

Numerical simulations with PLUTO code

Miljenko Čemeljić

Nicolaus Copernicus Astronomical Center of the
Polish Academy of Sciences,
Warsaw, Poland



&

Academia Sinica Institute of Astronomy and
Astrophysics, Taipei, Taiwan, Visiting Scholar



General outline

- **PART I:** General introduction to PLUTO code and its physics modules, installation of the code, testing of the installation with Sod shock tube test in 1D, visualization with gnuplot.
- **PART II:** Hands-on introduction to setup files of PLUTO. Setup of 2D simulation from template: Orszag-Tang test. Visualization of HD results with Paraview & VisIt. Animation of the results.
- **PART III:** Introduction to accretion disk. Setup of HD disk in axisymmetric 2D. Initial and boundary conditions, numerical methods and tricks.
- **PART IV:** Star-disk magnetospheric interaction simulations, with some examples. Setup of axisymmetric 2D setup with dipole and multipole magnetic field. Visualization of magnetic field lines with Paraview and Python.
- **PART V:** Full 3D setup: Orszag-Tang test in 3D. HD and then magnetic accretion disk. Restart and initialization from (modified) files. Visualization of 3D results. Various uses of the results: light-curves for comparison with observations, post-processing of the results (PLUTO+DUSTER): radiative effects in motion of dust grains. Concluding remarks.

Required packages: PLUTO source code, C-compiler (+MPI), Python (matplotlib, NumPy), Gnuplot, Paraview, VisIt.

Outline, Part I

- “Simulations” ver. “computations”.
- PLUTO physics modules.
- Documentation, test problems, templates.
- Practical part: installing the code.
- Testing with the use of 1D Sod shock tube test provided with the code.
- Visualization with gnuplot.

Simulation ver. computation

- Computation is actually an evaluation: you plug the numbers in a known algorithm and obtain the result. For this, you need to know the analytical expression. You still can use calculator or a computer to actually obtain numbers and plot e.g. trajectory of a bullet, but you **do** know the equation for the solution.
- Simulation is when you do not know the equation for the solution. You set the governing equations, e.g. differential equations, and use a numerical method to find the solutions. This is usually done in time steps, and we obtain the solution to some precision. We do not know the analytical expression for the solution, we just have numbers=numerical simulation.
- Why PLUTO? I used other codes, but PLUTO is constantly evolving (and is used in variety of problems) **and** the development is followed in the manual-which is not so often the case in the coding world. It gives chance to students to go through the-still steep-learning curve in the shortest possible time. Check also Doxygen html files in PLUTO/Doc .

PLUTO physics modules, equations

6.1 The HD Module

The HD module may be used to solve the Euler or the Navier-Stokes equations of classical fluid dynamics. The relevant source files and definitions for this module can be found in the `Src/HD` directory.

With the HD module, **PLUTO** evolves in time following system of conservation laws:

$$\frac{\partial}{\partial t} \begin{pmatrix} \rho \\ \mathbf{m} \\ E_t + \rho\Phi \end{pmatrix} + \nabla \cdot \begin{pmatrix} \rho\mathbf{v} \\ \mathbf{m}\mathbf{v} + p\mathbf{l} \\ (E_t + p + \rho\Phi)\mathbf{v} \end{pmatrix}^T = \begin{pmatrix} 0 \\ -\rho\nabla\Phi + \rho\mathbf{g} \\ \mathbf{m} \cdot \mathbf{g} \end{pmatrix} \quad (6.1)$$

where ρ is the mass density, $\mathbf{m} = \rho\mathbf{v}$ is the momentum density, \mathbf{v} is the velocity, p is the thermal pressure and E_t is the total energy density:

$$E_t = \rho e + \frac{\mathbf{m}^2}{2\rho}. \quad (6.2)$$

An equation of state provides the closure $\rho e = \rho e(p, \rho)$.

The source term on the right includes contributions from body forces and is written in terms of the (time-independent) gravitational potential Φ and the acceleration vector \mathbf{g} (§5.4).

6.2 The MHD Module

The MHD module is suitable for the solution of ideal or resistive (non-relativistic) magnetohydrodynamical equations. Source and definition files are located inside the Src/MHD directory.

