sizes (up to $r = 1m$). The smallest particles do not enter the Hill sphere of the planet due to their strong coupling to the gas fluid. With increasing size, some particles can be captured by the denser gas in the CPD in regions closer to the planet and their orbits cross, which is a typical feature for decoupled particles, [33].

5. Other studies, which focused on a CSD with a single embedded planet, have shown that the structure and evolution of the gap depends on the cooling time of the disk (see [53][15]), in the sense that with increasing cooling time t_c, the gaps in their simulation became more stable, the growth rates of the non-axisymmetric modes[16] decreased and the dominant wavenumber m[17] decreased.

6. In his master thesis [54], F. Binkert studied planetary gaps in the dust and gas fluid in the JUPITER code by introducing a new pressure-less fluid that simulates the dust in the CSD and a function that couples the dust fluid to the gas fluid, thus taking into account that the two components interact with each other. Simplified, while the gas in the CSD, moves with a sub-Keplerian velocity, the dust experiences a headwind due to the gas. The underlying exchange of angular momentum between these two fluids causes the dust particles to spiral inwards towards the central star. This has been accounted for by introducing an additional drag term F_{drag} to the momentum equation (5), see Sections 2.3 and 4.3 in [54]. The main findings of that work are that for opening a gap in the gas fluid, a higher planetary mass is required than for opening a gap in the dust fluid. The simulations have further revealed that the gap in the gas is deeper the in 3D or adiabatic case than in the 2D or isothermal version. When including dust, the gap profiles are narrower. Simulations with varying temperatures and density distributions of the fluids underlined the importance of taking adiabatic effects into account.

[15]The simulation assumed an inviscid disk, such that the formation of Rossby wave instabilities is favoured. A Rossby wave instability is the phenomenon when in a disk of low viscosity, potential vorticity extrema can cause the gaps to become dynamically unstable. To measure the relative vorticity, the Rossby number Ro has been introduced, and $Ro < 0$ corresponds to a rotation in anti-cyclonic direction w.r.t. the rest of the disk.

[16]in Fourier-space; see eq. (13) in their paper

[17]m is defined in their eq. (12) via the Fourier transform of the time-dependant surface density.

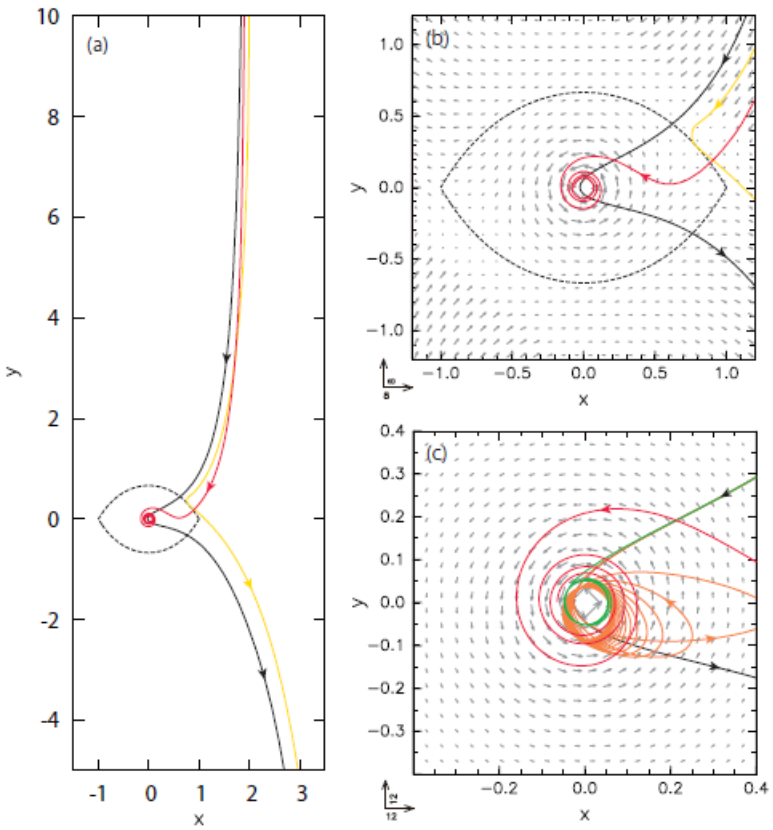

Figure 12: Flow of large particles ($r \geq 1m$) on a prograde orbit, [33]. For $r = 0.1m$, the particles do not enter the Hill sphere of the planet, due to the strong coupling to the gas. With increasing size ($r = 10m$ and $r = 100m$) some accretion onto the CPD occurs and the particles spiral towards the planet until the orbit is circularized. For $r = 10000m$, the gas drag is negligible and the circularization time very long, thus they are on an eccentric orbit around the planet.

1.8 Structure of this thesis DONE

This work is organized as follows:

- In Section 2, we introduce the JUPITER code, the main equations (conservation equations and radiative transfer), the two modules *isothermal* and *adiabatic* (radiative) as well as an overview of the most common methods (SPH and AMR) to simulate / solve these equations. Furthermore, we give a brief introduction to the most important numerical methods for this code, such as the Godunov scheme, the Courant-Friedrichs-Lewy condition and the Riemann solver.

- In Section 3, we discuss the main properties of a tracer fluid and how it has been implemented to the JUPITER code. For completeness, we first introduce two methods how the tracer fluid can be implemented, namely the *Monte Carlo tracer fluid* and the *Velocity tracer fluid*, and then focus on the technical methods and difficulties how the tracer particles have been implemented. For that purpose, we show selected sections of the functions that have been written in the process. We first discuss the implementation and arising difficulties for massless tracers, followed by the implementation of massive tracer particles.

- In Section 4, we show the results of the simulation. Furthermore, we list the different tests that we have performed, including the compatibility of the tracer particle files on multiple CPUs and with multiple levels of grid refinement. For the compatibility tests, we have used a planet of mass $M_P = 1M_{Jup}$. For the other

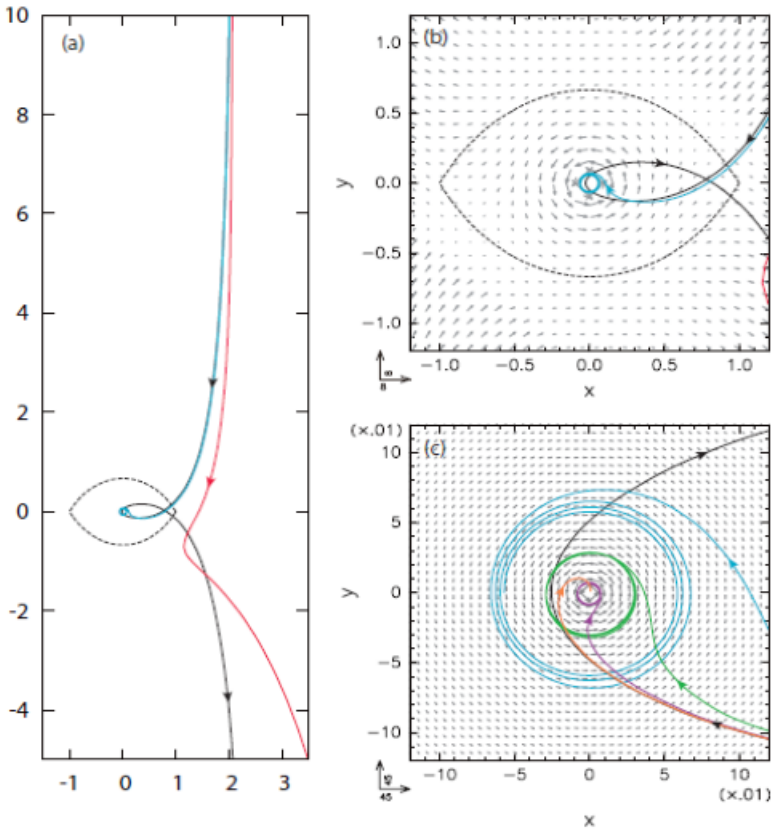

Figure 13: Flow of large particles ($r \geq 1m$) on a retrograde orbit, [33]. Particles of size $r = 10m$ to $r = 100m$, that are initially on a retrograde orbit, are forced onto a prograde orbit due to the higher gas density close to the planet. In the case of $r = 10000m$, the gas drag is too weak, thus these particles remain on their retrograde orbit.

simulations, we have used different planet masses, ranging from 10^{-4} (in code units) to 10^{-2} to study the influence of the planet mass on the meridional circulation.

- In Section 5, we discuss the main conclusions and findings of this work from the results of the simulation and compare the simulated meridional circulation to the motion found by other research groups.

- In Section 6, we briefly summarize in note form first the process and methods how the tracer particles have been implemented in the JUPITER code and then give an overview of the results.

- Finally, in Section 7, we discuss how the code part including the tracer particle implementation could be further improved and enhanced, as well as possible other areas of application in astrophysics.

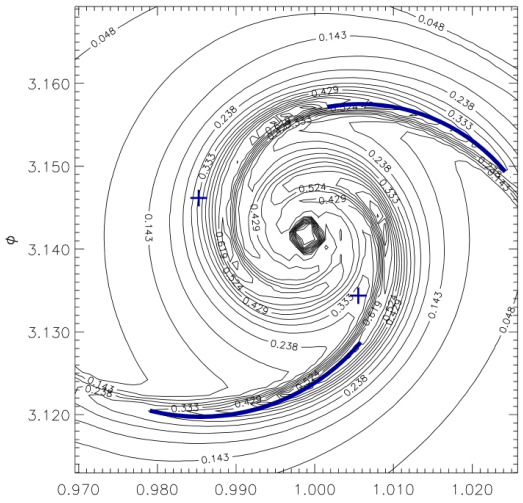

Figure 14: Contour plot of the surface density, [45]. The shape and strength of spiral arms formed by the planet depend on its mass.

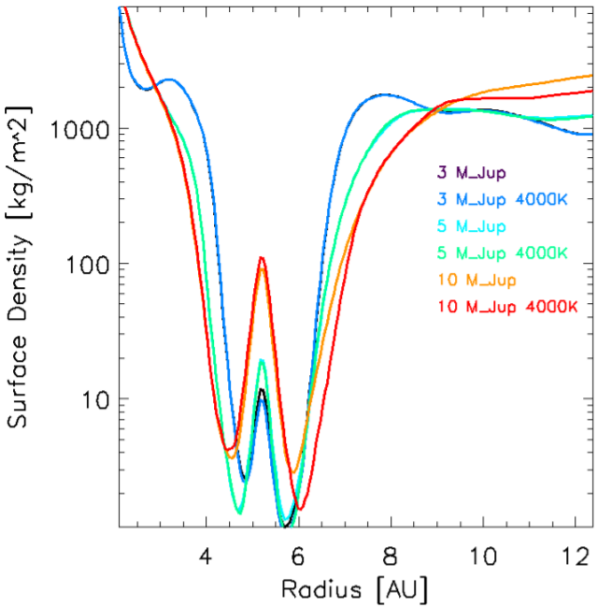

Figure 15: Gap profile for massive gas planets with $M_P = 3 - 10 M_{Jup}$; averaged over time. The profile is asymmetric with respect to the radial co-ordinate and depends on the planet's temperature, see [46].

2 THE MODEL

2.1 JUPITER Code: Preface

TODO The simulations in this work have been carried out with the JUPITER code, with was developed by F. Masset, see [1], and J. Szulágyi added the radiative treatment, see [2]). The code is parallelized using the Message Parsing Interface (MPI) library and each processor solves the Euler equations (see Section 2.2 for the governing equations) only for the fluid cell lying on its computational sub-domain. With this method, the fluid is transferred between neighbouring cells when it crosses the boundary of a processor, which is equivalent to crossing the boundary of that specific grid cell, i.e. each processor has to communicate only with its neighbours.

2.2 JUPITER Code: Governing Equations DONE

The JUPITER code solves the mass- / angular momentum- and energy-conservation equations on a nested mesh in 3 dimensions, [2]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{12}$$

$$\frac{\partial (\rho \vec{v})}{\partial t} + \nabla \cdot (\rho \vec{v} \otimes \vec{v} + P \cdot id) = \text{source terms} \tag{13}$$

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot ((\rho E + P)\vec{v}) = \text{source terms}, \tag{14}$$

where ρ, P, $\vec{v}$ denote the density, pressure and velocity of the fluid, respectively, id is the unit tensor and E the energy density.

As described in [2] and [56], the energy equation (13) with the source terms can be written as

$$\frac{\partial E}{\partial t} + \nabla (E\vec{v}) = -\rho \vec{v} \cdot \nabla \Phi + \nabla \cdot ((-P \cdot id + \overline{\tau}) \cdot \vec{v}) - \nabla \cdot F_{rad} + S, \tag{15}$$

with Φ the gravitational potential, F_{rad} the radiative flux and $\overline{\tau}$ is the stress tensor, given by

$$\overline{\tau} = 2\rho\nu \left(\overline{D} - \frac{1}{3} (\nabla \vec{v}) id \right), \tag{16}$$

with $\overline{D}$ the strain tensor. The term on the RHS of equation (14) including the stress-tensor $\overline{\tau}$ thus takes heating effects due to shear in the case of a non-vanishing kinematic viscosity ν in the disk into account. Furthermore, $-\rho \vec{v} \cdot \nabla \Phi$ describes the work done on the fluid by all external forces, $\nabla \cdot ((-P \cdot id + \overline{\tau}) \cdot v)$ is the advection and viscous heating and the last term, S, describes the stellar heating.

Because the planet's presence in the CSD affects the physics, the gravitational potential has to be corrected. In the JUPITER code, this is done by a *smoothing parameter*. As described in [45], a constant smoothing length over the whole CSD-grid and all levels of refinement can have some disadvantages and a level-adjusted smoothing length is to be preferred, see also comment in [2], p. 67.

The total energy E in equation (14) is given by

$$E = \varepsilon_{rad} + \varepsilon + \rho \frac{\vec{v}^2}{2}, \tag{17}$$

where $\varepsilon = \rho c_V T$ is the internal energy with c_V the specific heat at a constant volume. Note, that we can summarize $\varepsilon_{tot} := \varepsilon_{rad} + \varepsilon$ as the total internal energy of the fluid. The differential equation for the radiative energy[18] is derived in [56][19],

$$\frac{\partial \varepsilon_{rad}}{\partial t} = -\nabla F_{rad} + c\kappa_R \rho \left(\frac{B_\nu(T)}{c} - \varepsilon_{rad} \right). \tag{18}$$

[18] Note that we use $\kappa_R = \kappa_P$, see Section 2.3 and figure 1 in [57]: even though these two opacities are not equal in general, they take on similar values in a first approximation.

[19] Insert equation (16) into (14), then apply (11) and (24) after some algebra.

Here, κ_R is the Rosseland mean opacity, defined in equation (15) in [57],

$$\kappa_R^{-1} := \frac{\int \kappa_{\nu s}^{-1}(T,\rho)\frac{dB_\nu(T)}{dT}d\nu}{\int \frac{dB_\nu(T)}{dT}d\nu}, \tag{19}$$

with $\kappa_{\nu s}$ the frequency dependent opacity including scattering effects. $B_\nu(T)$ is the thermal blackbody function,

$$B_\nu(T) = \frac{2h\nu^3}{c^2}\frac{1}{exp(h\nu/k_BT)-1}. \tag{20}$$

The radiative flux is then written in the flux-limited approximation as (see [57])

$$F_{rad} = \frac{-c\lambda}{\rho\kappa_R}\nabla\cdot\varepsilon_{rad}, \tag{21}$$

with λ the flux limiter,

$$\lambda = \begin{cases} \frac{2}{3+\sqrt{9+10R^2}} & \text{if} \quad R \leq 2.0 \\ \frac{10}{10R+9+\sqrt{81+180R}} & \text{if} \quad R > 2.0 \end{cases} \tag{22}$$

with

$$R = \frac{1}{\rho\kappa_R}\frac{|\nabla E_R|}{E_R} \tag{23}$$

and E_R the radiative energy density. The parameter λ is introduced to account for different optical depths in the medium by allowing a smooth transition between them. In optically thick region we have $\lambda = \frac{1}{3}$, while in optically thin regions it is chosen such that

$$F_{rad} \to \frac{4\sigma T^4}{c},$$

with $\sigma = 5.670\cdot 10^{-8}\frac{W}{m^2K^4}$ the Stefan-Boltzmann constant.

The radiative transfer equation in the flux limited diffusion approximation is then given by

$$\frac{\partial(\rho E)}{\partial t} + \nabla\cdot((\rho E + P)\vec{v}) = -P\nabla\cdot\vec{v} + \nabla\left(\frac{4}{3\kappa_R\rho}\nabla(\sigma T^4)\right), \tag{24}$$

where $\rho, \vec{v}$ are the total gas and dust mass density and velocity respectively, P the gas pressure and T the gas+dust temperature.

The diffusion approximation is therefore included in σT^4 on the RHS of equation (23). It should be mentioned, that this approximation is valid in optically thick regions, e.g. in the disk midplane, where the optical depth is $\tau \sim 10^4$.

The set of equations (11)-(13) is completed by the adiabatic equation of state,

$$P = (\gamma - 1)\varepsilon, \tag{25}$$

with $\gamma = 1.43$ for the adiabatic index and $\varepsilon = \rho c_V T$.
The corresponding equation of state for the isothermal case is given by

$$P = \rho c_s^2, \tag{26}$$

where ρ is again the density of the fluid and c_s is the speed of sound, which can be written in terms of the temperature of the fluid,

$$c_s^2 = \frac{k_BT}{\mu}, \tag{27}$$

with $k_B = 1.380649\cdot 10^{-23}J/K$ the Boltzmann constant, T the *pre-defined, fixed* temperature and μ the mean molecular weight.

2.2.1 Isothermal Module DONE

As mentioned previously, the isothermal model assumes a constant, pre-defined, fixed temperature and solves the conservation equation for mass (11) and angular momentum (12) with the equation of state (25).
In this work, we include heating and cooling mechanisms, and thus use the adiabatic module, which is introduced in the following. We do not further discuss the isothermal model.

2.2.2 Radiative Module DONE

Several studies and simulations have shown that the CPD / CPE is not isolated from the CSD, and a rather complex interaction takes place between them, for example [35], [58] and [37]. For a realistic simulation of the material flow between the CSD and the CPD it is thus necessary to consider heating and cooling mechanisms, such as stellar irradiation, viscous heating, accretion heating of the gas, radiative dissipation, adiabatic expansion and compression as well as heating by shocks [37], by including the energy equation (13) and the radiative transfer equation (23) into the code and use the equation of state (24). In this work, we use this module for the gas fluid in the CSD.

2.3 The Riemann Solver

TODO?????????????????????????????? DONE?????????????????????????????? As mentioned in Section 2.2, the JUPITER code solves the Euler equations by making use of a high order Godunov scheme. This scheme is based on the Riemann solver, which we discuss in the following. For further detail and mathematics, the reader is referred to [59]. We summarize the key concepts and equations hereafter (focussing on Chapter 4.1 in [59] and Chapter 2 in [60]).

The Riemann problem is an inital value problem for a (set of) equations of conservation, such as the Euler equations in (11)-(13),

$$\frac{\partial U}{\partial t} + F\left(\frac{\partial U}{\partial x}\right) = 0, \tag{28}$$

for

$$\begin{aligned} U &:= (\rho,\ \rho u,\ E)^T \quad \text{vector of conserved variables} \\ F &:= (\rho u,\ \rho u^2 + p,\ u(E+p))^T \quad \text{vector of fluxes,} \end{aligned}$$

with $\rho = \rho(x,t)$, $u = u(x,t)$, $E = E(x,t)$ and initial conditions

$$U(x,0) = \begin{cases} U_L & \text{if} \quad x < 0 \\ U_R & \text{if} \quad x > 0 \end{cases}$$

According to equation (16), we can write the energy in terms of pressure, velocity and density, motivating the vector

$$W := (\rho, u, p)^T.$$

The situation is shown in figure 16. Together with the initial conditions, we have $W_L = (\rho_L, u_L, p_L)^T$ for $x < 0$ and $W_R = (\rho_R, u_R, p_R)^T$ for $x > 0$. Solving the eigenvalue problem (27) we obtain the solutions

$$\begin{aligned} \lambda_1 &= u - a, \\ \lambda_2 &= u, \\ \lambda_3 &= u + a. \end{aligned}$$

These eigenvalues describe waves and separate four constant states given by W_L, W_{*L}, W_R and W_{*R}, see figure 16. Note that the wave corresponding to $\lambda_2 = u$ is a *contact discontinuity* while the waves corresponding to $\lambda_{1,3} = u \pm a$ are *shock waves* or *rarefaction waves* and are thus non-linear.
In order to obtain solutions for the pressure and the fluid velocity in the star region, we introduce

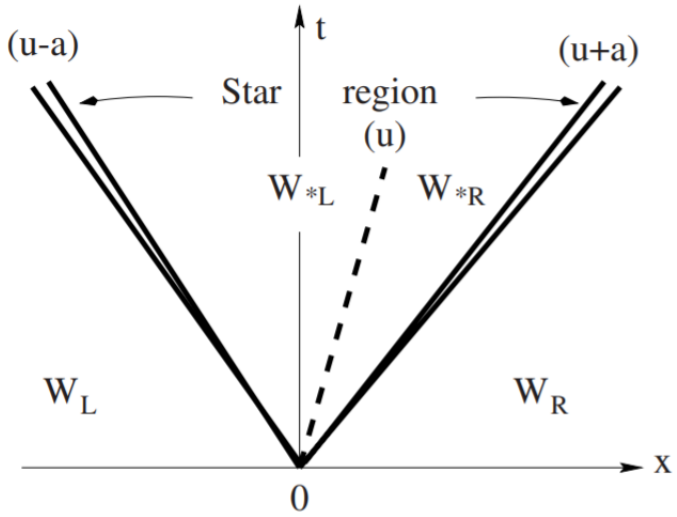

Figure 16: Set up of the Riemann problem for 1D Euler equations, [59].

Corollary 1 (Solution for p_* and u_*) *The solution for pressure, p_*, of the Riemann problem (27) is given by the root of the algebraic equation*

$$f(p,W_L,W_R) := f_L(p,W_L) + f_R(p,W_R) + \Delta u = 0, \tag{29}$$

where $\Delta u := u_R - u_L$, and

$$f_L(p,W_L) = \begin{cases} (p-p_L)\left(\frac{A_L}{p+B_L}\right)^{1/2} & \text{if} \quad p > p_L \text{ (shock)}, \\ \frac{2a_L}{\gamma-1}\left(\left(\frac{p}{p_L}\right)^{(\gamma-1)/2\gamma} - 1\right) & \text{if} \quad p \le p_L \text{ (rarefaction)}, \end{cases}$$

and

$$f_R(p,W_R) = \begin{cases} (p-p_R)\left(\frac{A_R}{p+B_R}\right)^{1/2} & \text{if} \quad p > p_R \text{ (shock)}, \\ \frac{2a_E}{\gamma-1}\left(\left(\frac{p}{p_E}\right)^{(\gamma-1)/2\gamma} - 1\right) & \text{if} \quad p \le p_E \text{ (rarefaction)}, \end{cases}$$

with constants that depend on the data

$$\begin{aligned} A_{L,R} &= \frac{2}{(\gamma+1)\rho_{L,R}}, \\ B_{L,R} &= \frac{\gamma-1}{\gamma+1} p_{L,R}. \end{aligned}$$

The final solution for the velocity, u_*, is then given by

$$u_* = \frac{1}{2}(u_L + u_R) + \frac{1}{2}(f_R(p_*) - f_L(p_*)). \tag{30}$$

2.4 The Godunov scheme DONE

This Section is based on [59][20].
The Godunov scheme is a *conservative scheme* for computing numerically the solution of a (system of) partial differential equations, i.e. for a scalar conservation law

$$0 = \frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x}, \tag{31}$$

[20] see Sections 5.3.2 and 5.3.3

with f the function of fluxes, we can write

$$u_i^{n+1} = u_i^n + \frac{\Delta t}{\Delta x} \cdot \left(f_{i-1/2} - f_{i+1/2} \right),$$

with

$$f_{i+1/2} := f_{i+1/2} \left(u_{i-l_L}^n, ..., u_{i+l_R}^n \right),$$

where l_L and l_R are two non-negative integers and $f_{i+1/2}$ is an approximation to the physical flux $f(u)$, satisfying the *consistency condition* (see also [60] for further details)

$$f_{i+1/2}(v, ..., v) = f(v).$$

When discretising each space direction into a finite amount of cells I_i with boundaries

$$x_{i-1/2} \leq x \leq x_{i+1/2},$$

the average value of $u(x,t)$ in cell i at a fixed time $t := t^n$ is given by

$$u_i^n = \frac{1}{\Delta x} \int_{i-1/2}^{i+1/2} u(x,t^n) dx.$$

We highlight that this average is constant over the whole cell and assign it therefore at the centre of the cell, since the $u(x,t)$ may have spatial variations.

With the Godunov method, the fluxes $f_{i+1/2}$ are computed by using solutions of the local Riemann problems, see Section 2.3, i.e. for each cell at a certain time. We then need to determine a global solution at a later time. The method is described in the following.
Suppose the Riemann problem (i.e. equation (30) with initial conditions) has been solved for the cells I_{i-1} (with boundaries x_{i-1} and x_i) and I_i (x_i, x_{i+1}) at a fixed time t^n, see figure 17 (left). We then take an integral average in the cell I_i of the combined solutions; the value is assigned to u_i^{n+1}, see figure 17 (right).

The combined solution is then given by

$$u_i^{n+1} = \frac{1}{\Delta x} \left(\int_0^{\Delta x/2} u_{i-1/2} \left(\frac{x}{\Delta t} \right) dx + \int_{-\Delta x/2}^0 u_{i+1/2} \left(\frac{x}{\Delta t} \right) dx \right),$$

and is therefore at a later time t^{n+1}. Note that Δt satisfies the CFL-conditions, see Section 2.5. The interval lengths are given by

$$l_{AB} = c \cdot \Delta x \text{ and } l_{BC} = \left(\frac{1}{2} - c \right) \Delta x$$

and thus

$$u_i^{n+1} = u_i^n + c(u_{i-1}^n - u_i^n)$$

2.5 The Courant-Friedrichs-Lewy (CFL) Condition and Boundary Conditions DONE

This Section is based on [59].
Let $[0,\ L]$ be a finite computational domain, e.g. the domain in x-direction, which we discretise into M cells I_i, $i = 1, ..., M$. In this domain we want to solve the Euler equations[21], which requires us to first apply boundary conditions for $x < 0$ and $x > L$. Before doing so, we extend the domain by two ghost cells at each end with indices $i = -1,\ 0$ for $x < 0$ and $i = M+1,\ M+2$ for $x > L$. The two options for the boundary conditions are:

- *Reflective boundary condition.*
  For simplicity, we assume a solid boundary only at $x = L$[22], denoted by u_{wall} and find
  $\rho_{M+1}^n = \rho_M^n$, $\rho_{M+2}^n = \rho_{M-1}^n$
  $u_{M+1}^n = -u_M^n + 2u_{wall}$, $u_{M+2}^n = -u_{M-1}^n + 2u_{wall}$
  $P_{M+1}^n = P_M^n$, $P_{M+2}^n = P_{M-1}^n$.

[21] We highlight that the CFL-condition is important for PDEs on a *finite* domain on which we want to solve an equation of advection. It is not important to consider when solving a steady state problem, [61].
[22] Similar conditions can be applied for a reflective boundary $\tilde{u}_{wall}$ at $x = 0$.

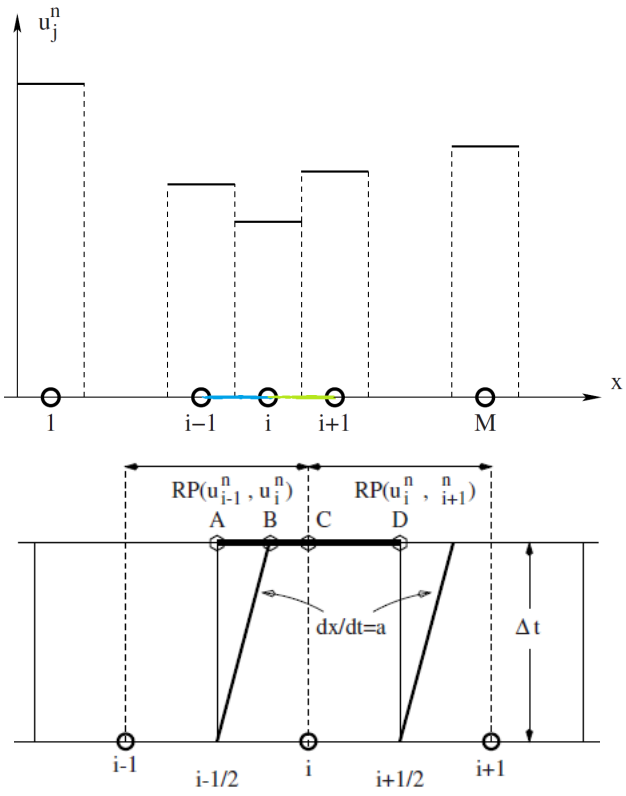

Figure 17: Schematic illustration of the Godunov method. Top: Data distribution at time t^n. Assume we solved the Riemann problem in the cells I_{i-1} and I_i. Bottom: Godunov step for a linear advection equation ($f(u) := a \cdot u$ with $a > 0$).

- *Transmissive boundary condition.*
  $W_0^n = W_1^n, W_{-1}^n = W_2^n,$
  $W_{M+1}^n = W_M^n, W_{M+2}^n = W_{M-1}^n,$
  with W the vector of conserved variables (e.g. energy, mass)

When solving a set of partial differential equations numerically, we often have to explicitly integrate over a certain time interval Δt to obtain a solution. The CFL-condition is a condition for the convergence and is given by

$$\Delta t = C_{CFL} \cdot \frac{\Delta x}{S_{max}^{(n)}}, \tag{32}$$

with $C_{CFL} \in (0,\ 1]$ the CFL coefficient,

$$S_{max}^{(n)} = max_i\{|u_i^n| + a_i^n\}. \tag{33}$$

Note, that the i-range depends on the chosen boundary conditions and thus inappropriate choices of $S_{max}^{(n)}$ make the Godunov scheme unstable.
Alternatively, we can write

$$c := \sum_i \frac{U_i \Delta t}{\Delta x_i} \leq c_{max}, \tag{34}$$

with c the Courant-number and U the dependent variable.

2.6 Numerical Methods DONE

As the main goal of this work is to add a tracer fluid to the JUPITER code, we first discuss in the following the two most commonly used numerical methods to solve the equations (11)-(13) with their advantages and disadvantages: the *Smoothed Particle Hydrodynamics* and the *Adaptive Mesh Refinement*. In both methods, the underlying idea is

to divide the problem into smaller sub-problems by discretising certain quantities. We then discuss in more detail how the JUPITER code solves the Euler equations in the radiative module.