With the MHD module, **PLUTO** solves the following system of conservation laws:

$$\begin{aligned}
 \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) &= 0 \\
 \frac{\partial \mathbf{m}}{\partial t} + \nabla \cdot \left[\mathbf{m} \mathbf{v} - \mathbf{B} \mathbf{B} + \mathbf{I} \left(p + \frac{\mathbf{B}^2}{2} \right) \right]^T &= -\rho \nabla \Phi + \rho \mathbf{g} \\
 \frac{\partial \mathbf{B}}{\partial t} + \nabla \times (c \mathbf{E}) &= 0 \\
 \frac{\partial (E_t + \rho \Phi)}{\partial t} + \nabla \cdot \left[\left(\frac{\rho \mathbf{v}^2}{2} + \rho e + p + \rho \Phi \right) \mathbf{v} + c \mathbf{E} \times \mathbf{B} \right] &= \mathbf{m} \cdot \mathbf{g}
 \end{aligned} \tag{6.4}$$

where ρ is the mass density, $\mathbf{m} = \rho \mathbf{v}$ is the momentum density, \mathbf{v} is the velocity, p is the gas (thermal) pressure, \mathbf{B} is the magnetic field² and E_t is the total energy density:

$$E_t = \rho e + \frac{\mathbf{m}^2}{2\rho} + \frac{\mathbf{B}^2}{2}. \tag{6.5}$$

where an additional equation of state provides the closure $\rho e = \rho e(p, \rho)$ (see Chapter 7). The source term on the right includes contributions from body forces and is written in terms of the (time-independent) gravitational potential Φ and the acceleration vector \mathbf{g} (see §5.4).

In the third of Eq. (6.4), \mathbf{E} is the electric field defined by the expression

$$c \mathbf{E} = -\mathbf{v} \times \mathbf{B} + \frac{\eta}{c} \cdot \mathbf{J} + \frac{\mathbf{J}}{ne} \times \mathbf{B} \quad \left(\mathbf{J} = c \nabla \times \mathbf{B} \right) \tag{6.6}$$

where the first term is the convective term, the second term is the resistive term (η denotes the resistivity tensor, see §8.2) while the third term is the Hall term (§8.1). Note that the speed of light c never enters

PLUTO physics modules, equations

6.3 The RHD Module

The RHD module implements the equations of special relativistic fluid dynamics in 1, 2 or 3 dimensions. Velocities are always assumed to be expressed in units of the speed of light. The special relativistic module comes with 2 different equations of state, and it also works in curvilinear coordinates. Gravity in Newtonian approximation can also be incorporated. The relevant source files and definitions for this module can be found in the `Src/RHD` directory.

The special relativistic module evolves the conservative set U of state variables

$$U = \left(D, m_1, m_2, m_3, E_t \right)^T$$

where D is the laboratory density, $m_{x1,x2,x3}$ are the momentum components, E_t is the total energy (including contribution from the rest mass). The evolutionary conservative equations are

$$\frac{\partial}{\partial t} \begin{pmatrix} D \\ \mathbf{m} \\ E_t \end{pmatrix} + \nabla \cdot \begin{pmatrix} D\mathbf{v} \\ \mathbf{m}\mathbf{v} + p\mathbf{l} \\ \mathbf{m} \end{pmatrix}^T = \begin{pmatrix} 0 \\ \mathbf{f}_g \\ \mathbf{v} \cdot \mathbf{f}_g \end{pmatrix}$$

where \mathbf{v} is the velocity, p is the thermal pressure. Primitive variables \mathbf{V} always include the rest-mass density ρ , three-velocity $\mathbf{v} = (v_{x1}, v_{x2}, v_{x3})$ and pressure p . With PLUTO 4.4, the acceleration term \mathbf{f}_g is treated consistently³ with the formalism of [Tau48]. If \mathbf{a} is the acceleration vector,

$$\mathbf{f}_g = \rho\gamma^2 [\gamma^2 \mathbf{v} (\mathbf{v} \cdot \mathbf{a}) + \mathbf{a}] . \quad (6.12)$$

PLUTO physics modules, equations

6.4 The RMHD Module

The RMHD module implements the equations of (ideal) special relativistic magnetohydrodynamics in 1, 2 or 3 dimensions. Velocities are always assumed to be expressed in units of the speed of light. Source and definition files are located inside the Src/RMHD directory.