2.6.1 Smoothed Particle Hydrodynamics (SPH) DONE

The first method is the *smoothed particle hydrodynamics (SPH)* scheme, which was developed by R. Gingold and J. Monaghan ([62]) and L. Lucy ([63]). It is a particle-based Lagrangian-like method. Starting from the equations of motion (EoM), this scheme makes use of statistical techniques to recover analytical expressions for the physical variables from an initial distribution of N fluid elements with volume Δv_j and density ρ_j,

$$\rho_j \Delta v_j \frac{d^2\vec{r}_j}{dt^2} = -\Delta v_j \nabla P + \rho_j \Delta v_j \vec{F}_j, \tag{35}$$

where $\vec{F}_j$ is the total body force acting on the j^{th} fluid element, ∇P is the pressure gradient at $\vec{r}_j$, the CoM. In the limit of vanishing Δv_j, equation (34) can be written as

$$\frac{d^2\vec{r}_j}{dt^2} = -\frac{1}{\rho_j}\nabla P + \vec{F}_j. \tag{36}$$

Starting from the assumption that the positions of these point-like fluid elements[23] are distributed randomly according to the density $\forall t$[24], two techniques are suitable to then determine the probability distribution of this sample: the *smoothing kernel method* [64] and the *delta spline method* [65], which both use the Monte Carlo method.
In this approach, the tracer particles do not mix with the simulated fluid, which makes tracking the particles' individual evolutions and trajectories possible, [66]. On the other hand, the EoMs (35) is not integrated correctly, since the Monte Carlo method only estimates the mass density ρ.

2.6.2 Adaptive Mesh Refinement (AMR) DONE

The second method we discuss is called *adaptive mesh refinement* (AMR), which is an Eulerian-like method that allows for a dynamical adjustment of the resolution level. Instead of discrete physical quantities as in the previous method, the AMR approach discretises the volume, i.e. creates a grid of rectangular cells. For more detail on the description of the grid, the reader is referred to [67] - we briefly summarize the main aspects in the following.
A finite area D (e.g. the CSD) is divided into a grid with cells $G_{i,k}$ for refinement levels $i \in \{1,...,i_{max}\}$ such that

$$G_i = \cup_k G_{i,k}. \tag{37}$$

With increasing refinement level, we require that the fine grid's boundaries match the coarser one, see figure 18

The code then solves equations (11)-(13) in each cell with boundary conditions given by the solution of all adjacent cells. The major advantage of this approach is that the level of resolution can by dynamically adjusted: regions of interest can be simulated with a finer grid while the surrounding environment, that might only act as a reservoir is simulated with the initial (coarse) grid. This method allows for a more cost and time efficient simulation. Due to the interaction between the cells, i.e. the exchange of energy, mass and angular momentum according to the equations (11)-(13), it becomes impossible to track the evolution and trajectories of the fluid elements: the information about the past evolution of the fluid is lost due to this interaction with the neighbouring cells. Another advantage of this method is the correct tracking of the mass flow.

Because of the advantages of the AMR method (time and cost efficiency), a method to bypass the aforementioned problem of information loss has been elaborated. The solution is to add another fluid to the simulation, which does not actively take part in the physics, i.e. it does not exert any additional force field on the other fluids, but traces the motion and flow of the material. We call it therefore *tracer fluid*, [66] and highlight again the difference between these two types of fluids:
for the physical fluids (such as gas) in the simulations, we discretise their physical quantities, while for the tracer fluid, the space is discretised and only the equation describing the advection is solved.

[23] i.e. in the SPH method, physical quantities of the fluid, such as energy and mass, are discretised

[24] This assumption arises from the fact that for large N-body systems, the fluid elements follow a complicated motion and can thus be treated as randomly distributed. This assumption does not necessarily hold, if a certain underlying symmetry is assumed.

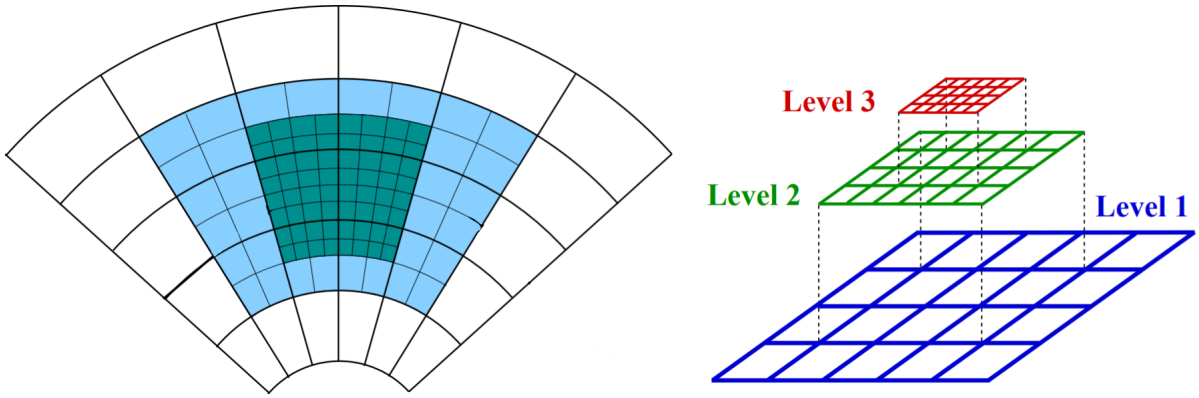

Figure 18: Examples for a nested grid. Top: in a spherical co-ordinate system, [45]. Bottom: in a Carthesian co-ordinate system, [68]. The advantage of this technique is that certain regions in the simulated grid can be studied in greater detail, without wasting computational resources and time by constructing a fine grid over the complete simulation volume.

2.7 JUPITER's Solving System DONE

In the following, we give a more detailed explanation of the underlying solving system that the JUPITER code makes use of, including the treatment of the different levels of refinement of the grid. This Section is based on Section 3 of [45][25], who make use of the AMR technique, which we describe further in Section 2.6.2. For further details, we refer the reader to the aforementioned reference. This method is very common in (astrophysical) studies, for example, in the simulations performed by [69], [70] and [71], a similar scheme has been used to study various astrophysical phenomena, such as the collapse of a proto-stellar cloud or collisions between stars.

As described above, the JUPITER code uses a nested grid. The advantage of this approach is that the regions of greater interest can be simulated with a higher resolution, while the surrounding environment remains on the coarse initial resolution. Thereby, the simulation is not only more time- but also more cost-efficient than if we would simulate the whole grid on a fine level. In our case, we simulate the CSD on a coarse level of refinement, while we choose a higher resolution for the CPD, with increasing level of refinement towards the planet[26]. Figure 19 illustrates the underlying cycle that is used in order to solve the Euler equations (11)-(13) among the different levels of refinement (the graphic assumes three levels for simplicity). We denote with $\Delta t_i(l)$ the time step i on the grid level l.

Each new cycle (over the whole grid and all refinement levels) starts from the finest grid level, as can be seen in figure 21, and thus the time step from that level determines the CFL condition for the whole grid during this cycle. The first step is to integrate the relevant equations on this level and then check whether the new CFL condition is satisfied. In the next step, the finest level is evolved one time step further, followed by the first evolution (i.e. integration of the governing equations) of the next coarser grid level. At this point, the finest level has evolved two time steps, while the next coarser has only evolved for one time step[27]. A first communication between these two levels has to take place as well. We replace the solution obtained on the coarser level, with the one obtained on the finer grid in the corresponding cells and update the boundary conditions on the finer level accordingly, because the coarser level covers more area.

Note that between each grid level $l+1$ and the next coarser level l are so called *ghost-cells*[28], that contain the boundary conditions, which are used when two successive grid levels communicate with each other. In figure 20, the ghost cells are coloured in light blue, while the grids levels are in dark blue and white.

We highlight, that the CFL stability condition (see Section 2.5) must be fulfilled during each integration because successive grid levels must communicate with each other and the Euler equations are solved independently in this

[25] Note, that the aim of their paper was such, that they did not need to solve the equation of energy conservation (13). In the JUPITER code, the system of equations (11)-(13) is solved, as described in Section 2.2 and [2]. We further highlight that the discussion of [45] is based on an isothermal CSD, while the JUPITER code can simulate purely isothermal CSDs or include adiabatic effects. In this work here, we work only in the adiabatic version of the code.

[26] Or more general: around the area of interest

[27] A general formula to deduce the number of integrations performed on a grid level l is given by 2^{l-1}, see [45].

[28] The JUPITER code uses two ghost cells in each direction.

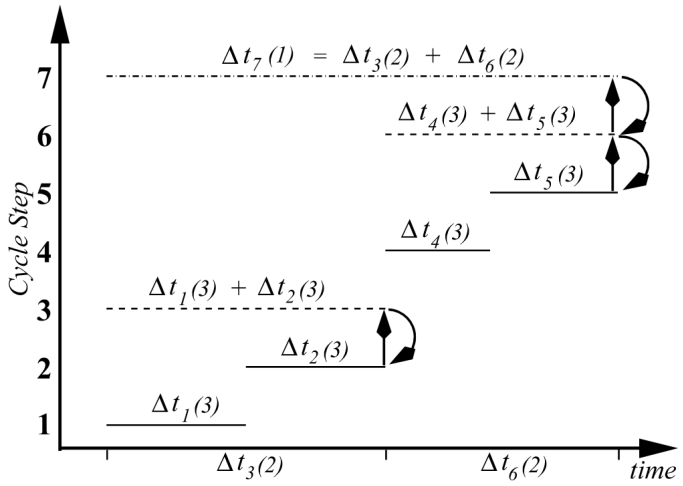

Figure 19: An illustration of the solving scheme for a grid with three levels of refinement, [45].

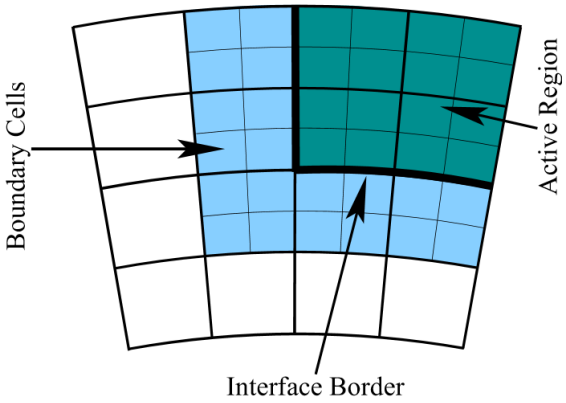

Figure 20: Schematic illustration of the ghost cells (light blue), containing the boundary conditions. They are used in the communication step between two adjacent grid levels (white and dark blue), see [45].

scheme on each level of refinement.

In figure 21, solving scheme of JUPITER is shown in a similar way. We solve the system of equations (11)-(13) on the finest level of refinement first and use the so obtained solutions as initial conditions for the next coarser level, etc. This step is called *restriction*. To do so, the vectors on the fine grid level are successively projected onto the next coarser grid level. The full solution is finally determined on the coarsest level. In figure 21 this is marked by the green dot at the bottom of the V. In the following, the result is successively *prolongated* to the finest resolution level by interpolating the vectors from the coarse grid to the finer ones, where the final solution is then determined.

To summarize, JUPITER's general approach to solving the equations (11) - (13) is by using a high order Godunov scheme (see Section 2.4), which is a conservative numerical scheme to solve partial differential equations. It makes use of the solution of the Riemann problem, that is defined at each inter-cell boundary (see figure 20) with piecewise constant initial data, [73]. Within the Godunov scheme, the above explained method, pictured in figures 19 and 21 is used.

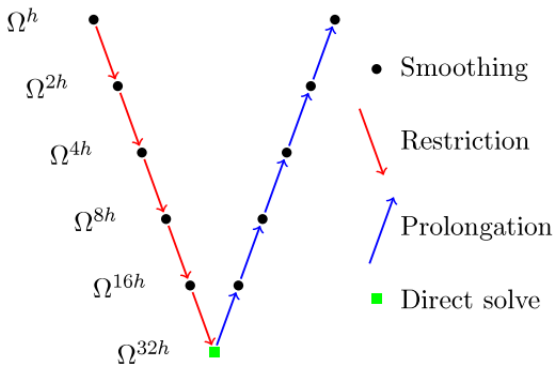

Figure 21: Schematic illustration of the V-cycle, the underlying scheme the JUPITER code uses to solve equations (11)-(13) on a nested grid, [72]. In this example, the grid has six different levels of refinement. The multi-grid method consists of *smoothing*, i.e. an iterative relaxation scheme, a *restriction* step, during which the vectors from the finer grid are projected onto the next coarser one, the *prolongation* step, during which the coarse-grid vectors are interpolated onto the next finer grid level.

2.8 Solving the Equations of Hydrodynamics for the Radiative Module

2.8.1 The Energy Equation DONE

As mentioned in Section 2.2, in the radiative (gas-only[29]) module, the JUPITER code solves equation (14) together with (17):

$$\frac{\partial E_{gas}}{\partial t} + \nabla\left((E + Pid - \overline{\tau})\cdot\vec{v}\right) = -\rho\vec{v}\cdot\nabla\Phi - \rho\kappa_R c\left(\frac{B_\nu(T)}{c} - \varepsilon_{rad}\right) + S, \tag{38}$$

$$\frac{\partial \varepsilon_{rad}}{\partial t} = -\nabla F_{rad} + \rho\kappa_R c\left(\frac{B_\nu(T)}{c} - \varepsilon_{rad}\right). \tag{39}$$

The main goal of this Section is to derive the matrix form of equation (37) that the JUPITER code then can solve. As a result, we find the temperature field in the system.
The Riemann solver (see Section 2.3) in the JUPITER code solves the Euler equations (11)-(13) (or (14), (37), respectively) first without the source terms and adds them later in the HydroKernel function[30], i.e. we first solve the following system of equations:

$$\varepsilon = E - \rho\frac{\vec{v}^2}{2}, \tag{40}$$

$$\frac{\partial E}{\partial t} = -\rho\kappa_R c\left(\frac{B_\nu(T)}{c} - \varepsilon_{rad}\right), \tag{41}$$

$$\frac{\partial \varepsilon_{rad}}{\partial t} = -\nabla F_{rad} + \rho\kappa_R c\left(\frac{B_\nu(T)}{c} - \varepsilon_{rad}\right). \tag{42}$$

To do so, we follow the method described in Section 3.4. in [74]:

$$\frac{\varepsilon_{rad}^{n+1} - \varepsilon_{rad}^{n}}{\Delta t} = \rho^n\kappa_P^n\cdot(4\sigma(T^{n+1})^4 - c\varepsilon_{rad}^{n+1}) + \nabla\frac{c\lambda^n}{\kappa_R\rho^n}\nabla\varepsilon_{rad}^{n+1} \tag{43}$$

with

$$(T^{n+1})^4 \simeq 4(T^n)^3T^{n+1} - 3(T^n)^4. \tag{44}$$

The index n and $n+1$ refer to the variable before and after the implicit update, i.e. before and after the time step.

[29] $E_{gas} := \varepsilon + \rho\frac{\vec{v}^2}{2}$
[30] Some authors refer to this method as *operator splitting*.

Using the notation of [2] (see p. 60), we write the last term of equation (42) explicitly

$$\begin{aligned}\nabla\frac{c\lambda^n}{\kappa_R\rho^n}\nabla\varepsilon_{rad}^{n+1} &= \frac{1}{\Delta x}\left(\overline{D}^x_{i+1,j,k}\frac{\varepsilon^{n+1}_{rad,i+1,j,k}-\varepsilon^{n+1}_{rad,i,j,k}}{\Delta x}-\overline{D}^x_{i,j,k}\frac{\varepsilon^{n+1}_{rad,i,j,k}-\varepsilon^{n+1}_{rad,i-1,j,k}}{\Delta x}\right)\\ &+ \frac{1}{\Delta y}\left(\overline{D}^y_{i,j+1,k}\frac{\varepsilon^{n+1}_{rad,i,j+1,k}-\varepsilon^{n+1}_{rad,i,j,k}}{\Delta y}-\overline{D}^y_{i,j,k}\frac{\varepsilon^{n+1}_{rad,i,j,k}-\varepsilon^{n+1}_{rad,i,j-1,k}}{\Delta y}\right)\\ &+ \frac{1}{\Delta z}\left(\overline{D}^z_{i,j,k+1}\frac{\varepsilon^{n+1}_{rad,i,j,k+1}-\varepsilon^{n+1}_{rad,i,j,k}}{\Delta z}-\overline{D}^z_{i,j,k}\frac{\varepsilon^{n+1}_{rad,i,j,k}-\varepsilon^{n+1}_{rad,i,j,k-1}}{\Delta z}\right).\end{aligned}$$

The indices $i,\ j,\ k$ number the cells in $x,\ y,\ z$ direction, respectively. Note, that here the overline denotes the average (in equation (15), we have used the overline to highlight that $\overline{D}$ is a tensor quantity),

$$\overline{D}^x_{i,j,k} = \frac{1}{2}(D_{i,j,k}+D_{i-1,j,k}), \tag{45}$$

$$\overline{D}^y_{i,j,k} = \frac{1}{2}(D_{i,j,k}+D_{i,j-1,k}), \tag{46}$$

$$\overline{D}^z_{i,j,k} = \frac{1}{2}(D_{i,j,k}+D_{i,j,k-1}), \tag{47}$$

$$\text{with } D_{i,j,k} = \frac{\lambda c}{\rho_{i,j,k}\kappa_{R;i,j,k}}. \tag{48}$$

We now discuss the term in the brackets in equation (42). By adding the stellar heating S and the advection & viscous heating effects (which we express as Q^+ according to [57]), we arrive at

$$\frac{\partial\varepsilon}{\partial t} = -\rho\kappa_R c\left(\frac{B_\nu(T)}{c}-\varepsilon_{rad}\right)+S+Q^+ \tag{49}$$

In the adiabatic module, we have $\varepsilon := \rho c_V T$, which by inserting into the LHS of equation (48) and dividing by ρc_V yields

$$\frac{T^{n+1}-T^n}{\Delta t} = -\frac{\kappa_R}{c_V}(4\sigma(T^{n+1})^4 - c\varepsilon^{n+1}_{rad})+\frac{S}{\rho c_V}+\frac{Q^+}{\rho c_V}.$$

with the solution

$$T^{n+1} = \eta_1+\eta_2\varepsilon^{n+1}_{rad}, \tag{50}$$

$$\text{where } \eta_1 := \frac{T^n+12\Delta t\frac{\kappa_R}{c_V}\sigma\cdot(T^n)^4+\frac{\Delta t S}{\rho c_V}+\frac{\Delta t Q^+}{\rho c_V}}{1+16\Delta t\frac{\kappa_R}{c_V}\sigma\cdot(T^n)^3}, \tag{51}$$

$$\text{and } \eta_2 := \frac{c\Delta t\frac{\kappa_R}{c_V}}{1+16\Delta t\frac{\kappa_R}{c_V}\sigma\cdot(T^n)^3}. \tag{52}$$

Finally, we can rewrite the differential equation for the radiation energy (42) in matrix form:

$$\begin{aligned}R_{i,j,k} = {}&\beta_{1;i,j,k}\varepsilon^{n+1}_{rad;i+1,j,k}+\beta_{2;i,j,k}\varepsilon^{n+1}_{rad;i-1,j,k}\\ &+\beta_{3;i,j,k}\varepsilon^{n+1}_{rad;i,j+1,k}+\beta_{4;i,j,k}\varepsilon^{n+1}_{rad;i,j-1,k}\\ &+\beta_{5;i,j,k}\varepsilon^{n+1}_{rad;i,j,k+1}+\beta_{6;i,j,k}\varepsilon^{n+1}_{rad;i,j,k-1}\\ &+\Gamma_{i,j,k}\varepsilon^{n+1}_{rad;i,j,k},\end{aligned} \tag{53}$$

with coefficients[31] (see [2]):

[31] The diffusion coefficients $\overline{D}$ are given in equation (47).

$$
\begin{aligned}
\beta_{1;i,j,k} &= -\frac{\Delta t}{\Delta x^2}\overline{D}^x_{i+1,j,k} \\
\beta_{2;i,j,k} &= -\frac{\Delta t}{\Delta x^2}\overline{D}^x_{i-1,j,k} \\
\beta_{3;i,j,k} &= -\frac{\Delta t}{\Delta y^2}\overline{D}^y_{i,j+1,k} \\
\beta_{4;i,j,k} &= -\frac{\Delta t}{\Delta y^2}\overline{D}^y_{i,j-1,k} \\
\beta_{5;i,j,k} &= -\frac{\Delta t}{\Delta z^2}\overline{D}^z_{i,j,k+1} \\
\beta_{6;i,j,k} &= -\frac{\Delta t}{\Delta z^2}\overline{D}^z_{i,j,k-1} \\
\beta_{1-6} &:= -\sum_{m=1}^{6}\beta_m \\
\Gamma_{i,j,k} &= (1 + c\rho\kappa_R\Delta t - 16\Delta_t\rho\kappa_R\sigma\cdot(T^n)^3\cdot\eta_2) + \beta_{1-6} \\
R_{i,j,k} &= 16\Delta_t\rho\kappa_R\sigma\cdot(T^n)^3\cdot\eta_1 - 12\Delta_t\rho\kappa_R\sigma\cdot(T^n)^4 + \varepsilon^n_{rad}
\end{aligned}
$$

2.8.2 The Mass Equation DONE

This Section is based on [60][32].
For the conservation equation of mass in its differential form,

$$\frac{\partial\rho}{\partial t} + \frac{\partial(\rho v)}{\partial x} = 0, \tag{54}$$

to be solved, we require that the velocity of the medium $v = v(x,t)$ is either known a priori or $v = v(\rho(x,t))$. In the latter case, the second term of equation (53) can be formulated as a function of the density only, $\rho v = f(\rho)$ and therefore

$$\frac{\partial\rho}{\partial t} + \frac{\partial f(\rho)}{\partial x} = 0. \tag{55}$$

In the first case, where v is a priori known, we have $\rho v(x,t) = f(\rho,x,t)$. For a constant velocity, $v(x,t) = const$, the mass conservation equation reads

$$\frac{\partial\rho}{\partial t} + const\cdot\frac{\partial\rho}{\partial x} = 0. \tag{56}$$

This equation is called *linear advection equation*. Together with the initial condition $\rho(x,t=0) := \rho_0(x)$ for $x\in\mathbb{R}$, its solution is given by $\rho(x,t) = \rho_0(x - const\cdot t)$.
Note, that in general the solution is more complicated, since we solve not only the mass continuity equation, but instead solve the Euler equation (11) - (13).
In equation (53), we have assumed a vanishing flux. A more general expression is given by

$$\frac{\partial\rho}{\partial t} + \frac{\partial(\rho v)}{\partial x} = D\frac{\partial^2\rho}{\partial x^2}, \tag{57}$$

with D a (constant) diffusion coefficient. Accordingly, equation (56) is called *advection-diffusion equation*.

2.8.3 The Angular Momentum Equation

TODO In order to obtain the velocity field in the system, we solve the angular momentum equation

$$\frac{\partial(\rho\vec{v})}{\partial t} + \nabla\cdot(\rho\vec{v}\otimes\vec{v} + P\cdot id) = \text{source terms}.$$

[32] Sections 2.1 - 2.3

3 THE TRACERS DONE

In Section 2, we have discussed the governing equations and in detail the radiative module of the JUPITER code as well as the solving scheme that is implemented. Hereafter, we first describe two possible methods to introduce a tracer fluid to a hydrodynamical simulation, namely the *Monte Carlo Tracer Fluid* and the *Velocity Tracer Fluid*, see Sections 3.1 and 3.2 respectively. In Section 3.3, we describe the method that has been chosen for this work: we introduce massless tracer particles, based on the paper [28] by Lissauer et al. We give detailed insight on the thought process during this work in Sections 3.3.1 and 3.3.2, and the different functions that we have written, Sections 3.3.4 - 3.3.11. The communication between multiple CPUs and multiple levels of refinement are discussed in Section 3.5 and 3.6, respectively. In Section 3.4 we discuss how we implemented massive tracers and what adjustments have been done to the functions for massless tracers. Figure 29 shows an overview of the whole implementation, including the most relevant (but not all) functions.

For the implementation that we choose, we require that the code runs without the functions or parts of functions related to the tracers as well. To ensure this condition is satisfied throughout the code, we insert additional conditions `if(TracerParticle == YES)`, where `TracerParticle` is a global boolean variable with default value `NO`. Furthermore, all functions that are written throughout this Master thesis are named `Tracer...()`, such that the user can easily determine which functions belong to the tracer implementation.

In Section 3.3 and 3.4 we show a few of the functions that were written throughout this work. The irrelevant parts are commented out with

`/* [...] */`.

As a final note before starting with the detailed discussion on tracer particles, we mention that certain functions, pointers etc. might be redundant or might not be synchronized. This is mainly due to the fact that the implementation became very quickly rather large[33] with more than 20 functions and special cases that had to be taken care of. At multiple occasions, we had to restructure the code because a problem could not be solved otherwise, or a special case that is nevertheless very relevant was forgotten in the previous development. On the other hand, the given task turned out to be much trickier than one might think when hearing *Add a tracer fluid to the* JUPITER *code.* Finally, the time for a Master thesis is rather short and I spent in total roughly two months just to understand the code[34]. It turned out that I had to understand 90% of the 260+ files in order to implement the tracers. It was therefore not possible to finish cleaning up this part at the end.

3.1 The Monte-Carlo Tracer Fluid DONE

Hereafter, we briefly discuss the implementation of a Monte-Carlo tracer fluid, which has to advantage of tracing the mass flow of the fluids present in the CSD (such as gas and dust) correctly. In this work, we will however use the discrete implementation of tracer particles, see 3.3. This Subsection is based on [66].
One method to trace the Lagrangian history of a cell is to assign a number of tracer particle to each cell, such that they follow the mass flow of that parent cell. We define the initial mass of cell i in the grid as M_i. Solving the governing equations, tracer particles between adjacent cells are exchanged. We define the exchange of mass between cell i and j as $\Delta M_{i,j}$[35]. At each time step, a list is created, containing the tracer particles that were initially in cell i as well as its reduced mass $M_i^{'}$, which is defined as $M_i^{'} := M_i - \Delta M_{i,j}$. By doing so, we keep track of the out-flux of mass. Given this set up,

$$P_{i,j}^{flux} = \frac{\Delta M_{i,j}}{M_i^{'}}$$

describes the probability for each tracer particle in cell i to move to cell j.
In the next step, we decide whether a certain tracer leaves cell i or not. Drawing a number $\alpha \in U(0,\ 1)$. A tracer moves form i to j, if $\alpha < P_{i,j}^{flux}$. We perform this for all cells in the grid (or fluid patch) and update the masses accordingly. We highlight that this method only has to take care of the out-flowing mass, since a mass influx is taken care of as mass out-flux of the neighbouring cells.
Note that with this method, the mass flow is correctly traced per construction. When using the Velocity Tracer

[33] almost 2000 lines of code, including some comments
[34] It certainly would have taken me even longer, if it was not for Dr. Ameya Prabhu's help.
[35] In this notation, mass flows from cell i to cell j. $\Delta M_{j,i}$ defines then the mass-flux from cell j to i.

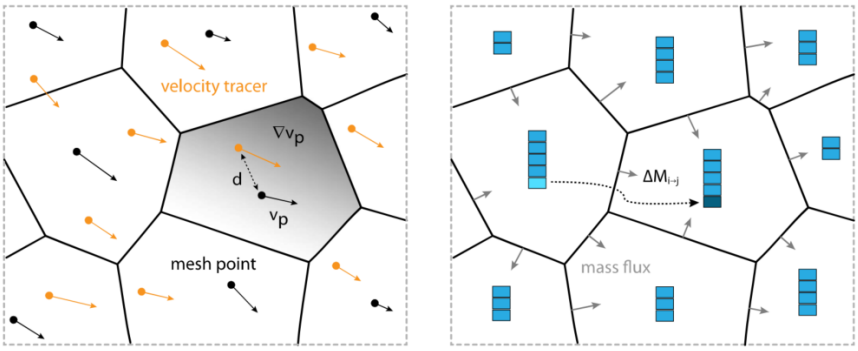

Figure 22: Schematic illustration of the *Velocity tracer fluid* (left) method and the *Monte Carlo tracer fluid* method (right), [66]. While the former one tracks the position and velocity of the fluid cell, the latter method traces the mass exchange between adjacent grid cells and thus the mass flow is correctly displayed.

Particle method, the mass flow is not necessarily correctly tracked. However, since these tracer particles are not assigned to certain spatial co-ordinates, we cannot track their positions and velocities. The tracers however are all in the global and parallelized linked list with an identity number, where we further keep track of the fluid properties as well as a pointer to the next tracer and one to the previous tracer within the same cell. Each cell requires then only one additional pointer to be accessed.
Figure 22 shows a schematic illustration of these the Monte-Carlo method (right) as well as the Velocity tracer fluid (left).

3.2 The Velocity Tracer Fluid DONE

Hereafter, we give an overview of the implementation of the velocity tracer fluid according to the scheme presented in [66]. This method has to advantage to correctly tracing the Lagrangian history of the fluid parcels' trajectories. However, as opposed to the Monte-Carlo tracers, the mass flow is not correctly visualized.
The velocity tracer fluid consists of massless particles for which we only solve the equation of advection (57). For these particles, we use an estimate of the underlying fluid (e.g. gas) at the position of each tracer particle. To do so, the two most common methods are the cloud-in-cell (CIC) interpolation scheme, see [75], or using the velocity of the nearest cell[36] as an approximation. To not further overcomplicate the matter, we use the latter method in this work but for completeness, we give a brief summary on the CIC method in Section 3.2.1.

3.2.1 The Cloud-In-Cell (CIC) method

A simplified description of the CIC-method can be found in [76], and we briefly summarize the most important aspects hereafter following that paper.
For simplicity, consider a 2-dimensional finite mesh, see figure 23, with nodes at positions $(x_i,\ y_i)$ (Cartesian co-ordinates); the index i spans the number of cells in our mesh. The extension to 3 space dimensions should be clear. We want to determine a certain (physical) quantity, denoted by $n_{i,j}$, at positions $(x,\ y)$ within the cells, i.e. $x_i < x < x_{i+1}$ and $y_i < y < y_{i+1}$. We then have

$$\begin{aligned} n_{i,j} &= \frac{S_{i,j}}{A^2_{i,j}} \cdot (x_{i+1}-x)(y_{j+1}-y), \\ n_{i+1,j} &= \frac{S_{i+1,j}}{A^2_{i+1,j}} \cdot (x-x_i)(y_{j+1}-y), \\ n_{i,j+1} &= \frac{S_{i,j+1}}{A^2_{i,j+1}} \cdot (x_{i+1}-x)(y_{j+1}-y), \\ n_{i+1,j+1} &= \frac{S_{i+1,j+1}}{A^2_{i+1,j+1}} \cdot (x-x_i)(y-y_i), \end{aligned}$$

[36] Or more precise: the velocity at the centre of the cell in which the tracer is, in case of the JUPITER code.