The RMHD module solves the following system of conservation laws:

$$\frac{\partial}{\partial t} \begin{pmatrix} D \\ \mathbf{m} \\ E_t \\ \mathbf{B} \end{pmatrix} + \nabla \cdot \begin{pmatrix} D\mathbf{v} \\ w_t\gamma^2\mathbf{v}\mathbf{v} - \mathbf{b}\mathbf{b} + p_t \\ \mathbf{m} \\ \mathbf{v}\mathbf{B} - \mathbf{B}\mathbf{v} \end{pmatrix}^T = \begin{pmatrix} 0 \\ \mathbf{f}_g \\ \mathbf{v} \cdot \mathbf{f}_g \\ 0 \end{pmatrix} \quad (6.13)$$

where D is the laboratory density, \mathbf{m} is the momentum density, E is the total energy (including contribution from the rest mass) while \mathbf{f}_g is an acceleration term (see [6.3](#)).

Primitive variables are similar to the RHD module but they also contain the magnetic field, $\mathbf{V} = (\rho, \mathbf{v}, p, \mathbf{B})$. The relation between \mathbf{V} and \mathbf{U} is

$$\begin{aligned} D &= \gamma\rho \\ \mathbf{m} &= w_t\gamma^2\mathbf{v} - b^0\mathbf{b} \\ E_t &= w_t\gamma^2 - b^0b^0 - p_t \end{aligned} \quad , \quad \begin{cases} b^0 = \gamma\mathbf{v} \cdot \mathbf{B} \\ \mathbf{b} = \mathbf{B}/\gamma + \gamma(\mathbf{v} \cdot \mathbf{B})\mathbf{v} \\ w_t = \rho h + \mathbf{B}^2/\gamma^2 + (\mathbf{v} \cdot \mathbf{B})^2 \\ p_t = p + \frac{\mathbf{B}^2/\gamma^2 + (\mathbf{v} \cdot \mathbf{B})^2}{2} \end{cases}$$

PLUTO physics modules, equations

A note on public vs. non-public modules.

Besides the official code release, a few modules have not yet been made available with the standard public version, as they are still under active development or testing stage. In other circumstances, a private module may have been implemented under specific collaboration policies, which do not grant its public distribution. These non-offical modules include:

- Lagrangian Particle Module
(Developers: B. Vaidya [*bvaidya@iiti.ac.in*], D. Mukherjee [*dipanjan@iucaa.in*]);
- Dust Module
(Developers: A. Mignone [*mignone@to.infn.it*], M. Flock [*flock@mpia.de*]);
- Relativistic Resistive MHD
(Developers: A. Mignone [*mignone@to.infn.it*], G. Mattia [*mattia@mpia.de*]);

Distribution, private sharing and usage of these modules is permitted only in the form of a collaboration between our partner institutions network, requiring co-authorship from at least one of the module developers.

6.5 The Resistive RMHD (*ResRMHD*) Module

Note: This module is not part of the public code release, see “Terms & Conditions of Use” at the beginning of this guide

The ResRMHD module deals with the non-ideal relativistic MHD equations using the approaches discussed in [MMBD19](#). Source and definition files are located inside the Src/ResRMHD directory.

The set of resistive relativistic equations arising from the time and space split of the covariant are, in vectorial form,

$$\begin{aligned}
 \frac{\partial D}{\partial t} + \nabla \cdot (D\mathbf{v}) &= 0, \\
 \frac{\partial \mathbf{m}}{\partial t} + \nabla \cdot (w\mathbf{u}\mathbf{u} + p\mathbf{l} + \mathbb{T}) &= 0, \\
 \frac{\partial \mathcal{E}}{\partial t} + \nabla \cdot \mathbf{m} &= 0, \\
 \frac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} &= 0, \\
 \frac{\partial \mathbf{E}}{\partial t} - \nabla \times \mathbf{B} &= -\mathbf{J},
 \end{aligned} \tag{6.14}$$

where \mathbf{l} is the identity matrix and the fluid conserved variables are the density $D = \rho\gamma$ as measured in the laboratory frame, the total momentum density $\mathbf{m} = w\gamma\mathbf{u} + \mathbf{E} \times \mathbf{B}$, and the total energy density

$$\mathcal{E} = w\gamma^2 - p + \mathcal{P}_{\text{EM}}. \tag{6.15}$$

In the expressions above, $w = \varepsilon + p$ is the specific enthalpy and $\mathcal{P}_{\text{EM}} = (E^2 + B^2)/2$ denotes the EM energy density. Finally,

$$\mathbb{T} = -\mathbf{E}\mathbf{E} - \mathbf{B}\mathbf{B} + \frac{1}{2}(E^2 + B^2)\mathbf{l} \tag{6.16}$$