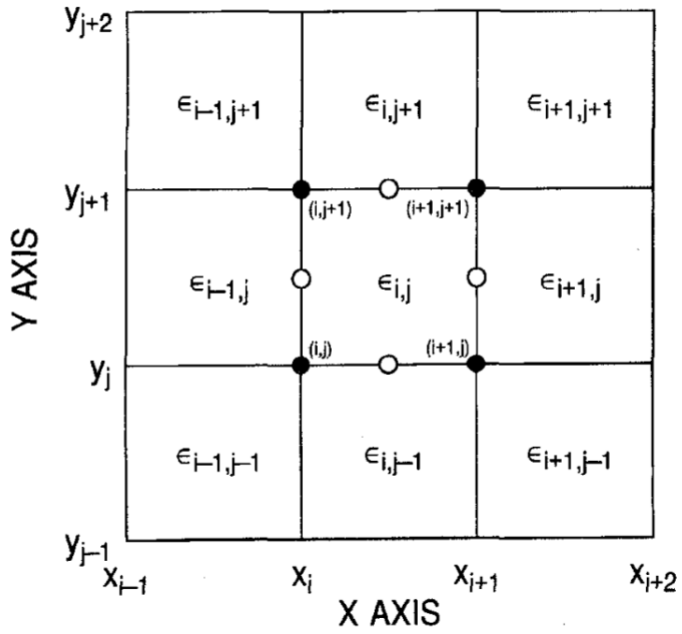

Figure 23: Example setup for a 2-dimensional mesh used in the CIC-method used in the velocity tracer fluid method, [75].

where $S_{i,j}$ are the statistical weights of the particles at $(x_i,\ y_j)$ and $A_{i,j} = \Delta x_i \Delta y_j$, with $\Delta(x,y)$ the constant spatial step in the corresponding direction. Note, that the value of $n_{i,j}$ depends on the cell (i,j) *and* the adjacent cells.

In the next step, the tracer particles' positions are updated by using an Euler method based on the hydrodynamical time-step dt and determine the parent cell of each tracer particle[37]. This is done by introducing an initial search radius that is based on the distance to the nearest cell in the previous time-step. Having determined the parent cells for all tracer particles, we can move them according to the linearly interpolated velocity field of their parent cells. In the following, we determine the gradient of the velocity field of the parent cells using the Gauss theorem, [77].

Theorem 2 (Gauss theorem) *Let $D \subset \mathbb{R}^n$ be an open set with boundary ∂D, for which at each point there exists a normal vector $\vec{n}$, such that*

- $|\vec{n}| = 1$
- *$\vec{n}$ is orthogonal to ∂D*
- *$\vec{n}$ points outwards from ∂D.*

Let F be a vector field in C^1, defined at least on the closure of D. We define the divergence

$$divF := \sum_{i=1}^{N} \frac{\partial F_i}{\partial x_i}.$$

Then,

$$\int_D divF dV_N = \int_{\partial D} F \cdot \vec{n} dV_{N-1},$$

where dV_N, dV_{N-1} denote the N and $N-1$ dimensional measure, respectively.

[37] For each cell in the mesh, we introduce one tracer particle, that lies within that cell, which we call parent cell.

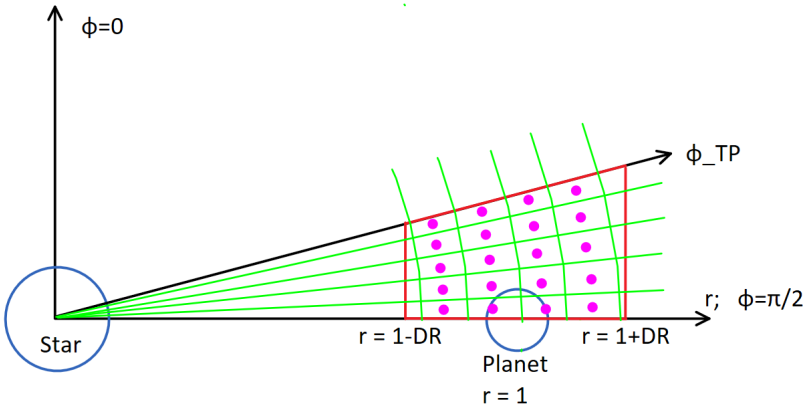

Figure 24: Illustration of the initial position of the tracer particles (in purple), not to scale for an arbitrary plane θ. Each cell with its centre being in the volume $[r_P - r_{TP},\ r_P + r_{TP}]$, colatitude $[\phi_{TP},\ \pi/2]$ and azimuthal angle θ in $[0,\ 2\pi]$ obtains a tracer particle, [78]. For simplicity, we draw in green only one level of mesh refinement. Only if the centre of the cell lies within the pre-defined red volume, we place a tracer particle in it.

Finally, we use the velocity gradient field to interpolate the velocity field at the tracer particles' positions. In order to record the history of a tracer particle, we can store certain properties of interest in an array that is updated in each time-step and written into a separate snapshot file. The values are then zeroed and the above process is repeated for the next time-step.

3.3 The Massless Tracer Particles DONE

To introduce massless tracer particles into the JUPITER code, the particles have to be advected with the gas fluid, which means

$$\frac{d\vec{x}}{dt} = \vec{v}, \tag{58}$$

has to be solved. In this work, the tracer particles are initially placed at the centre of the grid cells that are in close vicinity to the planet's orbit, such that the meridional circulation 1.6 can be visualized. We choose all cells within a certain volume around the planet's orbit, which can be described by

$$az_c \in [0,\ 2\pi], \tag{59}$$

$$r_c \in \left[R_P - \frac{R_P}{c_1},\ R_P + \frac{R_P}{c_1}\right], \tag{60}$$

$$col_c \in \left[COL_P - \frac{COL_P}{c_2},\ \pi/2\right], \tag{61}$$

where az_c, r_c, col_c are the co-ordinates of the centre of a grid cell, $c_{1,2}$ are constants, R_P is the planet's radius, which is at a fixed value of 1.0 in code units, and COL_P the planet's colatitude, fixed at a value of $\pi/2$. Figure 24 gives an illustration of the situation. For simplicity, only one level of grid refinement is drawn. The green lines define the boundaries of the grid cells and the magenta points highlight the cell centres which satisfy the conditions (58) - (60) and thus contain a tracer at the start of the simulation. In Section 3.3.4, the function that determines the total number of such cells, and equivalent the number of tracer particles in this volume, is introduced. However, we have to pay special attention to the choice of the constants $c_{1,2}$: if the number of grid cells is small, i.e. the cells are large, and $c_{1,2}$ are large as well, the resulting torus-like volume in which we intend to place the tracers will become too small. In this case it might occur that no cell centre lies in the defined volume and therefore no tracer particle is introduced[38]. The solution of this problem is discussed in Section 3.3.4.

With this choice, the initial positions of the tracer particles are at the centre of the cells, because at this location, the values of the physical quantities, such as energy, density and velocity, of the gas fluid are easily accessible in

[38]This is a very unlikely scenario, since the cells are usually small compared to the tracer volume (58) - (60) but we nevertheless have to ensure that this case never occurs.

the JUPITER code. This approach is similar to what has been used in [28], with the exception that their initial distribution of tracer particles was generated randomly (but within a certain radius from the planet) and in the JUPITER code, only the upper half of the disk is simulated, assuming symmetry with respect to the disk mid-plane. For all other positions in the grid cells (i.e. positions of tracer particles at later time steps), we have to interpolate these values using one of the following methods, see [2]:

- *GFO*: piece-wise constant method. It treats the error only to first order in space and thus only used for testing purposes.
- *PLM*: piece-wise linear method. It is second order accurate in space (i.e. errors are on the order of the square of the zone size) and used in isothermal simulations.
- *MUSCL*: Is second order accurate in space as well and used in adiabatic simulations.

We make a pointer `propertyarray[NrTPs][2][6]`, where `NrTPs` is the number of cells that lie in the volume shown in figure 24. The 2 refers to *pre-* and *post*-updated values and 6 refers to the three space co-ordinates (indices 0-2) and velocity components (indices 3-5). The initial conditions are therefore stored in the first step in `propertyarray[:][0][:]`.

Similar to the integration method used in [28], we update the positions of the tracer particles using a second-order Runge-Kutta method, which is described in the following. Further details on the algorithm can be found in every book on numerical methods, for example [79].

Theorem 3 (4^{nd}**-order Runge-Kutta algorithm)** *For a given differential equation of the form*

$$\frac{dx}{dt} = f(x,y), \tag{62}$$

with initial conditions $x(0) = x_0$, the solution at time $t = t_{n+1}$ is approximated by

$$x_{n+1} = x_n + \frac{h}{6}\cdot(k_1 + 2k_2 + 2k_3 + k_4), \tag{63}$$

see figure 25[39] for illustration, where the terms k_1 - k_4 are given by

$$\begin{aligned} k_1 &= f(t_n,\ x_n), && (64)\\ k_2 &= f\left(t_n + \frac{h}{2},\ x_n + k_1\frac{h}{2}\right), && (65)\\ k_3 &= f\left(t_n + \frac{h}{2},\ x_n + k_2\frac{h}{2}\right), && (66)\\ k_4 &= f(t_n + h,\ x_n + h\cdot k_3), && (67) \end{aligned}$$

and step-size $h > 0$.

For the 2^{nd} order Runge-Kutta algorithm, we use only k_1 and k_4 and an adjusted coefficient,

$$x_{n+1} = x_n + \frac{h}{2}\cdot(k_1 + k_4). \tag{68}$$

In our implementation of the tracer particles in the JUPITER code, we have $h = dt$, representing the time-step, $k_1 = v_n$ is the pre-update velocity and $k_4 = v_{n+1}$ is the updated velocity. Using the second order equation, we therefore set $k_2 = k_3 = 0$.
After the position update of the tracer particles, the velocities at the new positions have to be determined to finish

[39] The figure is for a fourth order Runge-Kutta algorithm.

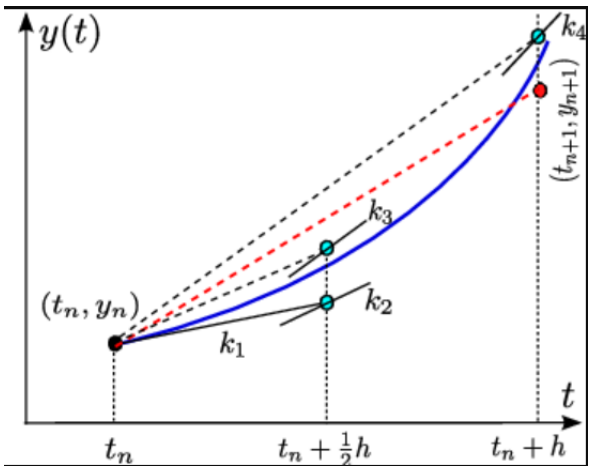

Figure 25: Illustration of the coefficients in the fourth-order Runge-Kutta algorithm, [80].

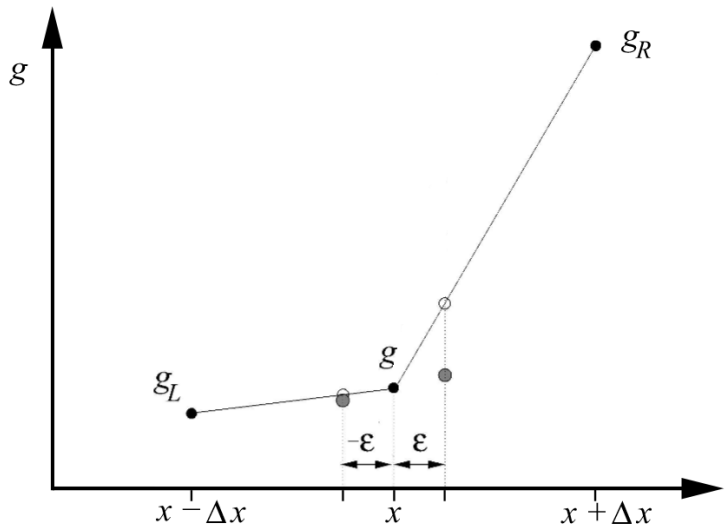

Figure 26: Monotonized harmonic mean method to update the velocities of the tracer particles after the position update. The filled circles denote the values at $x \pm \varepsilon$ in the case of a monotonized harmonic mean. In contrast to this, the empty circles are obtained by a monotonized geometric mean (which is not described in this work further), [45].

the integration (57). To do so, we have to take into account that in the surroundings of a forming planet, large velocity gradients can occur due to shocks and other mechanisms from the meridional circulation, see discussion in Section 1.6. Following [28], we perform the interpolation with a *monotonized harmonic mean method*, which is thoroughly described in Section 3.1.3. in [45][40] and is second-order accurate in space and time.

Theorem 4 (Monotonized harmonic mean method) *(From [45]) For a given function with values v_L, v and v_R at positions $x - \Delta x$, x and $x + \Delta x$, the function's averaged value at $x + \varepsilon$ is given by*

$$v_\varepsilon = v + \frac{2\varepsilon}{\Delta x} \cdot max\left(\frac{(v - v_L)(v_R - v)}{v_R - v_L}, 0 \right), \tag{69}$$

for $\varepsilon \in \left[-\frac{\Delta x}{2}, \frac{\Delta x}{2}\right]$. Figure 26 illustrates the situation.

In Section 3.3.1 we discuss how we implemented equation (68) in the code.

[40] This method was previously described in [81].

3.3.1 Attempt 1: Advecting the Tracers DONE

In equation (68) we have found a general formula to update the velocity of a tracer particle. In the case of our 3-dimensional simulation, we have to take all spatial directions into account, when updating the velocities (and positions) of the tracer particles. In a first attempt, we write for the velocity:

$$v_\varepsilon^r = v^r + \frac{2\varepsilon}{\Delta r} \cdot max\left(\frac{(v^r - v_L^r)(v_R^r - v^r)}{v_R^r - v_L^r}, 0\right) \tag{70}$$

$$v_\varepsilon^{az} = v^{az} + \frac{2\varepsilon}{\Delta az} \cdot max\left(\frac{(v^{az} - v_L^{az})(v_R^{az} - v^{az})}{v_R^{az} - v_L^{az}}, 0\right) \tag{71}$$

$$v_\varepsilon^{col} = v^{col} + \frac{2\varepsilon}{\Delta col} \cdot max\left(\frac{(v^{col} - v_L^{col})(v_R^{col} - v^{col})}{v_R^{col} - v_L^{col}}, 0\right), \tag{72}$$

where we have specified the velocity components r, az and col in the corresponding direction with a superscript. In the JUPITER code, we use equation (68) with $v \mathrel{\widehat{=}} \vec{v}_n$ the velocity at the previous position, $\varepsilon \mathrel{\widehat{=}} x_{n+1} - x_n$, Δx is half the cell width and v_L, v_R correspond to the velocities at the interfaces between the cells (left and right neighbouring cell), respectively. In the code, we implement the updated velocities by:

```
*propertyarray[h][1][j+3] = *propertyarray[h][0][j+3] + 2.0 * ( *propertyarray[h][1][j]
- *propertyarray[h][0][j]) / ( *cellwidthplus[h][j]) * (*beta[h][j]);
```

where we use $j \in \{0,\ 1,\ 2\}$ (azimuthal, radial, colatitude) and $j+3$ is the corresponding velocity, as well as

```
*beta[h][j] = MAXIMUM(0.0, ( *propertyarray[h][0][j+3] - *uL[h])*
( *uR[h] - *propertyarray[h][0][j+3]) / ( *uR[h] - *uL[h]) );
```

and `cellwidthplus` is the distance between the position of the tracer and the boundary of the cell in positive direction, `vperpR` and `vperpL` refer to the perpendicular velocities in positive (R, right) or negative (L, left) direction, `uR, uL` correspond to the velocities in R and L direction. We define `MAXIMUM` as the maximum function.

We note that for the position update - which we determine first - the value of the velocity at the new position is required. However, at this point, the velocity is not yet updated. We can solve this issue by inserting equation (68) into the 2^{nd}-order version of equation (62). After some algebra, we obtain a formula for the position update, that does not depend on the updated velocity,

$$x_{n+1} = \left(x_n + v_n \cdot dt - \beta \cdot \frac{x_n dt}{\Delta x}\right) \cdot \left(1 - \frac{\beta dt}{\Delta x}\right)^{-1}, \tag{73}$$

where we used again Δx as half of the cell width and

$$\beta := \text{MAXIMUM}\left(0.0, \frac{(v_n - v_L)(v_R - v_n)}{v_R - v_L}\right). \tag{74}$$

The derivation can be found in the appendix 10.2. The updated positions are thus given by:

```
*PA[h][1][j] = ( *PA[h][0][j]) + dt * *PA[h][0][j+3] -
         *beta[h][j]* *PA[h][0][j] * dt / *cellwidthplus[h][j] )
         / ( 1.0 - *beta[h][j]* dt / *cellwidthplus[h][j]);
```

where again $j \in \{0,\ 1,\ 2\}$, `dt` is the time step, see Section 2.5 for further details, and `PA` is the short form for `propertyarray`. At each time step, we store the updated positions and velocities of all tracer particles in `propertyarray[:][1][:]`.

Because in the JUPITER code, the planet accretes mass over roughly 50 - 100 orbits, starting from $M_P = 0$, which is shorter than in a real CSD, we initially planned to execute this additional tracer-function only once the planet has grown to its full mass. This condition was implemented by using the mass taper, see p. 67 in [2], which is defined as the time scale, over which the planet's mass increases from $M_P = 0$ to the pre-defined final mass.

Later, we decided to execute the tracer functions from the beginning of the planet formation in order to trace the gas fluid in this stage of the evolution already.
We then write the data stored in `propertyarray[:][:][:]` in the *.txt* files and copy the post-update data into the pre-update entries for the next iteration and set the entries for the integrated data to zero:

```
for (h = 0; h < NRTPs; h++)
  for (j = 0; j < 6; j++)
    propertyarray[h][0][j] = propertyarray[h][1][j];
    *propertyarray[h][1][j] = 0.0;
```

Running the simulation with the above described updating process results in unreasonable values for both position and velocity of the tracers. This is first of all due to the poor implementation of the update step as well as the usage of quantities from the struct *beam*, namely `uL` and `uR`, in a location in the code, where we should not make use of it anymore. Furthermore, if the velocities at the centres of adjacent cells are equal, `beta` can become `-inf` and thus leading to position and velocity components `-nan`[41]. We therefore have to work more on the updating step. This however requires to introduce intermediate points during the update procedure, for which we have to store pre- and updated positions and velocities as well. Furthermore, the format of the pointers have to be changed and several new functions have to be written to solve all the problems. We thus more or less start from zero again, 3.3.2.

3.3.2 Attempt 2: Advecting the Tracers DONE

Focussing again on the update step of the tracer particles, we describe in the following the scheme in more details regarding the numerical background, see [45]. As already mentioned, this method is used in [28].
Let's assume we want to update a specific tracer particle `hh`, where

$$\texttt{hh} \in \{0, ..., (\texttt{NrTPs} - 1)\}.$$

This particle is initially placed in the centre of a cell

```
m = i * stride[0] + j * stride[1] + k * stride[2]
```

within the volume shown in figure 24 defined by (58) - (60). First, we update the position in azimuthal direction, followed by the update of the velocity in the same direction. To do so, we have to determine the position of the points A, B, C, D, E, F, G, H and I, from figure 27, then use these results to integrate the velocities. This is done by using the $3 \cdot 3 \cdot 3$ blue points in figure 27. We first apply the second order Runge-Kutta equation (72) to determine the positions of the intermediate points A, B, C, D, E, F, G, H, I and then apply equation (68) to obtain the velocities at these points. Solving these two equations is equivalent to solving the advection equation (57) numerically. Note that in order to obtain one orange point, we need three adjacent blue points due to the usage of v_L and v_R in β, see equation (77), which are the velocities at the corresponding centres in the neighbouring cell of that direction. This is indicated by the blue lines. The nine orange points A-I therefore have the correct azimuthal components in the position and velocity vectors. We remind the reader, that updating in azimuthal direction first is the choice of the author. The final positions and velocities of the tracers should be independent on the order in which spatial direction we update.
In the next step, we apply equation (72) and (68) again, now on the nine orange points in order to obtain the intermediate points J, K ,L, indicated by the orange lines in figure 27, which have thus correct azimuthal and radial components. Finally, we obtain M by repeating the same process applied to the points J, K, L, indicated by the green lines in figure 28. Additionally, we store these position and velocity components in the pointer array `PropertyArr` which is printed in a text file, see Section 3.3.11[42]. Looping over all tracer particles for a certain time-step advects all tracers in the simulation with the gas fluid for the current time step. The code of the update functions is shown in Section 3.3.3 in listings 1 - 2.

[41] We encounter this problem again and discuss the solution there

[42] Note, that this last step - storing the final data in `PropertyArr` - is additional and not necessary. We could just print the pointer `point_M` in the output file. However, because we have already written the output function during the developement of 3.3.1 as well as other functions with `PropertyArr` and are used to this notation, we make this extra step. Furthermore, we change the size of the pointer in comparison to Section 3.3.1, where we used `PropertyArr[NrTPs][2][6]`. Hereafter we use `PropertyArr[2][6][NrTPs]` or `point_Q[2][11][NrTPs]`. This change was necessary, because the memory allocation now proved to be more complicated.

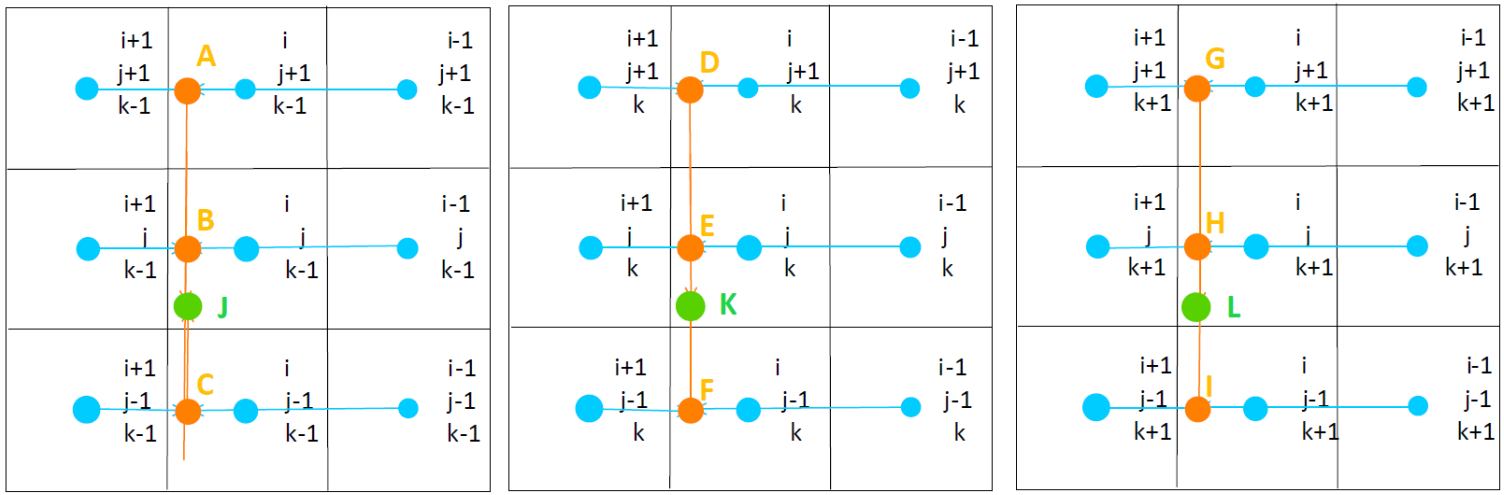

Figure 27: Illustration part 1 of the update scheme for a tracer particle that is placed in the blue point in cell i, j, k and moves the magenta point M (shown in figure 28). The complete update scheme of a single tracer requires 27 cells in total. We first update in azimuthal direction, which gives us the intermediate points A, B, C, D, E, F, G, H, I, using the second order Runge-Kutta equation, (72), for the position update and equation (68) for the velocity update. These points have therefore correct azimuthal components of position and velocity vector. In the next step, we update in radial direction using the orange points, resulting in the intermediate points J, K, L. These points have correct azimuthal and radial components. Finally, we update in colatitude direction using the green points to find the end position and velocity M, seen in figure 28. For simplicity, we draw the grid in Cartesian co-ordinates, however the process works in the same way in whatever co-ordinate system is preferred, spherical or cylindrical. The illustration is created by [78] based on [45].

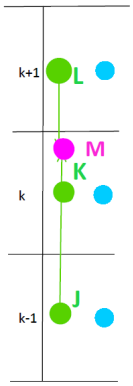

Figure 28: Illustration part 2 of the update scheme for a tracer particle that is placed in the blue point in cell i, j, k and moves the magenta point M. For details, see caption in figure 27. The illustration is created by [78].

Figure 29 shows an overview of the final implementation, including all functions that were written within the scope of this work with a brief explanation on their task.

3.3.3 The Update functions DONE

In this Section, we present the update functions for the scheme presented in Section 3.3.2 and figures 27 - 28. Within these function, several other functions are called, which are all explained in the corresponding sections.

The first function we call in order to update the tracers is `Tracer_Update_orange`, see listing 1, which takes the blue points (see 27) as input, that represent the centres of the cells, to determine `beta` from equation (77). We use this result to update the position, obtaining the orange points A-I. The three position co-ordinates are temporarily stored in `Storage`, which is input for several other functions in the following, see Sections 3.3.5 - 3.3.9 for further details on these function. These functions correct the azimuthal component due to the co-rotating grid and ensure periodicity in $[-\pi, \pi]$, correct the radial and colatitude component, such that they are within the boundaries given in [2] or determine the cell indices `i, j, k` after the position integration. This step - storing the updated position not directly in `point_Q` - is necessary, because after an arbitrary update step, a certain tracer can be in a new cell and we first have to determine in which exact cell it will be, see Section 3.3.5, because when running the code in parallel, the new position could lie outside the volume covered by the previous CPU, thus we have to communicate this tracer to the new CPU. The implementation on parallelizing the tracer part is discussed in Section 3.5. Once we determined the correct updated position, we call `TracerDistance()`, see listing 4. In short, this function determines the cell indices `i, j, k` and in case the tracer has flown to a new cell, it calls another function, `Tracer_CPUnumber` 17, that determines the number of the CPU which is responsible for that specific cell `i, j, k` and thus for that tracer. Here, we do not discuss further what happens if the tracer is not in the same CPU-patch anymore. For information on this, the reader is referred to Section we compute the velocity at the new position. A short section follows that is specifically for the velocity correction in case of massive tracers. This part is discussed in Section 3.5. With the correct updated position we can then determine the velocity at this point, which we have to correct in case of massive tracers, see 3.4 for more details. Finally, we multiply the azimuthal velocity component with -1.0, if the spatial azimuthal component had to be corrected to avoid accumulation of tracers at $\theta = \pm\pi$, see Section 3.3.8.

For the next step of the updating scheme, i.e. points J, K, L , we determine again the coefficient `beta`, `Storage` and then `point_Q[1][jj][hh]`, using the function `Tracer_Update_green ()`. This function performs similar computations as `Tracer_Update_orange` and we thus do not show the code. However this time, since the green points are the result of applying equations (72) and (68) on the orange points, we have to use those values (for clearness, see figure 27), i.e. the data from the orange points are input for the `Tracer_Update_green` function.

The same holds true for M, where we use the data from the green points as input, see `Tracer_Update_magenta` in listing 2. In addition, we store the data of M in `PropertyArr`, which we then use for the output, see Section 3.3.11.
Note that in `Tracer_Update_magenta()`, we call `Tracer_reposition()`, in case we had to re-position a tracer

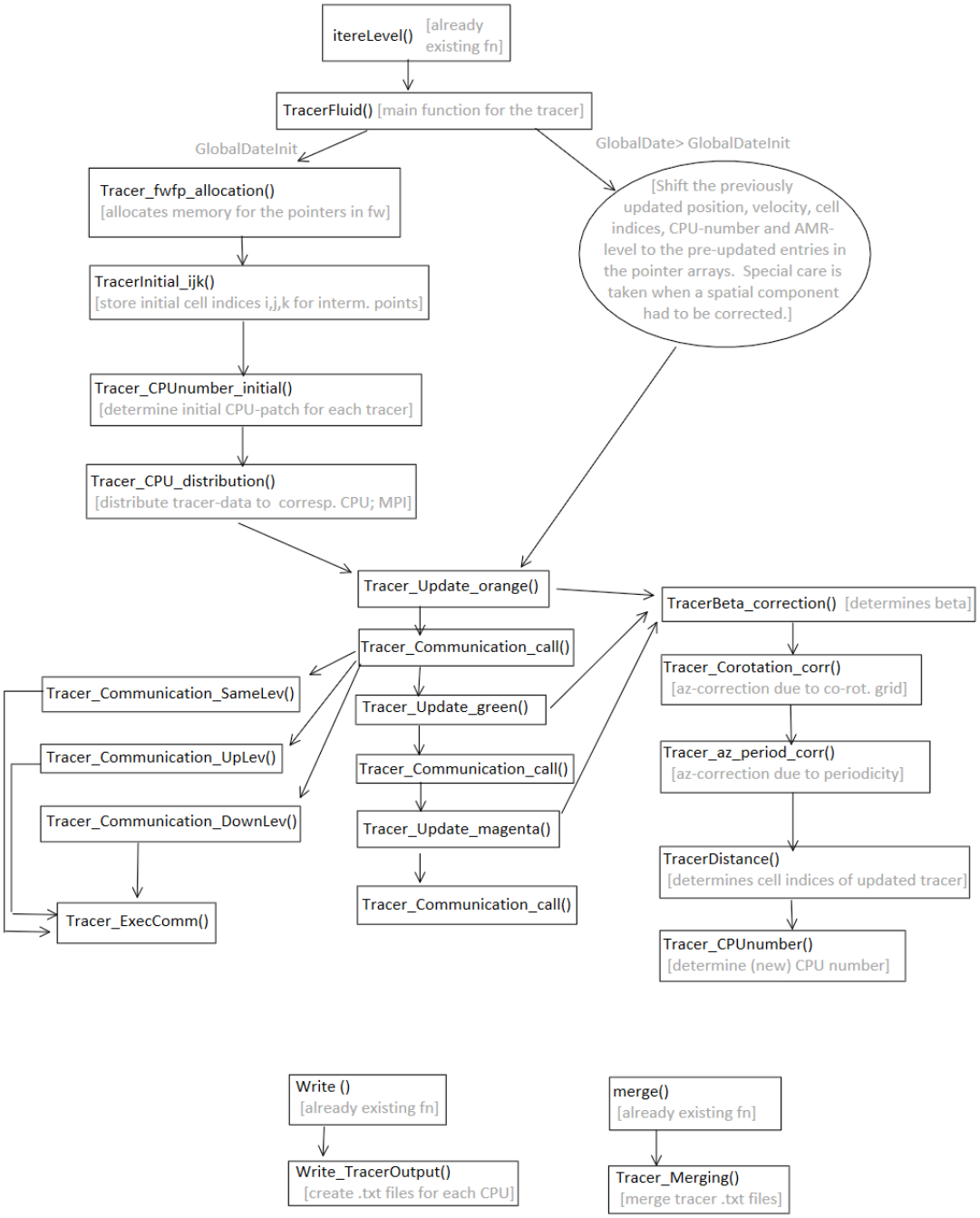

Figure 29: Overview of the tracer implementation and the functions used in this work, assuming `TRACERPARTICLE == YES`.

```
void Tracer_Update_orange (NrTPs, hh, Delta, dt, point_Q, density_gas, idenom, E,
    tracer_density)
  long NrTPs, hh;
  real *Delta;
  real dt, E;
  real ***point_Q;
  real density_gas, idenom, tracer_density;
{
  /* [...] */
  long jj, sum;
  real *Storage[3];
  real* beta[3];
  for (jj = 0; jj < 3; jj++){
    Storage[jj] = (real*)prs_malloc(sizeof(real));
    beta[jj] = (real*)prs_malloc(sizeof(real));
  }
  real *velocity[3], *center[3];
  long stride[3], gncell[3];
  /* [...] */

  if (GlobalDate == GlobalDateInit) {
    for (jj = 0; jj < 3; jj++){
      sum = (long)(point_Q[0][_AZIM_+6][hh]*stride[_AZIM_] + point_Q[0][_RAD_+6][hh]*
    stride[_RAD_] + point_Q[0][_COLAT_+6][hh]*stride[_COLAT_]);

      point_Q[0][jj][hh] = center[jj][fw->TP_m[hh]+ sum];
      point_Q[0][jj+3][hh] = velocity[jj][fw->TP_m[hh]+ sum];
    }
  }
  for (jj = 0; jj < 3; jj++){
      *beta[jj] = TracerBeta_correction(velocity[jj][fw->TP_m[hh]+stride[0]+sum],
    velocity[jj][fw->TP_m[hh]+sum],\
                velocity[jj][fw->TP_m[hh]-stride[0]+sum], fw->TP_m[hh]);
    if (TRACER_MASS == 0.0  ||  TRACER_MASS != 0.0){
      *Storage[jj] = (point_Q[0][jj][hh]+ dt * point_Q[0][jj+3][hh] + dt * point_Q[0][jj
    ][hh] /\
                    Delta[jj] * (*beta[jj])) / (1.0 - dt * (*beta[jj]) / Delta[jj]);
    }
  }
  Tracer_Corotation_corr (Storage);
  boolean AZ_corr = Tracer_az_period_corr (Storage);
  TracerDistance(Storage, hh, point_Q);
  for (jj = 0; jj < 3; jj++){
    point_Q[1][jj][hh] = *Storage[jj];
    point_Q[1][jj+3][hh] = point_Q[0][jj+3][hh] + 2.0 * (point_Q[1][jj][hh] - point_Q[0][
    jj][hh]) / Delta[jj] * (*beta[jj]);
    if (TRACER_MASS != 0.0) {
      point_Q[1][jj+3][hh] *= tracer_density * E;
      point_Q[1][jj+3][hh] += velocity[jj][fw->TP_m[hh]] *(1.0 + density_gas * E);
      point_Q[1][jj+3][hh] *= idenom;
    }
  }
  if (AZ_corr == TRUE) { point_Q[1][_AZIM_+3][hh] *= (real)(-1); }
}
```

Listing 1: The Tracer_Update_orange function determines the updated position and velocity of the intermediate orange points, A, B, C, D, E, F, G, H, I. For that, we always use the velocities at the centre of the cells as starting point as an approximation.

```
void Tracer_Update_magenta (NrTPs, hh, Delta, dt, point_Q, point_old, point_up,
    point_down, density_gas, idenom, changed_r_M, changed_col_M, E, tracer_density)
  long NrTPs, hh;
  real *Delta;
  real dt, E;
  real ***point_Q;
  real ***point_old; /* e.g. this is point_K when getting point_M */
  real ***point_up; /* e.g. this is point_L when getting point_M */
  real ***point_down; /* e.g. this is point_J when getting point_M */
  real density_gas, idenom, tracer_density;
  boolean *changed_r_M, *changed_col_M;
{
  /* [...] */
  long stride[3], gncell[3], jj, sum;
  real *Storage[3];
  /* [...] */
  real *beta[3];
  for (jj = 0; jj < 3; jj++){
    Storage[jj] = (real*)prs_malloc(sizeof(real));
    beta[jj] = (real*)prs_malloc(sizeof(real));
  }
  for (jj = 0; jj < 3; jj++){
    *beta[jj] = TracerBeta_correction ( point_up[1][jj+3][hh], point_old[1][jj+3][hh],
    point_down[1][jj+3][hh], fw->TP_m[hh]);
    if (TRACER_MASS == 0.0  ||  TRACER_MASS != 0.0){
      *Storage[jj] = (point_old[1][jj][hh]+dt * point_old[1][jj+3][hh]+dt * point_old[1][
    jj][hh]/Delta[jj] *\
                    (*beta[jj]))/(1-dt * (*beta[jj])/Delta[jj]);
    }
  }
  changed_r_M[hh] = Tracer_r_correction(Storage);
  if (changed_r_M[hh] == FALSE){
    changed_col_M[hh] = Tracer_col_correction(Storage);
  }
  Tracer_Corotation_corr (Storage);
  fw->AZ_corr[hh] = Tracer_az_period_corr (Storage);
  TracerDistance(Storage, hh, point_Q);
  for (jj = 0; jj < 3; jj++){
    point_Q[1][jj][hh] = *Storage[jj];
    if (point_Q == fw->point_M)  { fw->PropertyArr[1][jj][hh]  = point_Q[1][jj][hh]; }
    if ((changed_r_M[hh] == FALSE)  &&  (changed_col_M[hh] == FALSE)){
      point_Q[1][jj+3][hh] = point_old[1][jj+3][hh] + 2.0 * (point_Q[1][jj][hh] -
    point_old[1][jj][hh]) / Delta[jj] * (*beta[jj]);
    }
  }
  real vel[3] = {0.0, 0.0, 0.0};
  if ((changed_r_M[hh] == FALSE)  &&  (changed_col_M[hh] == TRUE)){
    vel[3] = Tracer_reposition(stride, gncell, hh, point_Q);
    for (jj = 0; jj < 3; jj++) { point_Q[1][jj+3][hh] = vel[jj]; }
  } else if ((changed_r_M[hh] == TRUE)){
    vel[3] = Tracer_reposition(stride, gncell, hh, point_Q);
    for (jj = 0; jj < 3; jj++) {
      point_Q[1][jj+3][hh] = vel[jj];
      if (TRACER_MASS != 0.0) {
        point_Q[1][jj+3][hh] *= tracer_density * E;
        point_Q[1][jj+3][hh] += fw->Velocity[jj][fw->TP_m[hh]] *(1.0 + density_gas * E);
        point_Q[1][jj+3][hh] *= idenom;
      }
    }
  }
  if (fw->AZ_corr[hh] == TRUE) { point_Q[1][_AZIM_+3][hh] *= (real)(-1); }
  for (jj = 0; jj < 3; jj++) { if (point_Q == fw->point_M) { fw->PropertyArr[1][jj+3][hh]
      = fw->point_M[1][jj+3][hh]; } }
}
```

Listing 2: The Tracer_Update_magenta function determines the final position and velocity of the complete update step, see figure 27 and 28.

because the updated radial or colatitude position exceeded the boundaries given in [2]. For more information on this function, see listing 9. In this final update function, we store the position and velocity of each tracer in `PropertyArr`, which we use for the output.

Implementing this scheme requires us to keep track of the position and velocity of all intermediate points A, B, C, D, E, F, G, H, I, J, K, L and M, the three boolean pointer arrays `changed_r_magenta, changed_col_magenta` and `AZ_corr` in which we keep track of whether or not the radial, colatitude or azimuthal component of the updated position were corrected and we thus insert the corresponding pointer arrays in the struct for the current fluid patch. We discuss this problem in Section 3.3.5. As already mentioned, we deleted the condition that the tracer implementation is only executed when the planet has already grown to its full mass, such that we can trace the motion of the gas fluid during this accretion period as well.

3.3.4 The Number of Tracer Particles DONE

As shown in figure 24 and described by equations (58) - (60), the number of tracer particles, hereafter `NrTPs`, is determined by the number of cells, whose centre lie initially in that volume. We write a separate function that determines `NrTPs`, so that we can independently access and determine the number of tracers throughout the whole code. We choose the boundaries of the volume as written in (58) - (60) and choose $c_1 = 100.0$ and $c_2 = 60.0$, which we make global, user defined input parameters and call them `Tracer_R_Min`, `Tracer_R_Max` and `Tracer_Col_Min`. The function's main part is shown in listing 3.

The cell index `m` iterates over all active cells in the current fluid patch, `_AZIM_` , `_RAD_` and `_COL_` are integers[43] referring to the spatial coordinates. `TRACER_AZ_MIN`, `TRACER_AZ_MAX` and `TRACER_COL_MIN` are the boundaries of the tracer volume shown in figure 24, `PlanetCOL` is the value of the colatitude co-ordinate of the planet. As mentioned in the beginning of Section 3.3, depending on the size of the grid cells and the choice of $c_{1,2}$, no tracer particle is introduced to the code, even though this scenario is very unlikely. This occurs when the volume defined by the above listed `if`-conditions does not contain a centre of a grid cell. To check this, we add the warning at the end of the function, see listing 3. The user is then informed that a different choice of either `Size1`, `Size2`, `Size3` in the parameter file (decreasing the size of the cells) or $c_{1,2}$ (smaller values) is required. We advise to do the latter one. In addition to that, the code exits to avoid unnecessary computational time.

3.3.5 Keeping Track of the Cells DONE

The reader may have noticed in the updating scheme described in Section 3.3.2, that at one point in the simulation, the tracers would flow from one cell to another. As described, both equation (72) and (68) require the knowledge of the next and previous cells in whichever direction the update is performed, see equation (77). This implies, that we have to keep track of the cell indices `i, j, k` as well. We solve this issue by first applying equation (72). Because we know in which cell the tracer is pre-update, we can easily use the velocities of the two adjacent cell centres to determine `beta`. Following this, we determine in a separate function the distance between this new point - we do not know in which cell the tracer is now - and the centre of the cell in which this particular tracer was pre-update, as well as the distance to the centres of all cells around this one. The shortest distance then defines the cell in which the tracer lies after the position update. We first determine the distance to the pre-update cell centre, because we assume that in most cases, the tracer has not left the cell. We store the cell indices `i, j, k` of that cell. In listing 4, we show this function for a tracer `hh`. The distances between two points P and $\tilde{P}$ in the corresponding co-ordinate system - Cartesian, cylindrical or spherical - are given by

$$d_{Cart} = \sqrt{(x-\tilde{x})^2 + (y-\tilde{y})^2 + (z-\tilde{z})^2}, \tag{75}$$

$$d_{Spher} = \sqrt{r^2 + \tilde{r}^2 - 2r\tilde{r}[sin(\theta)sin(\tilde{\theta})cos(\phi-\tilde{\phi}) + cos(\theta)cos(\tilde{\theta})]} \tag{76}$$

$$d_{Cyl} = \sqrt{r^2 + \tilde{r}^2 - 2r\tilde{r} * cos(\theta-\tilde{\theta}) + (z-\tilde{z})^2} \tag{77}$$

[43] In the JUPITER code, the order of co-ordinates as well as the co-ordinate system itself can be changed and thus `_AZIM_` , `_RAD_` and `_COL_` are used instead of numbers.

```
long TracerParticle_Counter()
{
  /* [...] */
  real planet_radius = 1.0;
  real planet_azimuth = 0.0;
  real planet_colatitude = PI / 2.0;
  real tracer_radius_min = planet_radius - (planet_radius / TRACER_R_MIN);
  real tracer_radius_max = planet_radius + (planet_radius / TRACER_R_MAX);
  real tracer_colatitude_min = planet_colatitude - (planet_colatitude / TRACER_COL_MIN);
  //real tracer_colatitude_max = planet_colatitude;       // !!!
  real tracer_azimuth_min =RANGE1LOW;  // !!!
  real tracer_azimuth_max = RANGE1HIGH;
  int cell_counter = 0;
  real *center[3];
  long stride[3], gncell[3], i, j, k, m, NrTPs;

  /* [...] */

  /* for loops to iterate through all cells in the whole grid */
        m = i*stride[0]+j*stride[1]+k*stride[2];
        if((center[_AZIM_][m] >= tracer_azimuth_min) && (center[_AZIM_][m] <=
    tracer_azimuth_max)){
          if ((center[_RAD_][m] >= tracer_radius_min) && (center[_RAD_][m] <=
    tracer_radius_max)){
            if ((center[_COLAT_][m] >= tracer_colatitude_min) &&  (center[_COLAT_][m] <=
    planet_colatitude)){
              cell_counter += 1;
            }
          }
        }

  NrTPs = cell_counter;
  MPI_Bcast(&NrTPs, 1, MPI_LONG, 0, MPI_COMM_WORLD);
  if (NrTPs == 0){
    pWarning ("Warning: NrTPs == 0\n");
    pWarning ("You should increase Size1, Size2, Size in the parameter file...\n");
    pWarning ("...or increase the tracer torus' size\n");
    exit (1);
  }
  if (GlobalDate == GlobalDateInit) {pInfo ("NrTPs = %ld in TracerParticle_Counter()\n",
    NrTPs);}
  return NrTPs;
}
```

Listing 3: The TracerParticle_Counter function, which determines the number of tracer the user places in the CSD according to the chosen boundaries of the tracer volume shown in figure 24.

Note that regardless of the chosen co-ordinate system, we name the spatial components `rad`, `az` and `colat` rather than `x`, `y`, `z` for Cartesian or `rad`, `az`, `z` for cylindrical, but the distance formula has to be adjusted.

```
void TracerDistance (Storage, hh, point_X)
  real *Storage[3];
  real*** point_X;
  long hh;
{
  /* [...] */
  long i, j, k, I, J, K, gncell[3], stride[3];
  real *distance[2], *center[3];
  /* [...] */
  long A = stride[0], B = stride[1], C = stride[2];
  for (i = 0; i < 3; i++) {   center[i] = fw->desc->Center[i]; }
  for (i = 0; i < 2; i++) { distance[i] = (real*)prs_malloc(sizeof(real)); }
  I = (long int)point_X[0][_AZIM_+6][hh];
  J = (long int)point_X[0][_RAD_+6][hh];
  K = (long int)point_X[0][_COLAT_+6][hh];
  if (__CARTESIAN){
    *distance[0] = sqrt((*Storage[0]-center[0][fw->TP_m[hh]+I*A+J*B+K*C]) * (*Storage[0]-
    center[0][fw->TP_m[hh]+I*A+J*B+K*C]) + \
                  (*Storage[1]-center[1][fw->TP_m[hh]+I*A+J*B+K*C]) * (*Storage[1]-center
    [1][fw->TP_m[hh]+I*A+J*B+K*C]) + \
                  (*Storage[2]-center[2][fw->TP_m[hh]+I*A+J*B+K*C]) * (*Storage[2]-center
    [2][fw->TP_m[hh]+I*A+J*B+K*C]));
  }
  if (__SPHERICAL){
    *distance[0] = sqrt( *Storage[_RAD_] * *Storage[_RAD_] + center[_RAD_][fw->TP_m[hh]+I
    *A+J*B+K*C] * center[_RAD_][fw->TP_m[hh]+I*A+J*B+K*C] - \
         2 * (*Storage[_RAD_]) * center[_RAD_][fw->TP_m[hh]+I*A+J*B+K*C] * \
         (sin(*Storage[_AZIM_]) * sin(center[_AZIM_][fw->TP_m[hh]+I*A+J*B+K*C]) *\
         cos(*Storage[_COLAT_] - center[_COLAT_][fw->TP_m[hh]+I*A+J*B+K*C]) +\
         cos(*Storage[_AZIM_]) * cos(center[_AZIM_][fw->TP_m[hh]+I*A+J*B+K*C]) ) );
  }
  if (__CYLINDRICAL){
    *distance[0] = sqrt ( *Storage[_RAD_] * *Storage[_RAD_] + center[_RAD_][fw->TP_m[hh]+
    I*A+J*B+K*C] * center[_RAD_][fw->TP_m[hh]+I*A+J*B+K*C] - \
         2 * (*Storage[_RAD_]) * center[_RAD_][fw->TP_m[hh]+I*A+J*B+K*C] * cos(*Storage[
    _AZIM_] - center[_AZIM_][fw->TP_m[hh]+I*A+J*B+K*C]) +\
         (*Storage[_COLAT_] - center[_COLAT_][fw->TP_m[hh]+I*A+J*B+K*C]) * (*Storage[
    _COLAT_] - center[_COLAT_][fw->TP_m[hh]+I*A+J*B+K*C]) );
  }
  point_X[1][_AZIM_+6][hh] = point_X[0][_AZIM_+6][hh];
  point_X[1][_RAD_+6][hh] = point_X[0][_RAD_+6][hh];
  point_X[1][_COLAT_+6][hh] = point_X[0][_COLAT_+6][hh];

  for ( i = -1; i < 2; i++){
    for (j = -1; j < 2; j++){
      for (k = -1; k < 2; k++){
        if (__CARTESIAN){
          *distance[1] = sqrt((*Storage[0]-center[0][fw->TP_m[hh]+i*A+j*B+k*C]) * (*
    Storage[0]-center[0][fw->TP_m[hh]+i*A+j*B+k*C]) + \
            (*Storage[1]-center[1][fw->TP_m[hh]+i*A+j*B+k*C]) * (*Storage[1]-center[1][
    fw->TP_m[hh]+i*A+j*B+k*C]) + \
            (*Storage[2]-center[2][fw->TP_m[hh]+i*A+j*B+k*C]) * (*Storage[2]-center[2][
    fw->TP_m[hh]+i*A+j*B+k*C]));
        }
        if (__SPHERICAL){
          *distance[1] = sqrt( *Storage[_RAD_] * *Storage[_RAD_] + center[_RAD_][fw->TP_m
    [hh]+i*A+j*B+k*C] * center[_RAD_][fw->TP_m[hh]+i*A+j*B+k*C] - \
           2 * (*Storage[_RAD_]) * center[_RAD_][fw->TP_m[hh]+i*A+j*B+k*C] * \
           (sin(*Storage[_AZIM_]) * sin(center[_AZIM_][fw->TP_m[hh]+i*A+j*B+k*C]) *\
           cos(*Storage[_COLAT_] - center[_COLAT_][fw->TP_m[hh]+i*A+j*B+k*C]) +\
           cos(*Storage[_AZIM_]) * cos(center[_AZIM_][fw->TP_m[hh]+i*A+j*B+k*C]) ) );
        }
        if (__CYLINDRICAL){
          *distance[1] = sqrt ( *Storage[_RAD_] * *Storage[_RAD_] + center[_RAD_][fw->
    TP_m[hh]+i*A+j*B+k*C] * center[_RAD_][fw->TP_m[hh]+i*A+j*B+k*C] - \
             2* (*Storage[_RAD_]) * center[_RAD_][fw->TP_m[hh]+i*A+j*B+k*C] * cos(*
    Storage[_AZIM_] - center[_AZIM_][fw->TP_m[hh]+i*A+j*B+k*C]) +\
             (*Storage[_COLAT_] - center[_COLAT_][fw->TP_m[hh]+i*A+j*B+k*C]) * (*Storage
```

```
    [_COLAT_] - center[_COLAT_][fw->TP_m[hh]+i*A+j*B+k*C]) );
56        }
57        if ( distance[1] < distance[0] ){
58          distance[0] = distance[1];
59          point_X[1][_AZIM_+6][hh] = (real)i;
60          point_X[1][_RAD_+6][hh] = (real)j;
61          point_X[1][_COLAT_+6][hh] = (real)k;
62          fw->TP_m[hh] += i*A+j*B+k*C;
63          Tracer_CPUnumber (hh, point_X);
64        }
65      }
66    }
67  }
68 }
```

Listing 4: The `TracerDistance()` function takes as input the pointer Storage, in which we previously stored the updated position. It then determines the distance to the centre of the cell of the previous position for the corresponding co-ordinate system and saves the corresponding cell indices `i,j,k`. In the next step, the distances to all other cells around cell `(i,j,k)` are determined and we check whether one of those distances is smaller than the very first one. If this is the case, it means that the tracer particle has moved into that corresponding cell and we correct the indices `i,j,k` on line 64-66 as well as the value of `m`. We discuss `Tracer_CPUnumber()` in Section 3.5.

Note, that `point_X` is the pointer arrays for one of the points , J, K, L, M that are in the current fluid patch, while `Storage` is a dummy container, that temporarily stores the updated position of that tracer. For shortness, we define `A = stride[0], B = stride[1], C = stride[2]`. `*center[3]` is the pointer that contains the centres of the grid cells in the current fluid patch. Furthermore, we store the `m` values that satisfy the initial if-conditions (see (58) - (60), figure 24 and listing 3) in a separate array `TP_m`, which is an element of the current fluid patch as well. With that, we can keep track of each tracer individually and do not have to apply the `if`-conditions again.

3.3.6 $\beta = -$NAN DONE

As mentioned previously, we delete the additional *if*-condition that the planet has to be fully grown in order for the tracer implementation to be executed. This however causes the following issue.
If we look again at the update equations (72) and (68), we find in both the coefficient β, defined as

$$\beta = \text{MAX}\left(0.0,\ \frac{(v_R - v_n)(v_n - v_L)}{(v_R - v_L)}\right). \tag{78}$$

If we start the simulation including the tracers, that is `TracerParticle == Yes`, from e.g. `output00000`, we run into the problem, that the second term in β is not defined. This occurs due to the fact that in the early stages of the simulation, the velocities in neighbouring cells are almost or even exactly[44] equal. In this case, we end up with

$$\beta = \text{MAX}\,(0.0,\ -inf)\,.$$

In this case, we get `beta = -inf`. Because we use previously determined velocities and positions for the the same update step (e.g. for points J, K, L or M) and for the next iteration, we end up with more and more `-nan` values. In order to avoid this problem, we write a new function `TracerBeta_correction()`, which takes as input arguments the three velocities `v_R`, `v_L`, `v_n` as well as `beta`. Finally, we return `beta`. Note, that using the machine epsilon in double format DBL_EPSILON requires

```
#include "float.h".
```

In listing 5 this function is shown.

3.3.7 Correcting the Azimuthal Components: Co-Rotation DONE

In general, the JUPITER code runs with a non-zero angular velocity `OmegaFrame`, implying we run the simulations with a co-rotating frame of reference, FOR, i.e. the grid rotates. The benefit if this is that the planet is always at the same position, in code units given by

$$\vec{x}_{Pl} := (az,\ r,\ col) = (0.0,\ 1.0,\ \pi/2). \tag{79}$$

[44] up to machine precision

```
#include "float.h"
#define MAXIMUM(X, Y) (((X) >= (Y)) ? (X) : (Y))

real TracerBeta_correction (v_R, v_n, v_L, hh, jj)
  real v_n, v_L, v_R;
  long hh, jj;
{
  real Beta;
  if ( (fabs((v_R - v_n)*(v_n - v_L)/(v_R - v_L)) <=  DBL_EPSILON )  ||  fabs((v_R - v_n)
    *(v_n - v_L)/(v_R - v_L))>= 1/DBL_EPSILON ){
    Beta = 0.0;
  } else {
    Beta = MAXIMUM(0.0, (v_R - v_n)*(v_n - v_L)/(v_R - v_L));
  }
  return Beta;
}
```

Listing 5: The TracerBeta_correction function determines the β factor in the formula for the position and velocity update, equation (72) and (68). In case the three velocities are very similar, the value of β would be $-inf$. To avoid this, we first check whether this is the case and set it to 0.0. Otherwise, we use equation (77).

However, as can be seen in figure 30, the implementation of the tracers is not compatible with this frame of reference but rather assumes a static grid and a moving objects (planet, gas fluid, etc.).

The co-rotating FoR gives rise to extra terms in the equations, representing pseudo forces such as the Coriolis force and the centrifugal force, [82]. In the following, we discuss the influence of the co-rotating grid to vector quantities, such as the velocity or the momentum, based on Section 10.1.5 from [82], and continue with how we implemented the correction for the tracers in the JUPITER code.

For a velocity vector $\vec{v} = (u, v, w)$, the Euler equations in discrete form for the Riemann solver in spherical co-ordinates are given by

$$\frac{\partial(\rho u V)}{\partial t} + \Delta_r(F_{rr}S_r) + \Delta_\theta(F_{r\theta}S_\theta) + \Delta_\phi(F_{r\phi}S_\phi) = \left(\frac{2P}{r} + \frac{\rho(v^2+w^2)}{r}\right)V \tag{80}$$

$$\frac{\partial(\rho u V)}{\partial t} + \Delta_r(F_{\theta r}S_r) + \Delta_\theta(F_{\theta\theta}S_\theta) + \Delta_\phi(F_{\theta\phi}S_\phi) = -\left(\rho\frac{uv}{r} - \frac{cos\theta}{sin\theta}\frac{\rho w^2 + P}{r}\right)V \tag{81}$$

$$\frac{\partial(\rho w V)}{\partial t} + \Delta_r(F_{\phi r}S_r) + \Delta_\theta(F_{\phi\theta}S_\theta) + \Delta_\phi(F_{\phi\phi}S_\phi) = -\left(\rho\frac{uw}{r} + \frac{cos\theta}{sin\theta}\frac{\rho vw}{r}\right)V, \tag{82}$$

where $F_{i,j}$ are momentum fluxes, S_j the surfaces of a grid cell in the corresponding direction, V is the volume of a grid cell, ρ the density and P the pressure.
In the co-rotating grid, we have to replace

$$\partial_t \rightarrow \partial_t - \Omega_F \cdot \partial_\phi \tag{83}$$

$$w \rightarrow w + \Omega_F \cdot r \cdot sin(\theta), \tag{84}$$

where Ω_F is the the angular velocity of the grid, corresponding to `OmegaFrame` in the JUPITER code. The two additional terms in the transformation include the pseudo forces. We can therefore either adjust the equation that we solve with these replacements or we can solve the equations as they are and correct the resulting azimuthal components. For simplicity, we choose the second option and call the function `Tracer_Corotation_corr()`, see listing 6, which shifts the azimuthal vector components into a co-rotating grid. We call this function after the position update but before the function that ensure periodicity in azimuthal direction, as explained in Section 3.3.8.

3.3.8 Correcting the Azimuthal Components: Periodicity DONE

During the development, we have come across simulation outputs that showed azimuthal co-ordinates $\theta > \pi$ or $\theta < -\pi$. While the gas fluid is written such that it is periodic in `RangeLow1` and `RangeLow2`[45], this is not the

[45] `RangeLow1` and `RangeLow2` are global variables for the first component. We usually use the co-ordinate permutation `(213)`, and thus `RangeLow1` and `RangeLow2` denote the lower and upper boundary for the azimuthal co-ordinate. Throughout this work, we set `RangeLow1`=

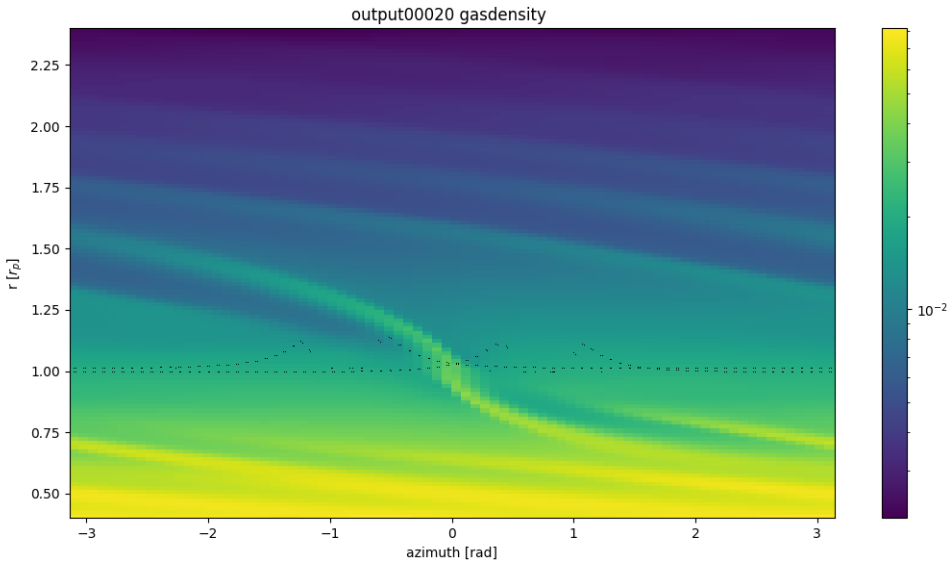

Figure 30: Simulation result, showing the gas density in the background and 320 massless tracers (black dots). This output does not take into account the additional corrections due to the co-rotating grid and thus the tracers have a wrong azimuthal position, [78].

case for the tracers. As a result, if we update particles that are close to these boundaries, they can likely have azimuthal positions $\theta > \pi$ or $\theta < -\pi$. To solve this, we write the function `Tracer_az_period_corr()`, see listing 7, which we call after the update of the position in the three update functions `Tracer_Update_orange()`, `Tracer_Update_green()` and `Tracer_Update_magenta()`, see listings 1 - 2.
A further problem is shown in figure 31. Because the implementation of the tracer particles is not periodic in $[-\pi, \pi]$, the tracers accumulate at these boundaries. Once they reach these azimuthal positions after a certain update step, they remain there for the rest of the simulation. To take care of this, we check whether the tracer is at such a position and if this is indeed the case, we multiply it by -1.0 and set the boolean variable `az_corr` to `TRUE`, which is then the return value of this function. Further information can be found in Section 3.3.2 and in the listings 1 - 2.

The input of this function is `Storage`, in which we have previously stored the updated positions, i.e. `Storage[0,1,2]` correspond to azimuthal, radial and colatitude component.

$-\pi$ and `RangeLow2` $= \pi$.

```
void Tracer_Corotation_corr (Storage)
  real *Storage[3];
{
  /* [...] */
  real *az  = Storage[_AZIM_];
  real *col = Storage[_COLAT_];
  real *rad = Storage[_RAD_];
  if (__SPHERICAL  || __CYLINDRICAL){
    *az = (*az)  * OMEGAFRAME;
  }
  if (__CARTESIAN){ /* az -> arctan (sqrt(x^2+y^2)/z) */
    *az = atan (sqrt((*az) * (*az) + (*rad) * (*rad)) / (*col)) * OMEGAFRAME;
  }
}
```

Listing 6: The Tracer_Corotation_corr function ensures that the tracers' azimuthal position takes the co-rotating grid into account.

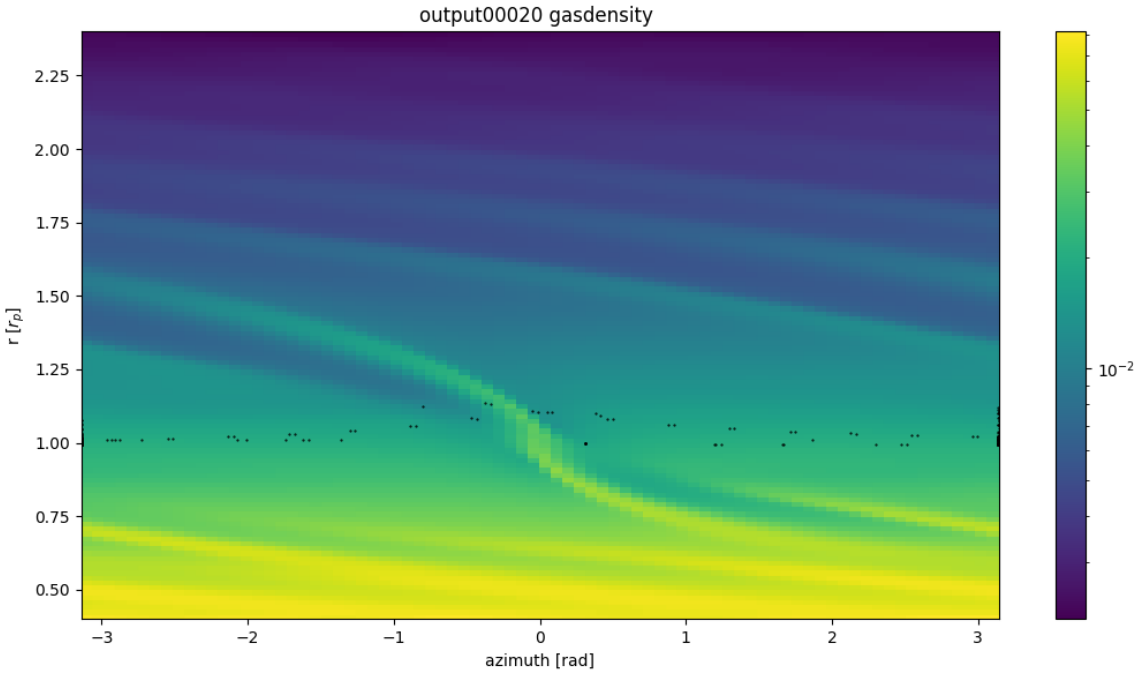

Figure 31: Simulation output, showing the accumulation of tracers at an azimuthal angle of $-\pi$ and π. The problem arises because the tracer implementation is not 'naturally' periodic in $\theta \in [-\pi, \pi]$. The point at $r = 1$, $az = \pi/10$ is because this tracer in the previous update step exceeded either the boundaries in ϕ or r direction and has thus been re-positioned, see Section 3.3.9.

Note that we could as well use the `remainder()` function of `C`, however this function behaves strange in certain situations, as can be seen in the documentation [83]. We thus refrain from writing a modulus function and rather use the `while`-condition, which might take longer to compute if the tracer is far outside of the azimuthal range. However, this applies only to the tracers that are indeed outside and we expect most of those tracers to be reasonably close to the interval boundaries even after the update step, since we make this check after every update step of all intermediate points.

As can be seen in various figures in Section 4, the current solution to avoid the accumulation of tracers at $\theta = \pm\pi$ does not work. We have therefore in a new attempt tried to re-position the tracers in case they are close to these boundaries,

$$\texttt{az} \in \{\texttt{3.14},\ \texttt{RANGE1HIGH}\}, \quad \texttt{az} \in \{\texttt{RANGE1LOW},\ \texttt{-3.14}\}.$$

The new position would then be the same as in Section 3.3.9. One problem is that this condition is satisfied for many more tracers and thus we effectively loose particles: assuming we have to re-position n tracers at a time step dt, they would all start from the same position in the next iteration and thus have the same velocity as well. For the remaining simulation, these n tracers would appear in the plot as one particle only. Another problem with this method is that we would not track the meridional circulation during the whole time interval, because once the tracer arrive in the above intervals, they do not further track their specific gas-volume. Lastly, re-positioning complicates the code, because we not only have to change the position and velocity of the tracer, we have to build the set up as shown in figures 27 and 28 as well, implying that all intermediate points A-I, J, K, L and M have to be replaced with respect to the position of the new location. Nevertheless, we have implemented this solution, resulting in a rather chaotic plot, which very apparently did not trace the gas motion correctly. Because of the finite time that was left at that time to finish this work, we had to leave this problem. The tracers therefore remain accumulated at $\theta = \pm\pi$ and we do not re-position them. We note that in case of a planet with mass up to 10^{-3} (in code units), an a large number of particles, the meridional circulation can still be traced. In case of a massive planet, the gas flow is more violent and thus more tracers accumulate at the azimuthal boundaries more quickly.

```
boolean Tracer_az_period_corr (Storage)
  real *Storage[3];
{
  boolean az_corr = FALSE;
  real *az = Storage[_AZIM_];
  if (__SPHERICAL || __CYLINDRICAL){
    if ((*az >= RANGE1LOW  && *az <= 0.0)  ||  (*az <= RANGE1HIGH && *az >= 0.0)){
      az = az;
      if (*az == RANGE1HIGH) {
        *az = RANGE1LOW;
        az_corr = TRUE;
      }
    } else if (*az > RANGE1HIGH){
      while (*az > RANGE1HIGH) { *az = *az - (2* M_PI); }
    } else if (*az < RANGE1LOW){
      while (*az < RANGE1LOW)  { *az = *az + (2* M_PI); }
    }
  }
  return az_corr;
}
```

Listing 7: The Tracer_az_period_corr function ensures that the tracers' azimuthal position is periodic between $-\pi$ and π for a spherical co-ordinate system.

3.3.9 Correcting the Radial and Colatitude Components DONE

Similar to the problem described in Section 3.3.8, in some simulations, depending on the choice of parameters, a few tracer particles are completely unrestricted in radial direction, leading to radial co-ordinates of $\sim 10^6$ or even 10^{11} (in code units). Because we want the tracers to be confined in both directions, we implement the `Tracer_r_correction()`, see listing 8 function to take care of this problem. The radial boundaries are from [2]. The correction function `Tracer_col_correction()` for the colatitude components of the tracers' position vectors has a similar structure. In case we have to re-position a tracer particle, we set the boolean variable `changed_rad` to `TRUE` and return the value of it. In case of the colatitude correction, this variable is called `changed_col`. Further information can be found in listings 1 - 2 and Section 3.3.2.

Assuming we have to either correct the radial or colatitude component and thus re-position a certain trace particle, we have to determine the velocity at the new position as well as creating new intermediate points A, B, C, D, E, F, G, H, I, J, K, L and M. For that, we first need to determine the cell indices of the new cell. As seen in listing 8, the new position of the tracer is given by

$$\theta = \frac{\pi}{10}, \quad r = 1.0, \quad \phi = \frac{\pi}{2}.$$

We then check in which cell this point is and store the corresponding cell indices, see listing 9. Once these indices are known, we can easily obtain the corresponding velocity at the centre of that cell, which we use as an approximation for the velocity at the new position of the tracer particle. The velocity vector is then the return value of the function, which we call in `Tracer_Update_magenta`, see listing 2.

3.3.10 Preparing the Pointer for the Next Time Iteration DONE

We have already mentioned in Section 3.3.1 that after all tracers are advected for a given time step `dt`, we have to shift the updated values to the pointer entries for the pre-updated values. The situation now is more complicated, because we have to remember for which tracer particle we had to apply a correction in radial or colatitude direction, in azimuthal direction due to the co-rotation of the grid or due to periodicity. The preparing is done in `TracerFluid()` and is hereafter briefly described.

1. `PropertyArr[0][k][hh] = PropertyArr[1][k][hh];`
   Since this pointer only stores values and is not used throughout the computations, we can simply shift *post-*

```
boolean Tracer_r_correction(Storage)
  real *Storage[3];
{
  long i;
  boolean changed_rad = FALSE;
  real *radial = (real*)prs_malloc(sizeof(real));
  real *colatitude = (real*)prs_malloc(sizeof(real));
  real *azimuthal = (real*)prs_malloc(sizeof(real));
  radial = Storage[_RAD_];
  azimuthal = Storage[_AZIM_];
  colatitude = Storage[_COLAT_];
  if (!__CARTESIAN){
    if ((*radial <= 2.4) && (*radial >= 0.4)){
      radial = radial;
      azimuthal = azimuthal;
      colatitude = colatitude;
    } else if ((*radial > 2.4)  ||  (*radial < 0.4)) {
      changed_rad = TRUE;
      *radial = 1.0;
      *azimuthal = 0.0 + (PI / 10.0);
      if (__SPHERICAL)  *colatitude = PI / 2.0;
      if (__CYLINDRICAL)  *colatitude = 1 * cos(PI / 10.0);
    }
  }

  if (__CARTESIAN){ /* x = r sin(az) cos (col)    y = r sin(az) sin(col)   z = r cos(az)
     */
    changed_rad = TRUE;
    *radial = 1 * sin(PI / 10.0) * cos(PI / 2.0);
    *azimuthal = 1 * sin(PI / 10.0) * sin(PI / 2.0);
    *colatitude =  1 * cos(PI / 10.0);
  }
  return changed_rad;
}
```

Listing 8: The Tracer_r_correction function ensures that the tracers' radial position is within reasonable distance to the planet's orbit, such that they help visualising the meridional circulation. A similar implementation is written for the colatitude co-ordinate.

```
real Tracer_reposition(stride, gncell, hh, point_Q)
  long stride[3], gncell[3], hh;
  real*** point_Q;
{
  /* [...] */
  real vel[3] = {0.0, 0.0, 0.0};
  long i, j, k, m, M;
  real rad, az, col;

  if (__SPHERICAL){
    az = (PI / 10.0);
    rad = 1.0; /* planet orbit radius */
    col =  PI / 2.0;
  }
  if (__CYLINDRICAL){
    az =  (PI / 10.0);
    rad = 1.0;
    col =  1.0 * cos(PI / 10.0);
  }
  if (__CARTESIAN){
    az = 1.0 * sin(PI / 10.0) * sin(PI / 2.0);
    rad = 1.0 * sin(PI / 10.0) * cos(PI / 2.0);
    col = 1.0 * cos(PI / 10.0);
  }
  /* iteration through all cell in the grid */
        m = i*stride[0]+j*stride[1]+k*stride[2];
        if ((rad >= fw->desc->Edges[_RAD_][idx[_RAD_]]) && (rad <= fw->desc->Edges[_RAD_
    ][idx[_RAD_]+1]) &&\
            (az >= fw->desc->Edges[_AZIM_][idx[_AZIM_]]) && (az <= fw->desc->Edges[_AZIM_
    ][idx[_AZIM_]+1]) &&\
            (col >= fw->desc->Edges[_COLAT_][idx[_COLAT_]]) && (col <= fw->desc->Edges[
    _COLAT_][idx[_COLAT_]+1]) ){
          M = m;
          point_Q[1][6][hh] = i;
          point_Q[1][7][hh] = j;
          point_Q[1][8][hh] = k;
          fw->Indices_reposition[0] = i;
          fw->Indices_reposition[1] = j;
          fw->Indices_reposition[2] = k;
        }

  for ( i = 0; i < 3; i++) {  vel[i] = fw->Velocity[i][M];  }
  return vel[3];
}
```

Listing 9: The Tracer_reposition function determines the new velocity of a re-positioned tracer.

update to *pre*-update for $k \in \{0,..,5\}$.

```
if (fw->changed_r_magenta[hh] == FALSE   &&   fw->changed_col_magenta[hh] == FALSE){
   for (k = 0; k < 11; k++){
       fw->point_A[0][k][hh] = (fw->AZ_corr[hh]==FALSE)? fw->point\_A[1][k][hh] : ((
    k==_AZIM_ || k==_AZIM_+3)? fw->point_A[1][k][hh]*(-1.0) : fw->point_A[1][k][hh])
     ;
```

If we did not re-position in any of the three spatial directions, we can trivially shift. If correction in azimuthal was necessary due to accumulation at $\theta = \pm\pi$, we multiply the azimuthal position and velocity with -1.0, but leave the radial and colatitude components as they are before shifting. The same holds true for all other intermediate points.

```
if (fw->changed_r_magenta[hh] == TRUE   ||   fw->changed_col_magenta[hh] == TRUE) {
   for (k = 0; k < 3; k++){ fw->point_M[1][k+6][hh] = (real)fw->Indices\_reposition[
    k]; }
   for (k = 6; k < 11; k++){
       if ((fw->AZ_corr[hh] == FALSE) || (fw->AZ_corr[hh] == TRUE)){
          fw->point_A[0][k][hh] = (k==7)? fw->point_M[1][k][hh] + (real)( 1) : ( (k
    ==8)? fw->point_M[1][k][hh]+(real)(-1) : fw->point_M[1][k][hh] );
```

In case we had to re-position because the updated spatial co-ordinates in either radial of colatitude direction exceed the boundaries, the tracer particle now starts form a different position as described in Section 3.3.9. If this is the case, we save the indices of that cell in `point_M` and then shift the cell indices for the other intermediate points as well. Note, that we need to add $\pm 1,\ 0$ depending on where the intermediate point lies in the grid with respect to `point_M`, see figure 27. The above code is thus only value for `point_A`. Note that it is irrelevant whether `AZ_corr[hh] == FALSE` or `AZ_corr[hh] == TRUE`, because if one of the conditions in the first `if` is satisfied, the tracer has been completely re-positioned.
With the shifted cell indices, we can now shift the positions and velocities as well,
`Tracer_ (center, velocity, A, B, C, hh, point_A, 0.0, 1.0, -1.0);`
The main part of the `Tracer_()` function is

```
for (long k = 0; k < 3; k++){
      point_Q[0][k][hh] = center[k][(long)point_Q[0][6][hh] * A + (long)point_Q
    [0][7][hh] * B + (long)point_Q[0][8][hh] * C];
      point_Q[0][k+3][hh] = velocity[k][(long)point_Q[0][6][hh] * A + (long)point_Q
    [0][7][hh] * B + (long)point_Q[0][8][hh] * C];
    }
```

where again `point_Q` is any intermediate point of the update scheme and `A, B, C = stride[0, 1, 2]`.

4. Finally, we reset the boolean pointers for the next iteration, regardless of which of the above cases has been selected.

```
fw->AZ_corr[hh] = FALSE;
fw->changed_r_magenta[hh] = FALSE;
fw->changed_col_magenta[hh] = FALSE;
```

3.3.11 Writing the Output File DONE

DONE The updated positions and velocities of the tracers[46] are printed in text files - one for each CPU - which are stored in the output directory `outputXXXXX` and name them `Property_Array_N_M.txt`, where N is the output number and M the number of the CPU. In the listing 10, we show the code of this function, which is called in `Write()`. The `if`-condition

```
if (GlobalDate > GlobalDateInit)
```

[46] i.e. `i = 1`

```
1  void Tracers_WriteOutput (fp, grid, number, TP_cpu_number)
2    /* [...] */
3    long number;
4    int TP_cpu_number;
5  {
6    char filename[MAXLINELENGTH];
7    long i, h;
8    /* [...] */
9    long NrTPs;
10   if ((GlobalDate > GlobalDateInit)){
11     NrTPs = (TracerParticle_Counter());
12     FILE *fptr;
13     fflush (stdout);
14     sprintf(filename, "Property_Array_%ld_%d.txt", number, CPU_Rank);
15     fptr = prs_opend (filename);
16
17     fprintf(fptr, "%ld\n", NrTPs);
18     for(h=0; h<NrTPs; h++){
19       for (i = 1; i < 2; i++){
20         if ((fw->PropertyArr[i][0][h] != 0)  &&  (fw->PropertyArr[i][1][h] != 0)  &&  (fw
     ->PropertyArr[i][2][h] != 0) &&\
21               (fw->PropertyArr[i][3][h] != 0)  &&  (fw->PropertyArr[i][4][h] != 0)  &&  (fw
     ->PropertyArr[i][0][h] != 0)  ){
22           fprintf(fptr, "%g  %g  %g  %g  %g  %g   (fw->PropertyArr[i][][h]; i=%ld, h=%ld)
     \n", fw->PropertyArr[i][0][h],\
23             fw->PropertyArr[i][1][h], fw->PropertyArr[i][2][h], fw->PropertyArr[i][3][h],
     fw->PropertyArr[i][4][h], fw->PropertyArr[i][5][h], i, h);
24         }
25       }
26     }
27     fclose (fptr);
28   }
29 }
```

Listing 10: The Tracers_WriteOutput function creates the output file Property_Array_X_X.txt for the tracers, in which we write the updated position and velocity components as well as the cell indices `i, j, k`. The first number in the file name corresponds to the output number, e.g. in the directory `output00001` we find the files Property_Array_1_X.txt. The second number corresponds to the CPU-number, i.e. if we run a simulation on two CPUs we have Property_Array_1_0.txt and Property_Array_1_1.txt in the same directory.

is to ensure that the output is only generated once the `TracerFluid()` function is performed at least once. Details on `NrTPs` can be found in Section 3.3.4. Because when running the simulation on multiple CPUs, a certain tracer occurs only in one CPU and it should not be printed in the output file of all other CPUs (at that particular time step). To do this, we check

```
if (Property_Array[i][0-5][h] != 0),
```

otherwise the CPU-files would contain a row of zeros whenever a certain tracer is not in that CPU-patch. The `for`-loop on line 19 can be changed to `for (i = 0; i < 2; i++)`, thus printing the *pre-* and *post*-updated position and velocites.

3.3.12 Merging DONE

In Section 3.3.11, we have shown how the data of the tracer particle is written in text files `Property_Array_N_M.txt`, where N is the output number and M is the CPU rank. The merging for the gas fluid data is done in the corresponding files. For the tracers, we write a similar function, `Tracer_merging()`, which we call in the beginning of `merge_desc.c`,

```
1  MPI_Barrier (MPI_COMM_WORLD);
2  char *sTring2 = "post";
3  if ((TRACERPARTICLE==YES) && (strstr(OUTPUTDIR, sTring2) != NULL)){
4    Tracer_merging (number);
5  }
```

```
void Tracer_merging (number)
  int number;
{
  FILE *file, *file_descr, *file_tot;
  char filename[MAXLINELENGTH], file_descriptor[MAXLINELENGTH], file_total[MAXLINELENGTH
    ];
  long ncpu;
  /* [...] */
  /* The relevant lines of the descriptor file are scanned and stored to get ncpu */
  fclose (file_descr);

  fflush (stdout);
  sprintf(file_total, "%soutput%05ld/Property_Array_%ld.txt", OUTPUTDIR, number, number);
  file_tot = prs_opend (file_total);

  char line[MAXLINELENGTH] = {0};
  real az, rad, col, vaz, vrad, vcol;
  long H, I;
  MPI_Barrier (MPI_COMM_WORLD);
  if (number > 0){
    for (long ii = 0; ii < ncpu; ii++){
      sprintf (filename, "%soutput%05ld/Property_Array_%ld_0.txt", OUTPUTDIR, number,
    number);
      file = fopen (filename, "r");
      if (file == NULL) { prs_error ("I don't find %s. I can't restart.", filename); }
      while (fgets(line, MAXLINELENGTH, file)) {
        fscanf (file, "%lf  %lf  %lf  %lf  %lf  %lf   (fw->PropertyArr[i][][h]; i=%ld, h
    =%ld)", &az, &rad, &col, &vaz, &vrad, &vcol, &I, &H);
        fprintf(file_tot, "%.12g  %.12g  %.12g  %.12g  %.12g  %.12g   (fw->PropertyArr[i
    ][][h]; i=%ld, h=%ld)\n", az, rad, col, vaz, vrad, vcol, I, H);
      }
    }
    fclose (file);
    fclose (file_tot);
    printf (" => Tracer_merging() Done.\n");
  }
}
```

Listing 11: The Tracer_merging function first reads from the descriptor file how many CPUs are used for the simulation. It then creates the end file `Property_Array_N.txt`, where *N* is the output number, and reads the tracer data from all 'CPU-files' and writes it to the new file, such that we end up with a text file that contains all tracer data from all CPUs. We use this file to create the plots in Section 3.3.13.

Listing 11 shows this function. We first have to determine the number of CPUs, `ncpu`, that the simulation uses. We obtain this information by scanning the descriptor file. In the next step, we create a text file `Property_Array_N.txt`, where *N* is the current output number. Finally, we scan each tracer output file, created for each CPU, and store all the information in the new file.

3.3.13 Plotting the Tracer Output File DONE

In Section 3.3.11, we have created one output file for each CPU at a certain output time during the simulation. We discussed the merging of these files in 3.3.12. In order to plot the `Property_Arrray_N.txt` file, we add a python routine to the already existing `pyJupiter.py` file, that reads the data from this file, see listing 12. In case the user has chosen to print both *pre-* and *post*-updated position and velocites in listing 10 line 19, we have to use
`TracerNew_0 = np.genfromtxt(self.folder+'/Property_Array_%.1d.txt'%(self.N), skip_header=NrTPs, usecols=0).`
and equivalent in the other two lines, which is why we include the number of tracers, `NrTPs`. To plot the tracers, we include the lines in listing 13 at the end of the corresponding, already existing, Python routines.

Note that in case the simulation runs sequential, we would not have to merge the outputs from the gas fluid. However, the python routine reads only tracer files `Property_Array_N.txt` and not `Property_Array_N_M.txt`, where *N* is the output number and *M* is the CPU-number. To avoid this, the user is advised to make a few changes in the python routine, namely

```
def readdataTracer(self, reflvl):
        TracerNew_0 = []        # post-update, 1st co-ordinate
        TracerNew_1 = []        # post-update, 2nd co-ordinate
        TracerNew_2 = []        # post-update, 3rd co-ordinate
        delimiter = "  "        # NOTE: between columns, we have two spaces

        # IMPORTANT: BEFORE PLOTTING THE TRACERS, YOU HAVE TO MERGE NO MATTER HOW MANY CPUs WOU USED
        # IF YOU ONLY USED 1 CPU, YOU CAN ALTERNATIVELY USE '/Property_Array_%.1d_0.txt' IN THE 3 LINES BELOW.
        NrTPs = np.loadtxt(self.folder+'/Property_Array_%.1d_0.txt'%(self.N), dtype='int', delimiter="\n", max_rows = 1)
        TracerNew_0 = np.genfromtxt(self.folder+'/Property_Array_%.1d.txt'%(self.N), skip_header=1, usecols=0)
        TracerNew_1 = np.genfromtxt(self.folder+'/Property_Array_%.1d.txt'%(self.N), skip_header=1, usecols=1)
        TracerNew_2 = np.genfromtxt(self.folder+'/Property_Array_%.1d.txt'%(self.N), skip_header=1, usecols=2)

        return TracerNew_0, TracerNew_1, TracerNew_2
```

Listing 12: Python routine to read the data from the tracer output file PROPERTY_ARR.TXT, see Section 3.3.11.

```
if TracerParticle == True:
  Tracer_coord = self.readdataTracer(lvl)
  plt.plot(Tracer_coord[0], Tracer_coord[1], 'k', linestyle='None', marker = 'o', markersize=0.5)
```

Listing 13: Python routine to plot the data from the tracer output file PROPERTY_ARR.TXT, see Section 3.3.11. We add these lines to already existing functions. This example is for an $r = r(az)$ plot in case of a spherical co-ordinate system and co-ordinate permutation (213).

`TracerNew_0 = np.genfromtxt(self.folder+'/Property_Array_%.1d_0.txt'%(self.N), skip_header=1, usecols=0)`
and equivalent in the other two lines.

3.4 The Massive Tracer Particles DONE

In order to simulate the trajectories of massive astrophysical objects, such as pebbles or planetesimals[47], we introduce a further global variable to the code, `Tracer_Mass`, which has a user defined value. Note that with this approach, all simulated tracers have the same mass. Contrary to the massless tracer implementation, we need a function for the massive case that couples the tracers to the gas fluid. To do so, we adjust the function, which couples the gas- and dust fluid introduced in [54], `FluidCoupling()`, such that it couples the gas fluid and the tracers. The two attempts are described below. The communication between the different CPUs and the several levels of grid refinement is implemented in the same way as in the code with massless tracer particles, see Sections 3.5 and 3.6.

The massive tracer module can be chosen in the parameter file, by selecting `TRACER_MASS == 1.0`. This value does not have to be changed, as the mass or mass distribution of the tracers is created in the code itself.

3.4.1 Attempt 1: Coupling two continuous fluids DONE

In a first attempt, we need to convert the discrete tracers, whose positions and velocities are stored in `propertyarray[NrTPs][2][6]` into a continuous field / fluid, since the already existing coupling-function works on fluids. For that, we set up two additional fields, one for the position and one for the velocity of the tracers, that have the value 0.0 everywhere in the current fluid patch and 1.0 and $|\vec{v}_{TP}|$, respectively, at the position of the tracer. In order to avoid point-like

[47] In the case of dust, the user is advised to use the dust implementation written by F. Binkert, [54]. The dust is treated as a fluid, instead of discrete particles.

tracer particles, we set up the conditions

$$\begin{aligned} r_{TP}^{old} - \frac{r_{TP}^{old}}{100} &< r < r_{TP}^{old} + \frac{r_{TP}^{old}}{100}, \\ az_{TP}^{old} - \frac{az_{TP}^{old}}{10} &< az < az_{TP}^{old} + \frac{az_{TP}^{old}}{10}, \end{aligned}$$

for the space co-ordinates and

$$\begin{aligned} v_{r,TP}^{old} - \frac{v_{r,TP}^{old}}{50} &< v_r < v_{r,TP}^{old} + \frac{v_{r,TP}^{old}}{50}, \\ v_{az,TP}^{old} - \frac{v_{az,TP}^{old}}{50} &< v_{az} < v_{az,TP}^{old} + \frac{v_{az,TP}^{old}}{50}, \\ v_{col,TP}^{old} - \frac{v_{col,TP}^{old}}{50} &< v_{col} < v_{col,TP}^{old} + \frac{v_{col,TP}^{old}}{50} \end{aligned}$$

for the velocity components, where x_{TP}^{old} refers to a pre-updates co-ordinate or velocity of the tracer. Note that $r,\ az,\ col,\ v_r,\ v_{az}$ and v_{col} denote any position / velocity co-ordinate in the current fluid patch.

After spending a significant amount of time to implement the above idea, a devastating problem became obvious: the JUPITER code does not know the coordinates of all points in the simulated volume, but rather only the centres and boundaries of the grid cells. As a consequence, converting the discrete tracers, which can be anywhere in the cell and not just in the centre or on the boundaries due to them being advected with the gas fluid, into a continuous fluid (vectorfield) would set the value of that fluid in the whole grid set to 1.0 and not only in a small region around the actual position of the tracer. When building the complete tracer fluid, we would end up with a function that has the constant value of 1.0 over the whole fluid patch and thus loose essentially all information regarding the positions. We therefore have to refrain from this attempt and find a new solution.

3.4.2 Attempt 2: Coupling the gas fluid with the tracer particles DONE

In the second attempt, we aim to adjust the `FluidCoupling()` function in such a way that it couples the gas fluid with the discrete tracer particles. We call this function `TracerFluidCoupling()` and call it in the `TracerFluid()`-function within an additional `if`-condition

```
if (TRACERMASS != 0.0)
```

in order to ensure that this function in only executed in case of massive tracers. It is executed after the position and velocity update scheme of a certain tracer is completed. Furthermore, we insert the `if`-conditions from equations (58) - (60) in the original version of the coupling function, and define a density and velocity for the tracer particles, d_{TP} and v_{TP}, in a similar way that is has been done for the gas.

The problem with this attempt is that we call `TracerFluidCoupling()` after the position and velocity update scheme is completed. However, as discussed in Section 1.6, depending on the mass of the tracers, their updated position should be closer to the pre-update position than in case of massless tracers. To achieve this, we would have to call this function before the update scheme, which leads us to the next attempt in which we simply modify the already written update functions 1 - 2.

3.4.3 Attempt 3: Modify the Update Functions

TODO The final attempt to implement massive tracers is to modify the three update functions in 1 - 2. After the position update, we call several correction functions and determine the cell indices and the CPU-number, followed by the velocity update. Up to this point, the functions perform the same computations both for massless and massive tracers. However, in case of massive tracers, we modify the updated velocity in a similar way as is already

implemented in `FluidCoupling()`, see [54]. For that, we define the tracer particles density as

```
    tracer_density = (3.0 / RHO0);
tracer_density *= (4.0 * PI * tracer_size * tracer_size * tracer_size / 3.0);
tracer_density *= invvol[TP_m[hh]]; .
```

We therefore assume a material density of $3g/cm^3$, which we convert into code units. The result is multiplied by the volume of the tracer, which we assume to be a perfect three-dimensional sphere. We finally multiply the result by the inverse volume of a cell. The tracer density is therefore taken to be the total mass of a tracer spread over the cell in which the tracer is. Note that this assumes there is only one tracer in the cell. In a first step, we define the tracers' size to be `tracer_size = 0.01 / R0` (in code units). We later try to implement a size distribution, where we use

$$\frac{dN}{da} \propto a^{-q}, \qquad q = 2.8 \pm 0.1, \tag{85}$$

from [84] with a the size of the tracers. The sizes in this work are then

$$a \in \{0.01,\ 0.1,\ 1.0,\ 10.0,\ 100.0,\ 10000.0\}, \tag{86}$$

which are the same tracer sizes in [33].
To adjust the updated velocity for massless tracer to massive tracers, we multiply first by the tracer density and `E`, the latter is defined as the time step `dt` times the coupling `tracer_coupling`. We use an equivalent formula for `tracer_coupling` as for the dust coupling. The resulting velocity is finally divided by the dimensionless parameter `idenom`, for which we again use an equivalent formula as in the dust-gas-coupling.

We highlight that very first updated position of the intermediate points A, B, C, D, E, F, G, H, I are incorrect for the massive tracer, because they do not include any adjustments. The corresponding velocities however are corrected. The very first updated positions at J, K, L are obtained by using the update orange positions and velocities, thus include incorrect values as well. The same holds true for the first M position. After the first update scheme, this error does not occur anymore, since we now use velocities that take the effect of the tracer's mass into account to obtain the new position.

3.5 Communication Between the CPUs

TODO Because the JUPITER code runs in parallel on multiple CPUs, we have to modify the tracer files such that the tracer particles are correctly transferred across the cell boundaries. Special care has to be taken, when the tracer particles are in the two rows of boundary cells (in each dimension) of the fluid patch, since these cells are active cells for the current fluid patch, but act as (passive) ghost cells for the fluid patches next (and previous) patches. Thus, the update step for the whole fluid patch includes these ghost cells, in which the particles are updated already (for the previous patch) or not yet (for the next patch).

In order to parallelize the tracer implementation, we use functions from the `Message Passing Interface (MPI)` library. Documentations to each of these functions can be found in [85], [86], [87] or [88]. Because the already existing communication functions are written for fluids, i.e. vector- and scalar-fields, we write new functions for the communication of the tracers, which are based upon them. Figure 29 shows the idea behind it: for a certain tracer particle, we first determine A, B, C, D, E, F, G, H, I and check whether one of the new position lies outside of the (active!) CPU-patch. If this is the case, we call `Tracer_Communication_call()`, see listing 14, which calls the other communication functions, depending on the AMR-level. The main purpose of this function is to make the main tracer function, `TracerFluid()` well-arranged and shorter.

The functions `Tracer_Communication_UpLev()` and `Tracer_Communication_DownLev()` are described in 3.6, see listing 18. In the next step, the positions and velocities for J, K, L are determined and again `Tracer_Communication_call()` is called. The same scheme is applied for M. We discuss the communication between CPUs on the same grid refinement level hereafter and the corresponding function is given in listing 15.

3.5.1 Keeping Track of the CPU-patch

```
void Tracer_Communication_call (point_Q, hh)
  real ***point_Q;
  long hh;
{
  /* [...] */
  if (point_Q[0][9][hh] != point_Q[1][9][hh]) {
    if (point_Q[0][10][hh]     == point_Q[1][10][hh]) {Tracer_Communication_SameLev(GHOST
    , point_Q, hh);}
    if (point_Q[0][10][hh] + 1 == point_Q[1][10][hh]) {Tracer_Communication_UpLev(GHOST,
    point_Q, hh);}
    if (point_Q[0][10][hh] - 1 == point_Q[1][10][hh]) {Tracer_Communication_DownLev(GHOST
    , point_Q, hh);}
  }
}
```

Listing 14: The `Tracer_Communication_call()` calls the communication functions for a certain tracer depending on whether the tracer remains on the same AMR-level, flows into a finer or coarser level.

```
void Tracer_Communication_SameLev (type, point_Q, hh)
  long type, hh;
  real ***point_Q;
{
  /* [...] */
  long lev = point_Q[1][10][hh];
  long i, nvar=11;
  /* INFO: "11": az, rad, col, vaz, vrad, vcol, i,j,k, CPU-nr, level */
  com->tracerbuffer = (real*)prs_malloc (nvar * sizeof(real));
  while (com != NULL) {
    if ((com->dest_level == lev) && (com->src_level == lev)) {
      if (com->CPU_src == CPU_Rank) {
        for (i = 0; i < nvar; i++) {  com->tracerbuffer[i] = point_Q[1][i][hh]; }
      }
    }
    com = com->next;
  }
  Tracer_ExecComm (GHOST, nvar, hh, point_Q);
}
```

Listing 15: The `Tracer_Communication_SameLev()`.

```
void Tracer_ExecComm(type, nvar, hh, point_Q)
  long type, nvar, hh;
  real ***point_Q;
{
  /* [...] */
  MPI_Status stat;
  long nr_req=0, levsrc, levdest;
  int sourceCPUrank, destCPUrank;
    levdest = point_Q[1][10][hh];
    levsrc = point_Q[0][10][hh];
    sourceCPUrank = point_Q[1][9][hh];
    destCPUrank = point_Q[0][9][hh];
  long comp[80];
  real *dest[80];
  while (com != NULL) {
    if ((com->dest_level == levdest)   &&   (com->src_level == levsrc)   &&   (type ==
    GHOST)) {
      if (com->CPU_src != com->CPU_dest) {
        if (com->CPU_src == sourceCPUrank) {
          MPI_Send (com->tracerbuffer, nvar, MPI_DOUBLE, com->CPU_dest, com->tag,
    MPI_COMM_WORLD);
          nr_req++;
        }

        if (com->CPU_dest == destCPUrank) {
          MPI_Recv (com->tracerbuffer, nvar, MPI_DOUBLE, com->CPU_src, com->tag,
    MPI_COMM_WORLD, &stat);
          nr_req++;
        }
      }
    }
    com=com->next;
  }
}
```

Listing 16: The `Tracer_ExecComm()`, in which we call MPI-routines to communicate the tracers' data between the CPUs.

```
void Tracer_CPUnumber
```

Listing 17: .

```
void Tracer_Communication_UpLev(type, point_Q, hh)
  long type;
  real ***point_Q;
  long hh;
{
  /* [...] */
  long lev = point_Q[1][10][hh];
  long l,nvar=11;
  com->tracerbuffer = (real*)prs_malloc (nvar * sizeof(real));
  while (com != NULL) {
    if ((com->dest_level == lev+1) && (com->src_level == lev) && (type == GHOST)) {
      if (com->CPU_src == CPU_Rank) {
        for (l=0; l<nvar; l++) { com->tracerbuffer[l] = point_Q[1][l][hh]; }
      }
    }
    com = com->next;
  }
  Tracer_ExecComm (GHOST, nvar, hh, point_Q);
}
```

Listing 18: The `Tracer_Communication_UpLev()` function for communciation between CPUs when the tracer flows from a certain refinement level to the next finer one. The `Tracer_Communication_DownLev()` is implemented in an equivalent way.

3.6 Communication Between the Grid Levels

TODO

Table 1: List of all parameter set ups used in this work. Note that the number of cells in azimuthal direction refers to the simulation with planet mass $M_P \neq 0.0$ (in code units), i.e. once the CSD is in hydrostatic equilibrium. We use the parameter file 30au1jup.par. All simulations are in three spatial dimensions, `NDIM = 3`. We set `TRACER_MASS` to `0.0`, unless the simulation runs with massive tracers, in which case we set `TRACER_MASS` to `1.0`. The mass and size of the tracer is defined in the code itself.

Reference	Section	NTOT	TRACERPARTICLE	TRACER_MASS	NrTPs	TRACER_R_MIN	TRACER_R_MAX	TRACER_COL_MIN	AMR	CPUs	nr. *az*-cells	nr. *r*-cells	nr. *col*-cells	M_P
1A	4.1	100	-	-	-	-	-	-	1	1	320	120	20	10^{-3}
1B	4.1	100	NO	0.0	0	0	0	0	1	1	320	120	20	10^{-3}
2A	4.2	100	YES	0.0	320	100	100	60	1	1	80	120	10	10^{-3}
2B	4.2	100	YES	0.0	320	100	100	60	1	3	80	120	10	10^{-3}
3A	4.3	100	YES	0.0	320	100	100	60	1	1	80	120	10	10^{-3}
3B	4.3	100	YES	0.0	320	100	100	60	3	1	80	120	10	10^{-3}
4A	4.4	200	YES	0.0		100	100	60	3	3	200	120	20	10^{-4}
4B	4.4	200	YES	0.0		100	100	60	3	3	200	120	20	10^{-3}
4C	4.4	200	YES	0.0		100	100	60	3	3	200	120	20	10^{-2}
5A	4.5	130	YES	1.0	320	100	100	60	1	3	80	120	10	10^{-4}
5B	4.5	130	YES	1.0	320	100	100	60	1	3	80	120	10	10^{-3}
5C	4.5	130	YES	1.0	320	100	100	60	1	3	80	120	10	10^{-2}

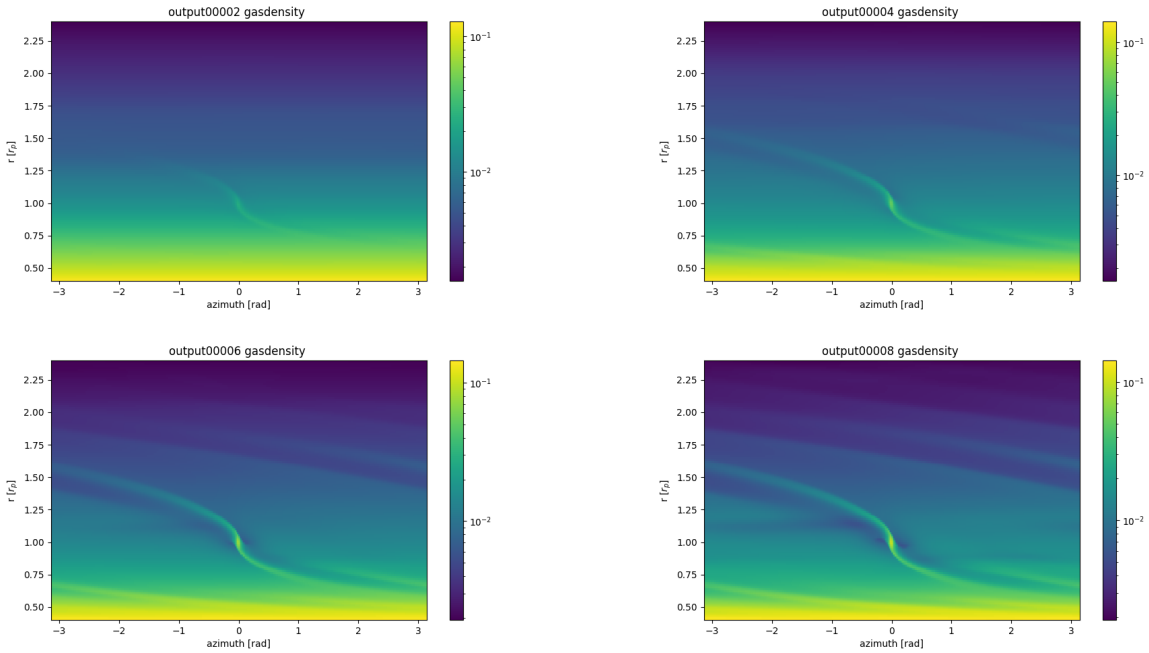

Figure 32: Snapshot results of the **1A** simulation (JUPITER code version not including the tracer particle implementation), see table 1 for the further details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

4 THE RESULTS

DONE In this section, we show the results of the simulation for several different parameter set ups and the results of our test runs, including the compatibility for multiple CPUs or grid refinement levels. Table 1 lists the various parameter set ups that were used in this work. For each simulation, a video is provided under [78] with the same reference name from table 1. In the following, the figures are snapshots from these results. Unless specifically mentioned otherwise, we show the density profile. Furthermore, we show only results after the CSD is in hydrostatic equilibrium, i.e. when $M_P \neq 0.0$ holds, since the tracer particle code is only relevant for this part of the simulation. Throughout the simulations, we work with the parameter file 30au1jup.par.

4.1 Test 1: Compatibility for TRACERPARTICLE == FALSE

For this test, we first run a version of the code, which does not include the files of the tracer particles, see **1A** in table 1, and in the next step run a newer version of the JUPITER code, which includes the files containing the tracer particles (we run this code with TRACERPARTICLE == FALSE) and the same parameter set up as before, see **1B** in table 1. We compare the simulation outputs for conformity. In order to reduce the running time, we wait only 8 outputs with planet mass $M_P \neq 0.0$ and $M_P = 0.0$, and reduce the number of cells in radial direction (with respect to the runs **4** and **5**).

In figure 32, we show the results of the simulation which does not include the tracer particle files, **1A**.

Similarly, figure 33 shows the results of simulation **1B**, where we run the code with the same parameter set up but now on a earlier version of code, which does not include the tracer particles.

Comparing the results of the test runs **1A** and **1B** in figures 32 and 33 at each time-step, we can verify the compatibility of the tracer-implementation with the rest of the JUPITER code. Since the two simulations result in the same density profile, we have ensured that the code including the tracer particles gives the same outputs as the version without the tracer implementation.

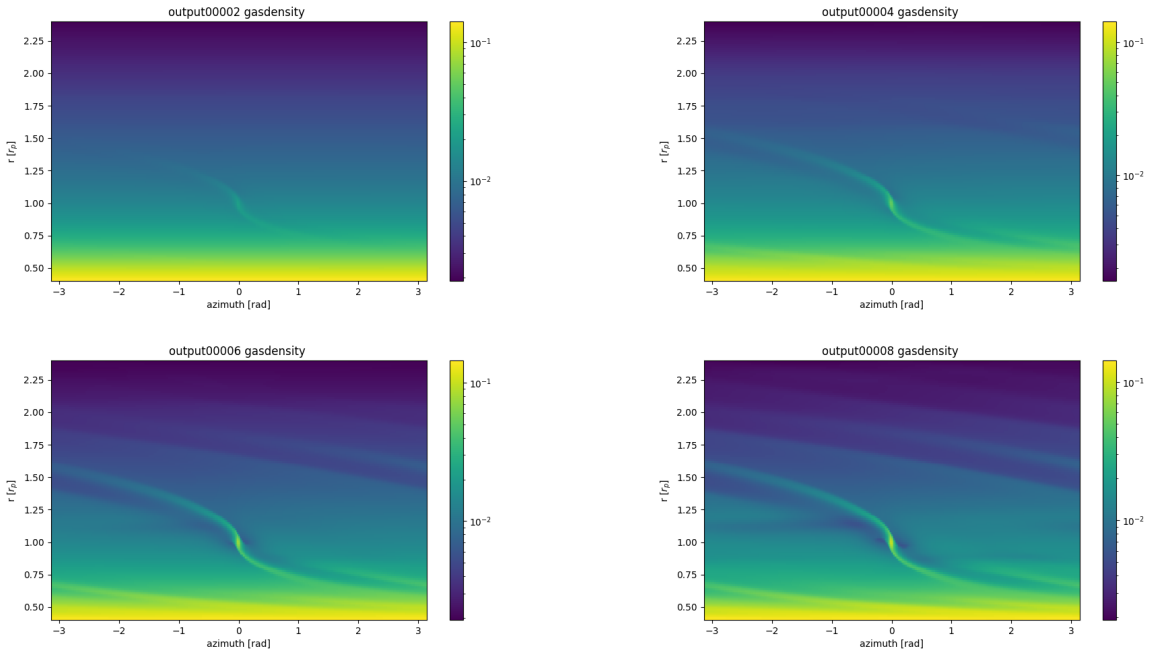

Figure 33: Snapshot results of the **1B** simulation (JUPITER code version including the tracer particles, setting TRACERPARTICLE == NO), see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

4.2 Test 2: Compatibility for Multiple CPUs

In order to test whether the code runs correctly on several CPUs, we run the code first on one CPU only (simulation **2A** in table 1) and on three CPUs (**2B** in 1), using otherwise the same parameter set up as before. To check the validity, the two simulations must agree at a certain output level, i.e. the position and velocities of the tracer particles must match. The two results are shown in the following. We note that the re-positioned tracer, see Section 3.3.9 or figure 31, is advected with the gas fluid after we have changed its position. Otherwise, this tracer would remain at this position for the rest of the simulation. In figure 34, we see that this is not the case.

4.3 Test 3: Compatibility for Multiple Grid Refinement Levels

With simulations **3A** and **3B** we test the compatibility of the code including the tracer particles when running with multiple refinement levels of the grid. To do so, we run the code with one level only in **3A** and with the same parameter set up but three levels of refinement in **3B**.

4.4 Tracing the Meridional Circulation with Massless Tracer Particles

Hereafter, we show the simulation outputs in the case of massless tracers for several different parameters, simulations **4A**, **4B** and **4C** in table 1.

4.5 Tracing the Meridional Circulation with Massive Tracer Particles

Finally, in this section, we show the simulation results of the meridional circulation with massive tracer particles and compare the difference to the results of Section 4.4, where we tracked the flow with massless tracers[48], see simulations **5A**, **5B** and **5C** in table 1. We examine the influence of the planet's mass on the massive tracers by choosing different planet masses, ranging form a massive planet with $M_P = 10^{-2} M_\odot$ in 49 and 50, to a low mass

[48] Note, that for this comparison, we use the same parameter set up as in the previously mentioned section.

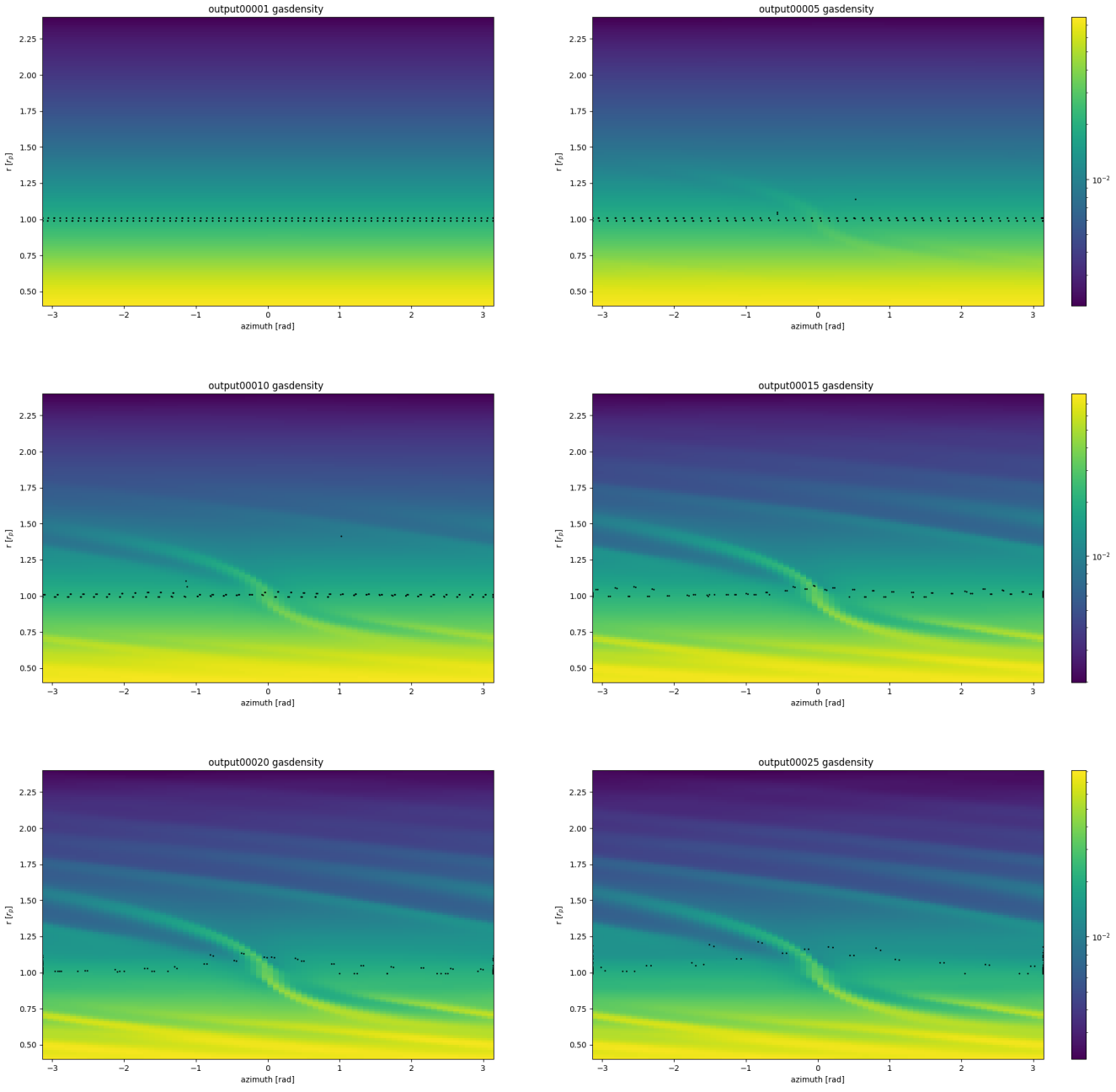

Figure 34: Snapshot results of the **2A** simulation (1 CPU and TRACERPARTICLE == YES), see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times in the $r = r(\theta)$ plane.

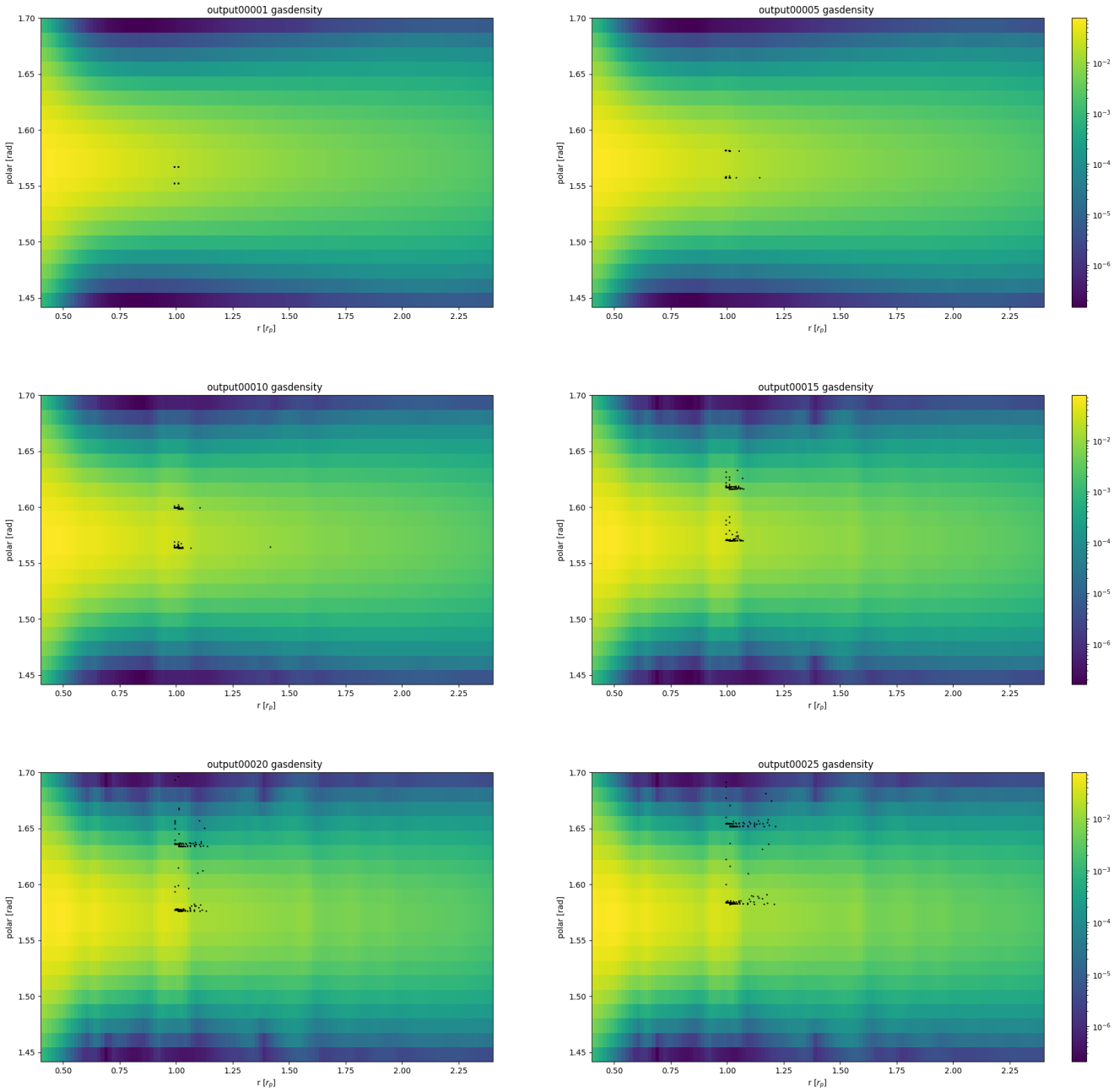

Figure 35: Snapshot results of the **2A** simulation (1 CPU and TRACERPARTICLE == YES), see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times in the $\phi = \phi(r)$ plane.

SIMULATION_RESULTS/Simulation_adiab_3CPU/postPlanet/gasdensity2.png

SIMULATION_RESULTS/Simulation_adiab_3CPU/postPlanet/gasdensity5.png

Figure 36: Snapshot results of the **2B** simulation (3 CPU and TRACERPARTICLE == YES), see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

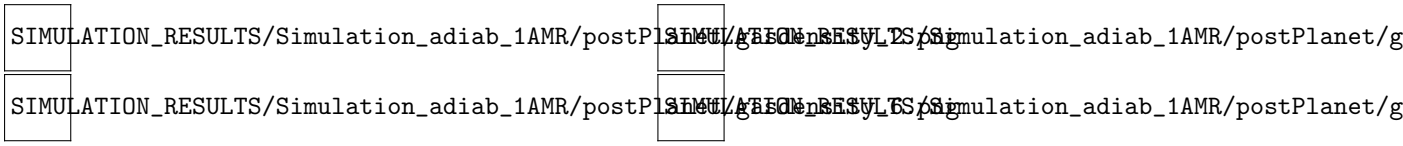

Figure 37: Snapshot results of the **3A** simulation (1 level of grid refinement and TRACERPARTICLE == YES), see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

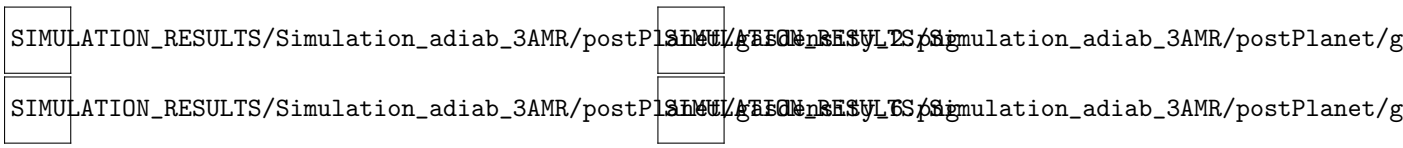

Figure 38: Snapshot results of the **3B** simulation (3 levels of grid refinement and TRACERPARTICLE == YES), see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

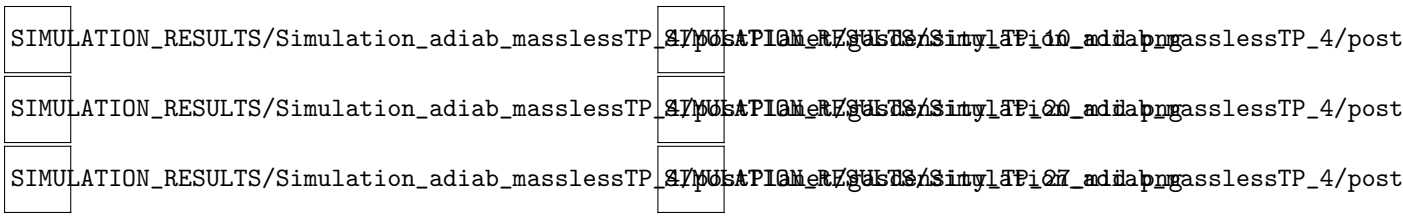

Figure 39: Snapshot results of the **4A** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

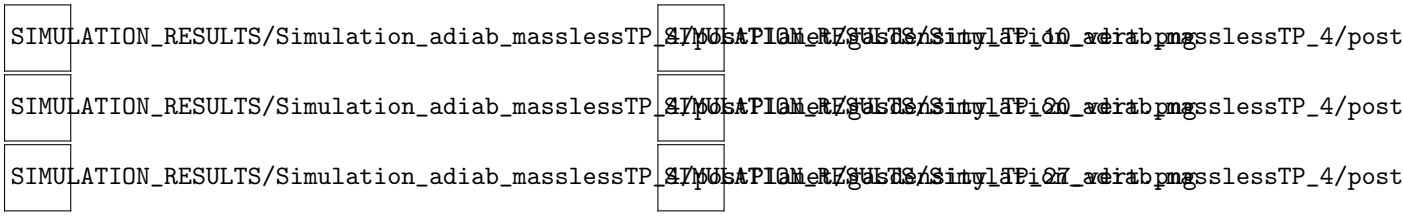

Figure 40: Snapshot results of the **4A** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

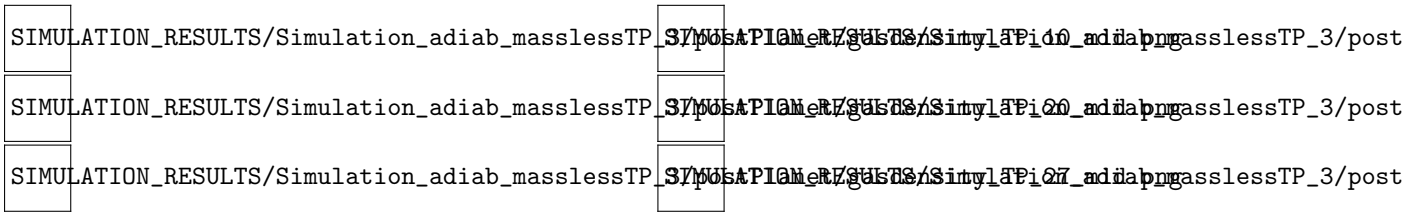

Figure 41: Snapshot results of the **4B** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

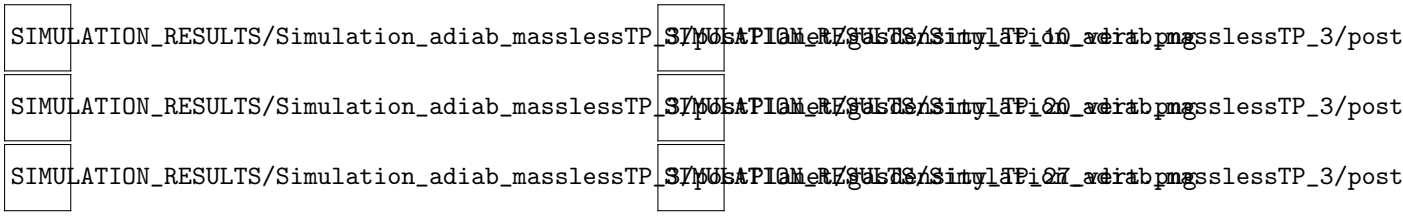

Figure 42: Snapshot results of the **4B** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

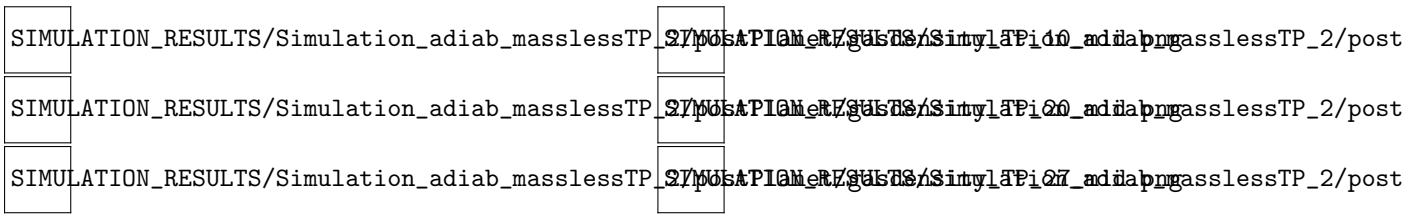

Figure 43: Snapshot results of the **4C** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

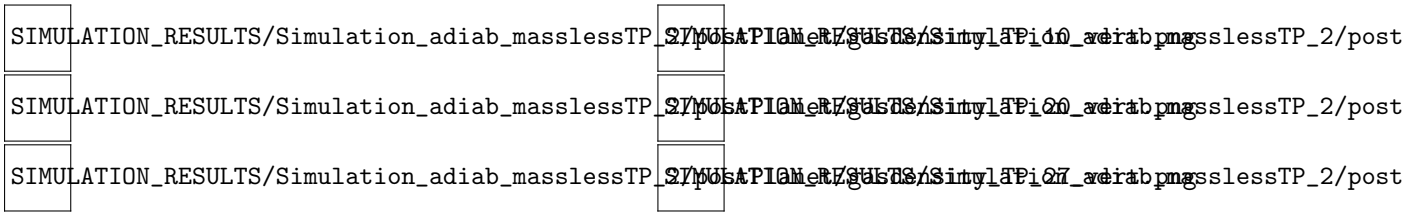

Figure 44: Snapshot results of the **4C** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times.

planet with $M_P = 10^{-4} M_\odot$ in 45 and 46.

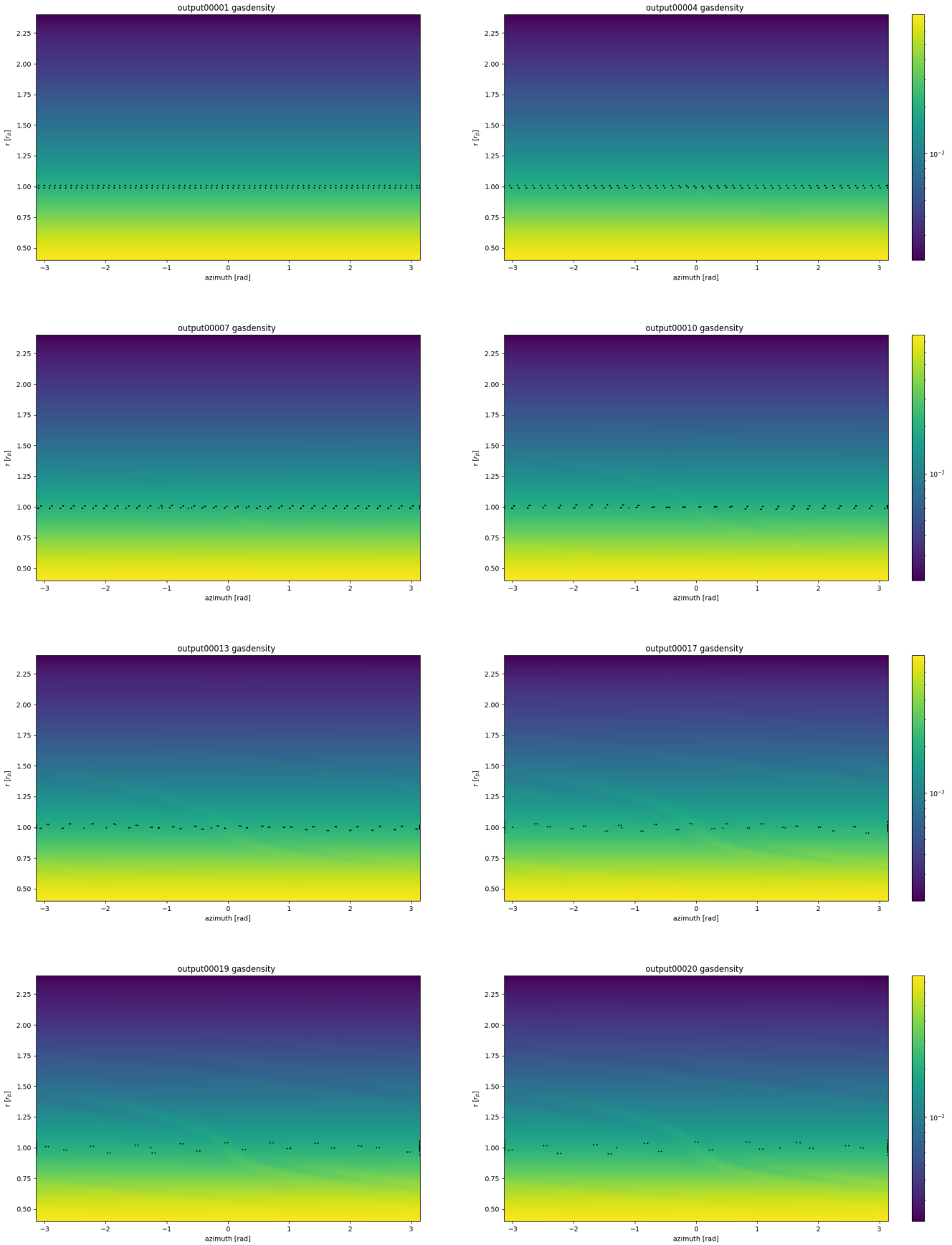

Figure 45: Snapshot results of the **5A** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times in the $r = r(\theta)$ plane.

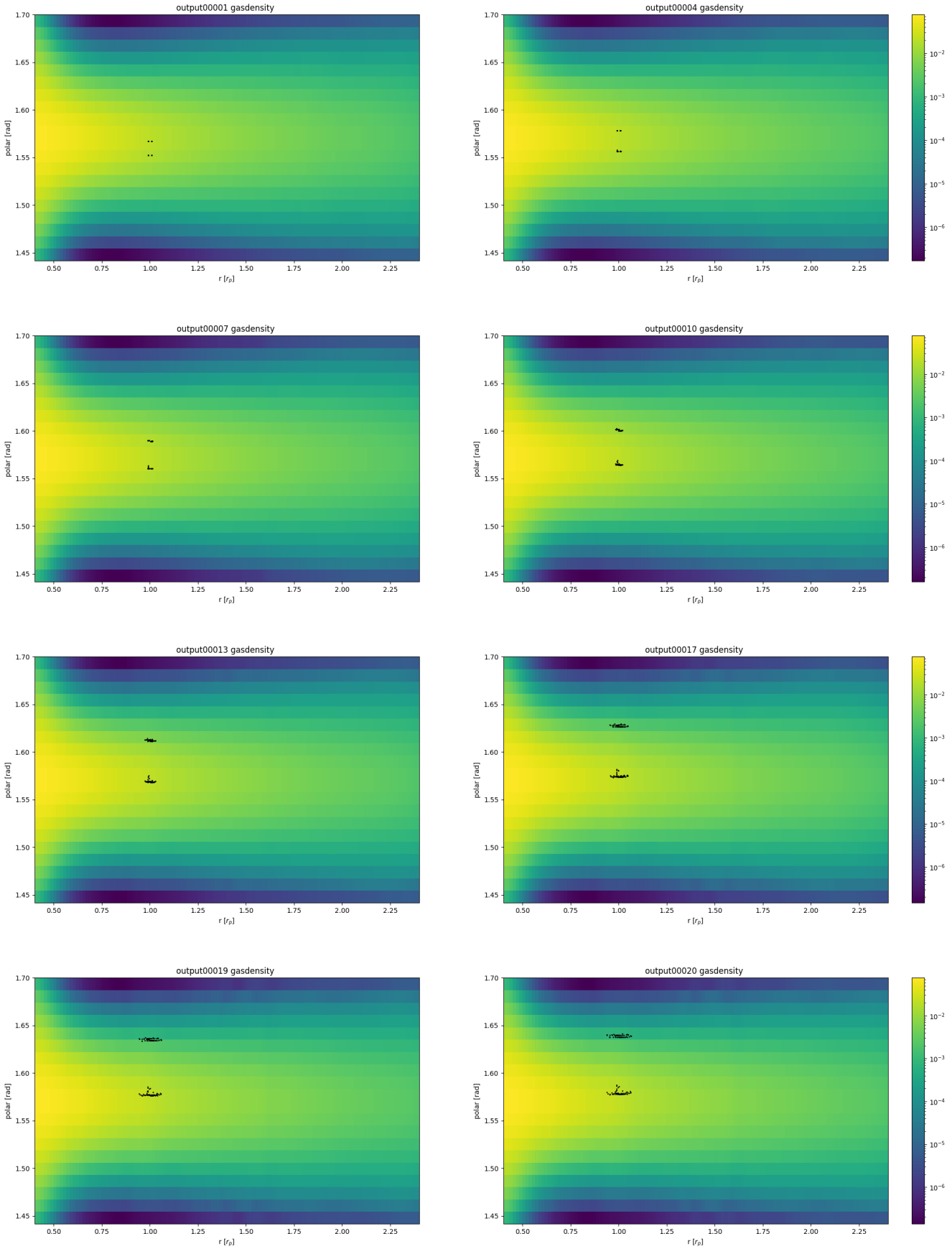

Figure 46: Snapshot results of the **5A** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times in the $\phi = \phi(r)$ plane.

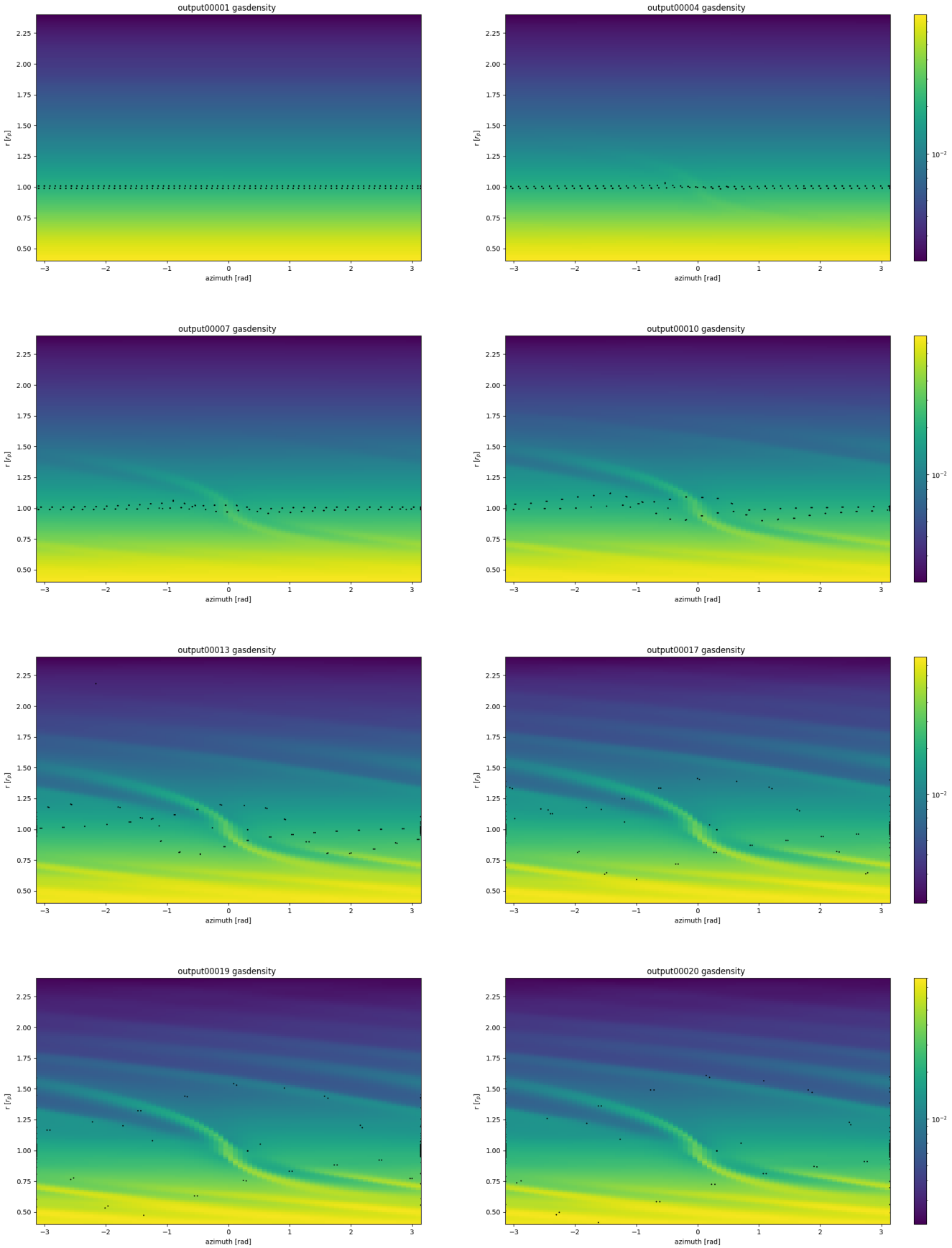

Figure 47: Snapshot results of the **5B** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times in the $r = r(\theta)$ plane.

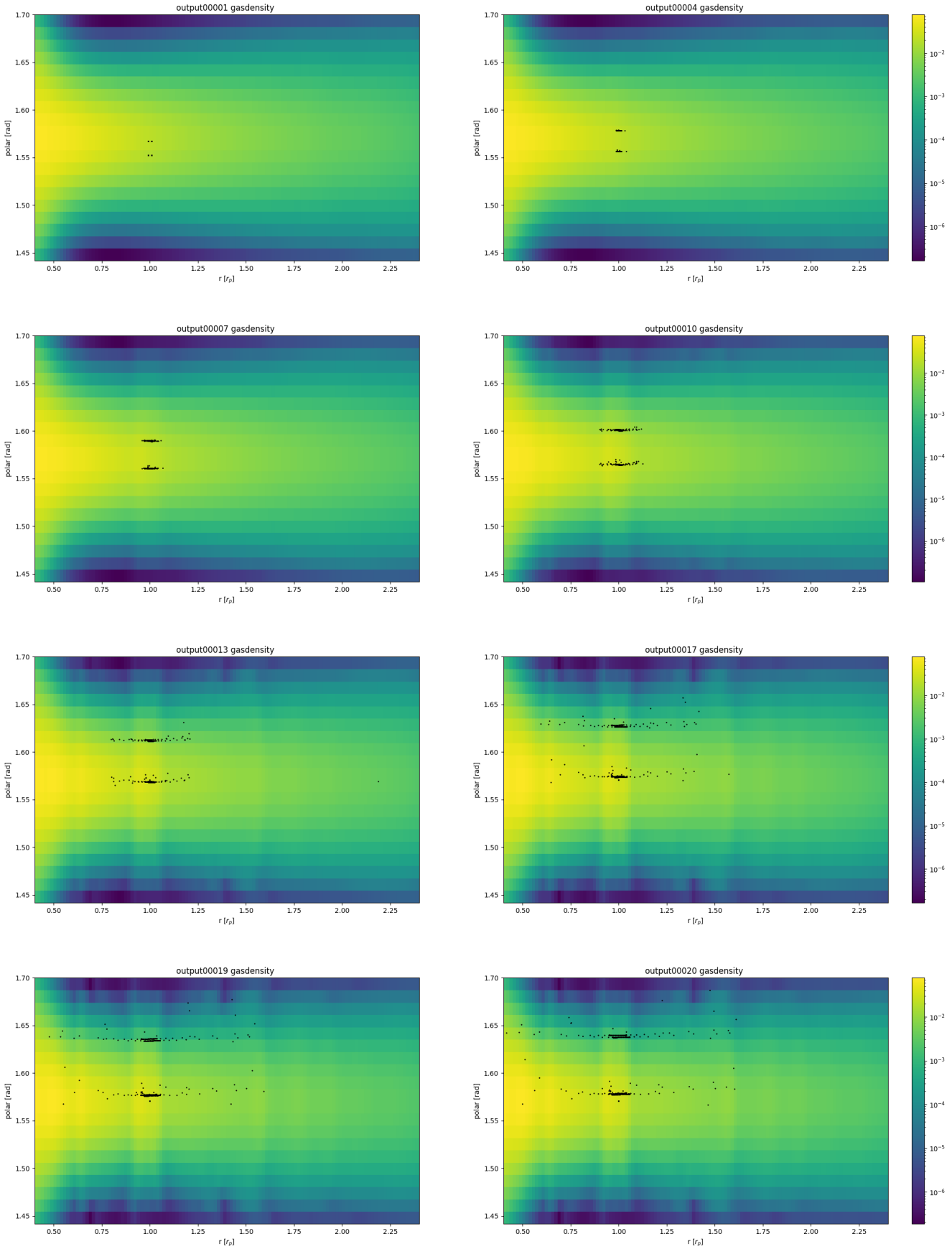

Figure 48: Snapshot results of the **5B** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times in the $\phi = \phi(r)$ plane.

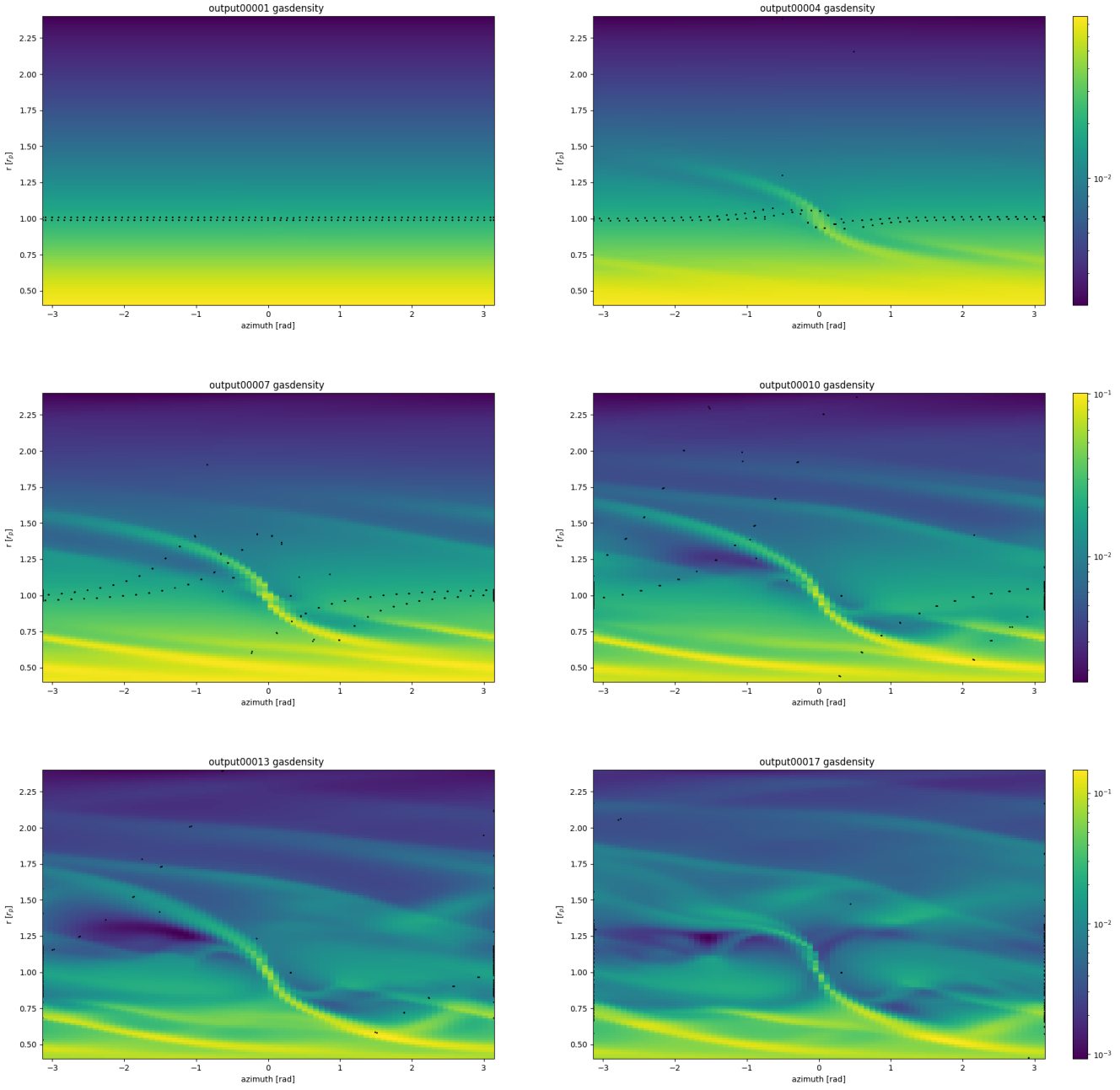

Figure 49: Snapshot results of the **5C** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times in the $r = r(\theta)$ plane.

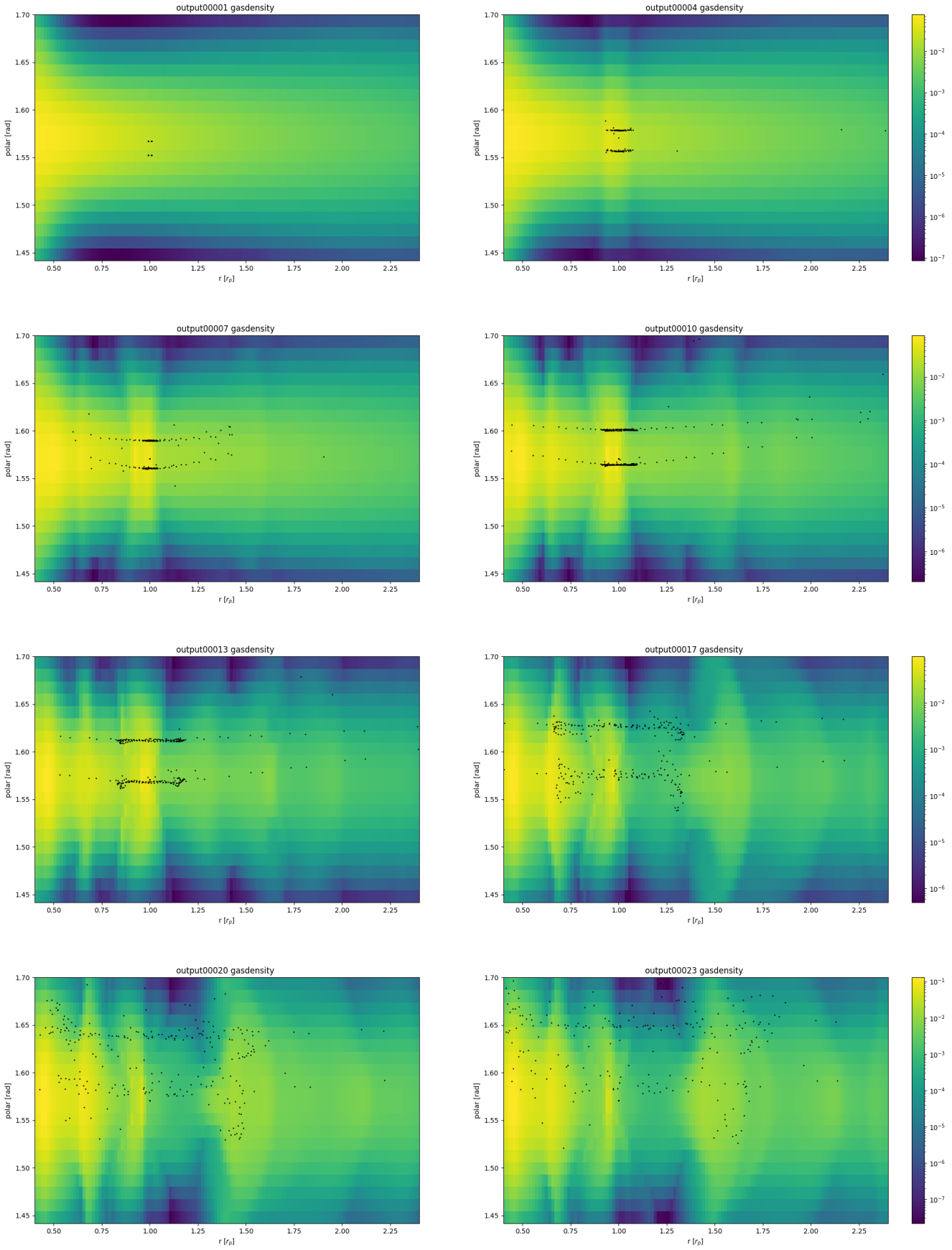

Figure 50: Snapshot results of the **5C** simulation, see table 1 for the details and [78] for the complete simulation output in video format. The figure shows the gas density at different output times in the $\phi = \phi(r)$ plane.

5 DISCUSSION AND CONCLUSIONS

DONE In this section, we discuss the simulation outputs from Section 4.

5.1 Test 1: Compatibility for TRACERPARTICLE == FALSE DONE

A comparison of the results in figure 32 and 33, we can easily see that the output in both versions is equal, which is due to the usage of the `if`-condition

```
if (TracerParticle == Yes)
```

ensuring that we execute the tracer particle-related functions if and only if the user previously sets the global variable TRACERPARTICLE to `True`[49]. Otherwise, we skip this additional computation.

5.2 Test 2: Compatibility for Multiple CPUs

TODO The figures 34 from Section 4.2 show that the tracer particles close to the position of the planet are pushed aside towards larger radii during the formation of the planet and the resulting spiral wake. On the other hand, the tracers that are initially placed at larger azimuthal angles, i.e. on the "other side" of the central star in the CSD, do not show the same amount of radial displacement. Figure 35 show that the gas flows towards larger colatitude and radial co-ordinates (note that we simulate the upper hemisphere of the star's environment). Both results agree with the discussion in Section 1.6 and figure 10 on the meridional circulation, namely that that gas spirals outwards in the equatorial plane (resulting in increasing radial components of the tracers), followed by an upward directed motion (increasing the tracers colatitude co-ordiante). Under certain conditions, the gas flows towards the planet again via the poles of the planet. We discuss the trajectories of the tracers more in Section 5.5 for massless particles and in Section 5.5 for massive ones. Hereafter, we focus on the comparison between the two outputs 34 and 36.

5.3 Test 3: Compatibility for Multiple Grid Refinement Levels

TODO 37 38

5.4 Massless Tracers

TODO In this section, we discuss the results from Section 4.4.

Figures 39 and 40 show the $r = r(\theta)$ and $\phi = \phi(r)$ planes with a planet of mass $M_P = 10^{-4} M_\odot$. todo

Figures 41 and 42 show the $r = r(\theta)$ and $\phi = \phi(r)$ planes with a planet of mass $M_P = 10^{-3} M_\odot$. todo

In case of a massive planet, $M_P = 10^{-2} M_\odot$, figures 43 and 44 show that the todo

5.5 Massive Tracers

TODO All simulations with massive tracers are computed in the same way as the massless tracers, with an additional coupling function that takes care of the massive particles being influenced by the gas, similar to how it has been done with the dust fluid, see [54]. This means, that we do not include an N-body solver for the massive tracers. Furthermore, we do not include self-gravity of this component, i.e. the massive tracers have an initial, user-defined mass, that does not change throughout the simulation, implying that the tracers do neither accrete[50], nor loose mass[51]. Furthermore, the tracers do not accrete onto the central star nor onto the planet, which can be seen throughout all the simulations **5A** - **5C** in Section 4.5: the number of tracers remains constant during the

[49] Note, that in the parameter file, we use `True` and `False` for the tracers, whereas throughout the code itself, we use the boolean expressions `Yes` or `No`, respectively.

[50] For example by accreting dust or other smaller tracers

[51] One possible mechanism could be the collision of two massive tracers.

simulation and only depends on how large the initial tracer volume is defined (see corresponding notes in section 7).

We start the discussion of the simulation outputs with massive tracers with the figures 47 and 48, in which we use a planet of Jupiter size, $M_P = 10^{-3} M_\odot$. todo

The runs with a low mass planet $M_P = 10^{-4} M_\odot$ show todo

In case of a massive planet, $M_P = 10^{-2} M_\odot$, figures 49 and 50 show that the overall motion of the tracers is more active and violent in comparison to the motion in the vicinity of a lower mass planet. This is a plausible result, since a more massive planet produces stronger spiral arms and thus the gaseous environment of the planet is more turbulent than in case of a low mass planet. Furthermore, such a planet is capable of producing gaps in the CSD, see Section 1.7. Figure 50 furthermore shows the evolution of the meridional circulation as expected from the discussion in Section 1.6: in `output00001`, the tracers are still very close to their initial position. They then move in radial direction away from their initial positions, `output00004`. At larger and smaller radii, $r \sim (0.75,\ 1.25) R_P$ an upward motion of the gas is apparent: the tracers' colatitude co-ordinates increase, `output00007` and `output00010`. The motion in colatitude direction becomes more and more active at later stages in the simulation, and the tracers flow again towards their initial radial position at higher and lower colatitude, `output00020` and `output00023`. Because of the problem mentioned in Section 3.3.8, figure 49 unfortunately only helps analysing the meridional circulation in the beginning of the simulation. Compared to simulations with massive tracers around a lower massive planet, 45 and 47, the tracer here have a significantly larger radial co-ordinate when flowing outwards (smaller, when flowing inwards) than at the same time step. This underlines again the effect of the planet's mass on the evolution of the surrounding gas flow. It further highlights that the interaction between gas in the CPE / CPD and gas in the CSD has a greater range, which can be explained by the increasing size of the Hill-sphere (**??**) for increasing planet mass.

6 SUMMARY

TODO In this work, we have discussed how tracer particles have been introduced to the JUPITER code, aiming to help tracing the meridional circulation of the gas fluid around a (gaseous) planet in the CSD. We have worked on the adiabatic version of the code rather than the isothermal version, for reasons discussed in sections 1.5 and 1.6. We have used the paper [28] by Lissauer et al. as a guidance for the implementation:

- We set up the initial distribution of the tracers, such that each cell whose centre lies within a torus-like volume around the central star, with mid radius equal to the planet radius, see equations (58)-(60). We write this process in a separate function, described in 3.3.4.

- For these tracer particles, we solve the equation of advection (57). For that purpose, we first determine the updated velocities using Theorem 4, the monotonized harmonic mean method. In the next step, the position of the tracers is updated using Theorem 4, the Runge-Kutta algorithm.

- We perform several test runs in order to ensure that the code works without the tracer part as well as on multiple CPUs and on a grid with multiple levels of refinement. The results are shown in section 4.1 - 4.3. For the test runs, we use a small grid to reduce the computational time, see table 1.

- We perform two sets of runs with tracer particles, massless and massive. The results can be found in 4.4 and 4.5, respectively. We use vary the planet mass in order to examine the planet's influence on the meridional circulation and especially the accretion of gas onto the circumplanetary region. For both sets, we increase the number of cells in the three directions to obtain more refined results.

The main results of this work are listed in the following:

- One
- Two
- Three

7 OUTLOOK

DONE Next efforts regarding the implementation of the tracer particles should be devoted to:

- the inclusion of the self gravitational effect of the massive tracer particles, in order to visualise the formation and clumping of planetesimals etc. In this scenario, we would have to implement a variable tracer mass instead of a fixed one. Dr. A. Prabhu is currently working on the implementation of self-gravitation in the JUPITER code. Furthermore, they should grow in the same way as the planet, rather than be implemented with their final mass from the very beginning. This would include the implementation of mass accretion onto the planetesimals.

- With the previous mentioned changes, the growth of planetesimals and thus their final mass needs to satisfy equation (84), i.e. low mass planetesimals are much more common than high mass objects.

- the implementation of tracer particles, which have initially different masses. With this simulation, we could more realistically visualize the object flow in the planet's surroundings, as there would be particles of pebble size up to planetesimal size. Including the above mentioned self-gravitation, a good tracking of the meridional circulation and mass accretion onto the planet could be simulated. A first idea could be to introduce the tracer particles whose masses satisfy a user defined function, for example a Gaussian distribution (or a superposition of several such functions)

$$f(m_{TP}) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot exp\left(-\frac{1}{2}\left(\frac{m_{TP}-\mu}{\sigma}\right)^2\right),$$

  where m_{TP} is the tracer mass. The parameters σ and μ would then be global variables, which the user would have to define.

- the implementation of collisions between massive tracer-tracer or massive tracer-planet. In the first case, we would have to discuss the conditions under which two massive tracers merge or destroy each other; in the second situation the massive tracer can accrete onto the planet. In both cases, the total number of tracers is not constant.

- the inclusion of an additional external potential onto the planet due to the presence of massive tracers in the CSD. This potential has to depend on the number of traces and their masses. In case the we include collisions between tracers and thus a non-constant number of tracers, this expression however could become too complicated and probably even unnecessary detailed.

- the implementation of a user defined initial distribution of the tracer particles. In the current version, the tracers are initially located at the centres of the cells, forming a torus-like volume around the central star. A different implementation could include a distance-dependent distribution, such that in the inner disk ($r < r_P = 1AU$) we have a significantly less tracers than in the outer CSD ($r > r_P = 1AU$); or a random distribution as it was used in [28].

- It should not be too great of a task to adjust the current version of the tracer particles to run with the inclusion of the dust fluid as well. The formation of the gap as described in Section 1.7 could be traced as well.

- A further interesting task would be to track the fluid's motion in case of a CSD with multiple gaseous planets. In this situation, we would have to discuss the situation where one of the planets migrates towards the other planet such that the initially distinct meridional circulation volumes overlap. A more complex flow would arise. In a next step, this code could then be adjusted to track the flow around two merging supermassive black holes: each SMBH would have its own disk but they would furthermore be surrounded by a circumbinary disk.

- As briefly mentioned in 1.3.2.2, the accretion of material occurs at different rates in different states of the planet's evolution in growth. For massive tracers that grow over time, this could be accounted for as well.

8 ACKNOWLEDGMENTS

The computations has partially been carried out on the ETH Zurich's Euler machine.
The author thanks Prof. Dr. Judit Szulágyi, ETH Zurich, for the opportunity to work on a simulation and to gain more insight in the academic everyday life. Her advices greatly helped me.
A special thanks to Dr. Ameya Prabhu, ETH Zurich, for the mental support as well as his generous support and patience, answering all my questions regarding the JUPITER code and its internal structure. I learnt a lot during our meetings.
I would like to thank Fabian Binkert, Ludwig-Maximilians-Universität München, for the amazing tutorials on how to run the JUPITER code.
Last but not least, I would like to thank Roman Gruber, ETH Zurich, for his help in the fight against the oddities of *Windows* as well as answering my questions regarding C and parallel programming.
This work would not have been finished without the help of them.

9 ACRONYMS

AMR	adaptive mesh refinement
CFL	Courant-Friedrichs-Lewy condition
CIC	cloud-in-cell method
CoM	centre of mass
CPD	circumplanetary disk
CPE	circumplanetary envelope
CSD	circumstellar disk
EoM	equation of motion
FoR	frame of reference
GFO	is a piecewise constant predictor method, diffusive with error in first order in space
HR	Hertzsprung-Russell (-diagram)
HST	Hubble Space Telescope
ISM	Interstellar matter
JWST	James Webb Space Telescope
LHS	left-hand side
MMSN	minimum mass solar nebula
MPI	Message Passing Interface
MUSCL	predictor method for hydrodynamic quantities, second order accurate in space
PLM	piece-wise linear method to predict hydrodynamic quantities, second order accurate in space
RHS	right-hand side
SED	Spectral Energy Distribution
SMBH	Supermassive Black Hole
SPH	smoothed particle hydrodynamics
VLT	Very Large Telescope
YSO	Young Stellar Objects

10 APPENDIX

10.1 Derivation of the Virial Theorem

This derivation is based on the lecture notes of *Astrophysics II* by Prof. Refregier from the Spring Semester 2021, ETH Zurich[52].
Starting from the collisionless Boltzmann equation for a distribution function $f = f(x,v,t)$,

$$0 \stackrel{!}{=} \frac{df}{dt} = \frac{\partial f}{\partial t} + \sum_i \left(\frac{\partial f}{\partial x_i} \cdot v_i - \frac{\partial f}{\partial v_i} \cdot \frac{\partial \Phi}{\partial x_i} \right),$$

where x_i are the positions, v_i the velocities with $v_i = \frac{\partial x_i}{\partial t}$, and the potential Φ.
In the steady state, this equation reduces to

$$0 = \frac{df}{dt} = \sum_i \left(\frac{\partial f}{\partial x_i} \cdot v_i - \frac{\partial f}{\partial v_i} \cdot \frac{\partial \Phi}{\partial x_i} \right)$$

We define $f := n \cdot < v_j >$, where n is the number density of particles, $n \cdot m = \rho$ (m the mass and ρ the density), and multiply both sides of the equation with

$$- \int d^3x m \cdot x_k$$

to get

$$0 = - \int d^3x \sum_i \left(m x_k \frac{\partial (n < v_j >)}{\partial x_i} \cdot v_i - m x_k \frac{\partial (n < v_j >)}{\partial v_i} \cdot \frac{\partial \Phi}{\partial x_i} \right).$$

We call the first term A and the second term B. For A, we continue with

$$\begin{aligned} A &= - \int d^3x \sum_i \left(m x_k \frac{\partial (n < v_j >)}{\partial x_i} \cdot v_i \right) \\ &= - \int d^3x \sum_i \left(\frac{\partial (m n x_k < v_i v_j >)}{\partial x_i} - \frac{\partial x_k}{\partial x_i} m n < v_i v_j > \right) \\ &= - \int d^3x \sum_i \left(0 + \delta_{i,k} \rho < v_i v_j > \right). \end{aligned}$$

In the first equation, we have used the reversed chain rule; in the second equation we have applied Green's theorem on the first term. We then find

$$0 = + \int d^3x \rho < v_i v_j > := 2T_{j,k}.$$

For the term B, we find

$$\begin{aligned} B &= - \int d^3x \sum_i \left(m x_k \frac{\partial (n < v_j >)}{\partial v_i} \cdot \frac{\partial \Phi}{\partial x_i} \right) \\ &= - \int d^3x \sum_i \left(\rho x_k \frac{\partial < v_j >}{\partial v_i} \cdot \frac{\partial \Phi}{\partial x_i} \right) \\ &= - \int d^3x \sum_i \rho x_k \delta_{i,j} \frac{\partial \Phi}{\partial x_i} \\ &:= -W_{j,k}, \end{aligned}$$

where we have defined $W_{j,k}$ the Chandrasekhar potential energy tensor, see equation (55) in [90]. In the first equation, we have used that $n = n(x,t)$ is independent of the velocities. Combining the two expressions for A and B, we find

$$0 = 2T_{j,k} + W_{j,k}$$

and taking the trace on both sides of the equation, we find the expression for the Virial theorem:

$$0 = 2T + V. \tag{87}$$

[52] This lecture was based on [89]. For the derivation of the Virial Theorem, see Section 5.4.4. in this book.

10.2 Derivation of Equation (72)

Starting from the 2^{nd}-order Runge-Kutta equation (67),

$$x_{n+1} = x_n + \frac{dt}{2} \cdot (v_n + v_{n+1}),$$

and insert equation (68) we find:

$$\begin{aligned} x_{n+1} &= x_n + \frac{dt}{2}\left(v_n + v_n + \frac{2(x_{n+1}-x_n)}{\Delta} \cdot max\left(0, \frac{(v_n - v_L)(v_R - v_n)}{v_R - v_L}\right)\right) \\ &= x_n + v_n dt + \beta \frac{x_{n+1} dt}{\Delta} - \beta \frac{x_n dt}{\Delta}. \end{aligned}$$

Subtracting the third term from the RHS from both sides, we find

$$x_{n+1} \cdot \left(1 - \frac{\beta dt}{\Delta}\right) = \left(x_n + v_n dt + \beta \frac{x_n dt}{\Delta}\right)$$

and dividing by the coefficient on the LHS, the resulting updated position is given by

$$x_{n+1} = \left(x_n + v_n dt + \beta \frac{x_n dt}{\Delta}\right) \cdot \left(1 - \frac{\beta dt}{\Delta}\right)^{-1},$$

where we defined $\beta = max\left(0, \frac{(v_n - v_L)(v_R - v_n)}{v_R - v_L}\right)$ and Δ is the distance between two neighbouring cells. Note, that this distance varies in general in each direction depending on how the grid is set up.

10.3 Command Routine Including Tracers

In tables 2 and 3, we give two examples how to run the code with the tracer simulation, the first one with massless tracers, the second one with non-trivial mass. In each case, the first command is the ensure that the CSD is in hydrostatic equilibrium. The second command is to merge the outputs and is only required if the simulation is performed on multiple CPUs. In the next step, we adjust the grid sizes and insert a planet. At this point, we need to set `TRACERPARTICLE == True`. The second last command performs the simulation with the tracers. Note, that for this final part, a different output directory is required, see 3.3.11. The last command merges the final outputs for the gas fluid and creates a new text file for the tracers, that contains all data from all CPUs. These files are used to create the figures. As mentioned in Section 3.3.12, in case the simulation runs sequential, we have to make a slight adjustment in the `pyJupiter.py` file and then this step is only required when multiple CPUs are used.

In case the simulation runs with many tracers, i.e. `TRACER_R_MIN`, `TRACER_R_MAX` and `TRACER_COL_MIN` are small, the computational time becomes very large. This can be avoided, by changing `TRACER_INCR_K`, `TRACER_INCR_J` and `TRACER_INCR_I`, which are the increments in the three `for`-loops that iterate over all cells in the grid. If all three parameters are set to `1`[53], every cell in the volume defined by (58) - (60) has initially a tracer. By changing for example `TRACER_INCR_I` to `2`, we place initially a tracer in every other cell in the first spatial direction (which is usually the azimuthal direction). This method reduces the number of tracers while still covering the user defined volume.

Finally, we mention a few things that need to be respected when running the code with tracer particles:

1. Even though we tried to ensure that the code runs with different co-ordinate systems, such as Cartesian and cylindrical, this part has been developed using spherical co-ordinates and it is likely that certain functions are not correctly working on other systems. The same holds true for other co-ordinate permutations than (213).

[53] The increments should not be chosen smaller than `1`!

2. The tracer implementation runs only with a co-rotating grid, i.e. `OMEGAFRAME = 1.0004998750624609`.

3. As seen in tables 2 and 3, the tracer part is only executed when the output directory's name includes the word `post`, referring to *after* the grid has been stretched. It is therefore crucial to use different output directory names for the whole set up than for the actual simulation. This solution was necessary because otherwise the tracer functions were executed before already.

4. We have not included the dust module in this work. Several adjustments would be necessary if the user intends to trace the meridional circulation of both the gas and dust fluid. One of the adjustments would be in `iter.c`, where the tracer main function is called right after the hydro kernel function. If the dust module should be included as well, we would have to include the changes according to the following idea.

```
/* [...] */
if ((Fluid->next==NULL)  &&  (TRACERPARTICLE == NO)){
  HydroKernel (dt);
} else if ((Fluid->next==NULL)  &&  (TRACERPARTICLE == YES)) && (DUST == NO)){
  HydroKernel (dt);
  char *sTring2 = "post";
    if (strstr(OUTPUTDIR, sTring2) != NULL){ TracerFluid(dt); }
} else {
  /* [...] */
  DustKernel (dt);
  if (TRACERPARTICLE == YES){ TracerFluid(dt); }
}
/* [...] */
if(TRACERPARTICLE == NO){ FluidCoupling (item, dt);}
else { GasDustTracerCoupling(item, dt); }
```

   The function `GasDustTracerCoupling()` is not written yet and should advect the tracer particles with the gas and dust, after these two fluids have been coupled. The boolean variable `DUST` does not exist either - and is perhaps not even necessary. Furthermore, a different parameter file has to be used that includes the relevant parameters for both the tracers and the dust.

Massless Tracers.

Table 2: Example routine for massless tracers and 3 CPUs.

Command	TracerParticle	TracerMass	Size1	Size2	Size3	PlanetMass	OUTPUTDIR
`mpirun -n 3 ./jupiter_rad in/30au1jup.par`	No	0.0	2	120	20	0.0	./pre
`./jupiter_rad -m N in/30au1jup.par`	No	0.0	2	120	20	0.0	/pre
`./jupiter_rad -S N in/30au1jup.par`	Yes	0.0	640	120	20	10^{-3}	./pre
`mpirun -n 3 ./jupiter_rad in/30au1jup.par`	Yes	0.0	640	120	20	10^{-3}	./post
`./jupiter_rad -m i`[54] `in/30au1jup.par`	Yes	0.0	640	120	20	10^{-3}	./post

Massive Tracers.

Table 3: Example routine for massive tracers and 3 CPUs.

Command	TracerParticle	TracerMass	Size1	Size2	Size3	PlanetMass	OUTPUTDIR
`mpirun -n 3 ./jupiter_rad in/30au1jup.par`	No	1.0	2	120	20	0.0	./pre
`./jupiter_rad -m N in/30au1jup.par`	No	1.0	2	120	20	0.0	./pre
`./jupiter_rad -S N in/30au1jup.par`	Yes	1.0	640	120	20	10^{-3}	./pre
`mpirun -n 3 ./jupiter_rad in/30au1jup.par`	Yes	1.0	640	120	20	10^{-3}	./post
`./jupiter_rad -m i in/30au1jup.par`	Yes	1.0	640	120	20	10^{-3}	./post

References

[1] M. de Val-Borro, F. Masset, et al., "A comparative study of disc–planet interaction," 2006. https://academic.oup.com/mnras/article/370/2/529/967036.

[2] J. Szulágyi, "Gas Accretion onto Giant Planets," 2015. https://people.phys.ethz.ch/~judits/thesisszulagyi.pdf.

[3] R. M. Hazen, Ph.D. , "The Heliocentric Theory: Nicolaus Copernicus and Galileo Galilei," Accessed 8 Feb. 2022. https://www.thegreatcoursesdaily.com/the-heliocentric-theory-nicolaus-copernicus-and-galileo-galilei/.

[4] Dictionary, Merriam-Webster, "Solar system," Accessed 8 Feb. 2022. https://www.merriam-webster.com/dictionary/solar%20system.

[5] I. Kant, "Universal natural history and theory of the heavens or essay on the constitution and the mechanical origin of the whole universe according to Newtonian principles (1755)," 2012. https://doi.org/10.1017/CBO9781139014380.007.

[6] P. S. Laplace, "The System of the World," 1809. https://archive.org/details/systemworld01laplgoog.

[7] C. O. Lim, "Telescope," Accessed 8. Feb. 2022. https://en.wikipedia.org/wiki/Telescope.

[8] div., "Immanuel Kant," Accessed 8. Feb. 2022. https://en.wikipedia.org/wiki/Immanuel_Kant.

[9] div., "Pierre-Simon Laplace," Accessed 8. Feb. 2022. https://en.wikipedia.org/wiki/Pierre-Simon_Laplace.

[10] National Schools' Observatory, "Arecibo Observatory," Accessed 8. Feb. 2022. https://www.schoolsobservatory.org/learn/eng/tels/groundtel/arecibo.

[11] National Schools' Observatory, "Very Large Telescope (VLT)," Accessed 8. Feb. 2022. https://www.schoolsobservatory.org/learn/eng/tels/groundtel/vlt.

[12] L. L. Christensen, B. Fosbury, M. Kornmesser, "Hubble - 15 Years of Discovery," 2004. ISBN: 0-387-28599-7.

[13] National Schools' Observatory, "Hubble Space Telescope (HST)," Accessed 8. Feb. 2022. https://www.schoolsobservatory.org/learn/eng/tels/spacetel/hst.

[14] National Schools' Observatory, "James Webb Space Telescope (JWST)," Accessed 8. Feb. 2022. https://www.schoolsobservatory.org/learn/eng/tels/spacetel/jwst.

[15] D. Ward-Thompson, A. P. Whitworth, "An Introduction to Star Formation," 2011. Cambridge University Press, ISBN: 978-0-521-63030-6.

[16] H.M. Schmid, "Astrophysics III: Galactic Astronomy," Spring Semester 2021, ETH Zurich.

[17] K. Righter, D. P. O'Brien, "Terrestrial planet formation," 2011. https://www.pnas.org/content/pnas/108/48/19165.full.pdf.

[18] A. Izidoro, S. N. Raymond, "Formation of Terrestrial Planets," 2018. https://arxiv.org/abs/1803.08830.

[19] P. Goldreich, W. R. Ward, "The Formation of Planetesimals," 1973. http://articles.adsabs.harvard.edu/pdf/1973ApJ...183.1051G.

[20] University of Arizona, "Lecture 13: The Nebular Theory of the origin of the Solar System," ? http://atropos.as.arizona.edu/aiz/teaching/nats102/mario/solar_system.html.

[21] J. R. Najita, S. J. Kenyon, B. C. Bromley, "From Pebbles and Planetesimals to Planets and Dust: the Protoplanetary Disk–Debris Disk Connection," 2021. https://arxiv.org/pdf/2111.06406.pdf.

[22] G. D'Angelo and J. J. Lissauer, "Formation of Giant Planets," 2018. https://arxiv.org/pdf/1806.05649.pdf.

[23] P. Bodenheimer, O. Hubickyja, J. J. Lissauer, "Models of the in situ formation of detected extrasolar giant planets," 1998. https://arxiv.org/pdf/2111.08776.pdf.

[24] A. B. Boss, "Evolution of the Solar Nebula. V. Disk Instabilities with Varied Thermodynamics," 2002. https://iopscience.iop.org/article/10.1086/341736/pdf.

[25] M. S. Marley, J. J. Fortney, O. Hubickyj, P. Bodenheimer, J. J. Lissauer, "On the Luminosity of Young Jupiters," 2007. https://iopscience.iop.org/article/10.1086/509759/pdf.

[26] J. Szulágyi, C. Mordasini, "Thermodynamics of Giant Planet Formation: Shocking Hot Surfaces on Circumplanetary Disks," 2016. https://arxiv.org/pdf/1609.08652.pdf.

[27] A. P. Boss, R. H. Durisen, "Chondrule-Forming Shock Fronts in the Solar Nebula: A Possible Unified Scenario for Planet and Chondrite Formation," 2005. https://iopscience.iop.org/article/10.1086/429160/pdf.

[28] J. J. Lissauer, O. Hubickyj, G. D'Angelo, P. Bodenheimer, "Models of Jupiter's Growth Incorporating Thermal and Hydrodynamic Constraints," 2008. https://arxiv.org/pdf/0810.5186.pdf.

[29] E. W. Thommes, M. J. Duncan, H. F. Levison, "Oligarchic growth of giant planets," 2003. https://arxiv.org/pdf/astro-ph/0303269.pdf.

[30] G. D'Angelo, S. J. Weidenschilling, J. J. Lissauer, P. Bodenheimer, "Growth of Jupiter: Formation in Disks of Gas and Solids and Evolution to the Present Epoch," 2020. https://arxiv.org/pdf/2009.05575.pdf.

[31] J. Szulágyi, L. Mayer, T. Quinn, "Circumplanetary disks around young giant planets: a comparison between core-accretion and disk instability," 2016. https://arxiv.org/pdf/1610.01791.pdf.

[32] P. Goldreich, W. R. Ward, "The Formation of Planetesimals," 1973. 1973ApJ...183.1051G.

[33] T. Tanigawa, "Accretion of Solid Materials onto Circumplanetary Disks from Protoplanetary Disks," 2014. https://arxiv.org/abs/1401.4218.

[34] J. Fung, E. Chiang, "Gap Opening in 3D: Single Planet Gaps," 2016.

[35] J. Fung, P. Artymowicz, Y. Wu, "The 3D flow field around an embedded planet," 2018. https://arxiv.org/abs/1505.03152.

[36] J. Szulágyi, A. Morbidelli, A. Crida, F. Masset, "Accretion of Jupiter-Mass Planets in the Limit of Vanishing Viscosity," 2014. https://iopscience.iop.org/article/10.1088/0004-637X/782/2/65/pdf.

[37] J. Szulágyi, F. Binkert, C. Surville, "Meridional Circulation of Dust and Gas in the Circumstellar Disk: Delivery of Solids onto the Circumplanetary Region," 2021. https://arxiv.org/abs/2103.12128.

[38] T. Moldenhauer, R. Kuiper, W. Kley, C. Ormel, "Steady state by recycling prevents premature collapse of protoplanetary atmospheres," 2021. https://arxiv.org/abs/2102.00939.

[39] R. O. Chametla, F. S. Masset, "Numerical study of coorbital thermal torques on cold or hot satellites," 2020. https://arxiv.org/abs/2011.12484.

[40] O. M. Guilera, N. Cuello, M. Montesinos, M. M. Miller Bertolami, M. P. Ronco, J. Cuadra, F. S. Masset, "Thermal torque effects on the migration of growing low-mass planets," 2019. https://arxiv.org/pdf/1904.11047.pdf.

[41] D. A. Velasco Romero, F. S. Masset, "Numerical study of dynamical friction with thermal effects – I. Comparison to linear theory," 2018. https://arxiv.org/pdf/1812.03191.pdf.

[42] D. A. Velasco Romero, F. S. Masset, "Dynamical friction with radiative feedback – II. High resolution study of the subsonic regime," 2020. https://arxiv.org/pdf/2004.13422.pdf.

[43] J. Papaloizou, R. Nelson, W. Kley, F. Masset, P. Artymowicz, "Disk-Planet Interactions During Planet Formation," 2006. https://arxiv.org/abs/astro-ph/0603196.

[44] D. Velasco-Romero, M. Han Veiga, R. Teyssier, F.. Masset, "Planet-disc interactions with Discontinuous Galerkin Methods using GPUs," 2018. https://arxiv.org/abs/1805.01443.

[45] G. D'Angelo, T. Henning, W. Kley, "Nested-grid calculations of disk-planet interaction," 2004. http://www.astro.ex.ac.uk/people/gennaro/publications/AA.2002.385.647.pdf.

[46] J. Szulágyi, "Effects of the Planetary Temperature on the Circumplanetary Disk and on the Gap," 2017. https://arxiv.org/abs/1705.08444.

[47] R. Chametla, F. Sanchez-Salcedo, F. Masset, A. Hidalgo-Gamez, "Gap formation by inclined massive planets in locally isothermal three-dimensional discs," 2017. https://arxiv.org/abs/1704.01626v1.

[48] K. Kanagawa, T. Muto, H. Tanaka, "Dust rings as a footprint of planet formation in a protoplanetary disk," 2021. https://arxiv.org/abs/2109.09579.

[49] S. Zhang, X. Hu, Z. Zhu, J. Bae, "Self-consistent ring model in protoplanetary disks: temperature dips and substructure formation," 2021. https://arxiv.org/abs/2110.00858.

[50] A. Crida, A. Morbidelli, F. Masset, "On the width and shape of gaps in protoplanetary disks," 2005. https://arxiv.org/pdf/astro-ph/0511082.pdf.

[51] S. J. Morrison, K. M. Kratter, "Gap Formation in Planetesimal Disks Via Divergently Migrating Planets," 2021. https://arxiv.org/pdf/1809.10209.pdf.

[52] C. Migaszewski, "On the migration of three planets in a protoplanetary disc and the formation of chains of mean motion resonances," 2016. https://arxiv.org/pdf/1511.01417.pdf.

[53] R. Les, M.-K. Lin, "Gap formation and stability in non-isothermal protoplanetary discs," 2018. https://arxiv.org/pdf/1501.01979.pdf.

[54] F. Binkert, "Planetary Gaps in Dust and Gas," 2019. https://people.phys.ethz.ch/~judits/Ms_Binkert_v1.pdf.

[55] J. Szulágyi, "Effects of the Planetary Temperature on the Circumplanetary Disk and on the Gap," 2017. https://arxiv.org/pdf/1705.08444.pdf.

[56] D. Mihalas, B. Mihalas, W. Arnett, "Foundations of Radiation Hydrodynamics," 1984. https://sgp.fas.org/othergov/doe/lanl/docs1/00418561.pdf.

[57] B. Bitsch, A. Crida, A. Morbidelli, W. Kley, I. Dobbs-Dixon, "Stellar irradiated discs and implications on migration of embedded planets - I. Equilibrium discs," 2012. https://www.aanda.org/articles/aa/pdf/2013/01/aa20159-12.pdf.

[58] J. Fung, E. Chiang, "Gap Opening in 3D: Single-Planet Gaps," 2016. https://iopscience.iop.org/article/10.3847/0004-637X/832/2/105.

[59] E. Toro, "Riemann Solvers and Numerical Methods for Fluid Dynamics," 2009. https://link.springer.com/book/10.1007/b79761.

[60] R. LeVeque, "Numerical Methods for Conservation Laws," 1992. https://link.springer.com/book/10.1007/978-3-0348-8629-1.

[61] , "Significance of Courant Number for Stability and Convergence of CFD Simulations," 2019. https://www.udvavisk.com/significance-courant-number/.

[62] R. Gingold, J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars," 1977. https://academic.oup.com/mnras/article/181/3/375/988212.

[63] L. Lucy, "A numerical approach to the testing of the fission hypothesis.," 1977. https://ui.adsabs.harvard.edu/abs/1977AJ.....82.1013L/abstract.

[64] M. Bartlett, "Statistical Estimations of Density Functions," 1962. https://www.jstor.org/stable/25049271.

[65] L. Boneva, D. Kendall, I. Stefanov, "Spline Transformations: Three New Diagnostic Aids for the Statistical Data-Analyst," 1971. https://rss.onlinelibrary.wiley.com/doi/10.1111/j.2517-6161.1971.tb00855.x.

[66] S. Genel, M. Vogelsberger, D. Nelson, D. Sijacki, V. Springel, L. Hernquist, "Following the flow: tracer particles in astrophysical fluid simulations," 2013. https://academic.oup.com/mnras/article/435/2/1426/1039776.

[67] M. Berger, "Local Adaptive Mesh Refinement for Shock Hydrodynamics," 1987. https://crd.lbl.gov/assets/pubs_presos/AMCS/ANAG/A113.pdf.

[68] G. D'Angelo, T. Henning, W. Kley, "Migration and Accretion of Protoplanets in 2D and 3D Global Hydrodynamical Simulations," 2002. https://arxiv.org/pdf/astro-ph/0208580.pdf.

[69] H. W. Yorke, P. Bodenheimer, "The Formation of Protostellar Disks. III. The Influence of Gravitationally Induced Angular Momentum Transport on Disk Structure and Appearance," 1999. https://iopscience.iop.org/article/10.1086/307867/pdf.

[70] M. Ruffert, "Collisions between a white dwarf and a main-sequence star. II. Simulations using multiple-nested refined grids," 1992. https://articles.adsabs.harvard.edu/pdf/1992A%26A...265...82R.

[71] A. Burkert, P. Bodenheimer, "Multiple fragmentation on collapsing protostars," 1993. https://articles.adsabs.harvard.edu/pdf/1993MNRAS.264..798B.

[72] R. Sampath, P. Barai, P. Nukala, "A parallel multigrid preconditioner for the simulation of large fracture networks," 2010. https://iopscience.iop.org/article/10.1088/1742-5468/2010/03/P03029/pdf.

[73] R. Teyssier, O. Agertz, "Computational Astrophysics 4 - The Godunov method," 2009. https://www.ics.uzh.ch/~teyssier/comp_astro_lectures/compastro_lecture4.pdf.

[74] B. Commerçon, R. Teyssier, E. Audit, P. Hennebelle, G. Chabrier, "Radiation hydrodynamics with Adaptive Mesh Refinement and application to prestellar core collapse," 2011. https://arxiv.org/abs/1102.1216.

[75] S. Laux, "On Particle-Mesh Coupling in Monte Carlo Semiconductor Device Simulation," 1996. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=541446.

[76] "The Cloud-in-a-Cell algorithm," 2007. https://www.gnu.org/software/archimedes/manual/html/node29.html.

[77] F. Jones, "Chapter 14 Gauss' theorem," http://www.owlnet.rice.edu/~fjones/chap14.pdf.

[78] C. Luessi, "," 2021, 2022. https://github.com/Brillouin-Zone/Master_thesis.

[79] C. Runge, "Über die numerische Auflösung von Differentialgleichungen," 1895. https://link.springer.com/article/10.1007%2FBF01446807.

[80] M. Fadlisyah, "A Rewriting-Logic-Based Approach for the Formal Modeling and Analysis of Interacting Hybrid Systems," 2014. https://www.researchgate.net/figure/The-Runge-Kutta-2nd-order-and-4th-order-methods_fig29_265444798.

[81] B. van Leer, "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method," 1979. https://doi.org/10.1016/0021-9991(79)90145-1.

[82] C.P. Dullemond, H.H. Wang, "Chapter 10: Coordinate systems and gridding techniques," 2009. https://www.ita.uni-heidelberg.de/~dullemond/lectures/num_fluid_2009/.

[83] "remainder, remainderf, remainderl," Accessed 27. Mar. 2022. https://en.cppreference.com/w/c/numeric/math/remainder.

[84] J. B. Simon, P. J. Armitage, R. Li, A. N. Youdin, "The Mass and Size Distribution of Planetesimals Formed by the Streaming Instability. I. The Role of Self-Gravity," 2016. 2016 ApJ 822 55.

[85] "Using MPI with C," Accessed 29.03.2022. https://curc.readthedocs.io/en/latest/programming/MPI-C.html.

[86] SPI, "Open MPI: Open Source High Performance Computing," Accessed 29.03.2022. https://www.open-mpi.org/.

[87] W. Kendall, "MPI Broadcast and Collective Communication," Accessed 29.03.2022. https://mpitutorial.com/tutorials/.

[88] M. Hjorth-Jensen, "9. Parallelization with MPI and OpenMPI," Accessed 29.03.2022. http://compphysics.github.io/ComputationalPhysics2/doc/LectureNotes/_build/html/parallelization.html.

[89] H. Mo, F van den Bosch, S. White, "Galaxy Formation and Evolution," 2010. Cambridge University Press, ISBN: 978-0-521-85793-2.

[90] S. Chandrasekhar, N. R. Lebovitz, "The Potentials and the Superpotentials of Homogeneous Ellipsoids," 1962. 1962ApJ...136.1037C.