7. Equation of State

In the current implementation, **PLUTO** describes a thermally ideal gas obeying the *thermal* Equation of State (EOS)

$$p = nk_B T = \frac{\rho}{m_u \mu} k_B T \quad (7.1)$$

where p is the pressure, n is the total particle number density, k_B is the Boltzmann constant, T is the temperature, ρ is the density, m_u is the atomic mass unit and μ is the mean molecular weight. The thermal EOS describes the thermodynamic state of a plasma in terms of its pressure p , density ρ , temperature T and chemical composition μ . Eq. (7.1) is written in CGS physical units. Using code units for p and ρ while leaving temperature in Kelvin, the thermal EOS is conveniently re-expressed as

$$p = \frac{\rho T}{\mathcal{K} \mu} \quad \iff \quad T = \frac{p}{\rho} \mathcal{K} \mu \quad \left(\text{where } \mathcal{K} = \frac{m_u v_0^2}{k_B} \right) \quad (7.2)$$

where \mathcal{K} is the **KELVIN** macro which depends explicitly on the value of `UNIT_VELOCITY`.

8. Nonideal Effects

In this chapter we give an overview of the code capabilities for treating dissipative (or diffusion) terms which, at present, include

- Hall MHD (MHD), described in §8.1
- Resistivity (MHD), described in §8.2
- Thermal conduction (HD, MHD), described in §8.3
- Viscosity (HD, MHD), described in §8.4

Each modules can be individually turned on from the physics sub-menus accessible via the Python script.

PLUTO documentation, templates and test examples

- PLUTO is freely available. Source files of the code are downloadable from <http://plutocode.ph.unito.it/>
- After unpacking, the source code is in PLUTO directory. In PLUTO/Doc is the manual, userguide.pdf. Follow the “Quick start” at the beginning of the document to **install and test the code. Produce the gnuplot output specified in the manual, to verify if the setup works.**
- The code comes with templates of subroutines which are usually changed. They are given in the PLUTO/Src/Templates
- Test examples, under /PLUTO/Test_Problems, are basic versions of setups used in previous versions of the code, during the developments of the setups for simulations presented in publications by various groups. It is a good library of examples for faster start of one's own project.
- I will provide the setup for HD and MHD accretion disk for this lectures.

Files for setup and their modifications

- The PLUTO original source will not be changed, for our simulations we just append to it our version of some files in a pre-compiling step.
- The changed version of a file in the work directory (for which I suggest the name /home/Pluto) has, by default in PLUTO, priority to the original version in PLUTO/Src directory.
- The files to be copied into the work directory from the working version into a new setup are **init.c**, **pluto.ini**, **definitions.h**, **res_eta.c**, **visc_nu.c**.
- The file **definitions.h** is the only *.h file changed in the work directory. All the other *.h files are to be changed directly in PLUTO/Src directory.
- In the file **pluto.ini** are defined the grid, solvers and run parameters.
- Most of the entries in **definitions.h** are done through the python environment during the setup of the run, but some entries are to be done by hand editing the **definitions.h** file, prior to compilation.
- In the file **init.c** is defined the physics setup.

Shell variable setup, aliases

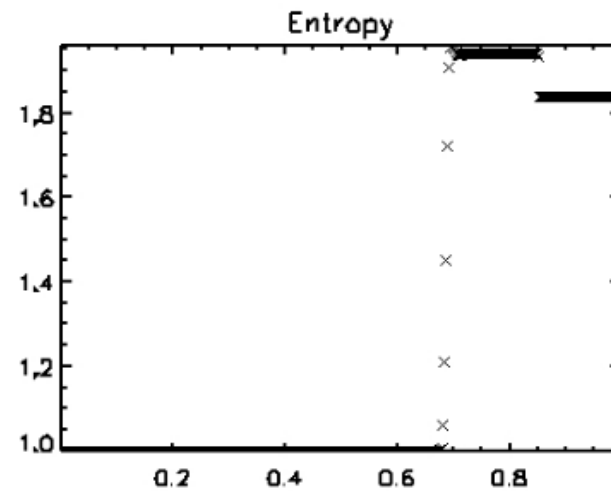
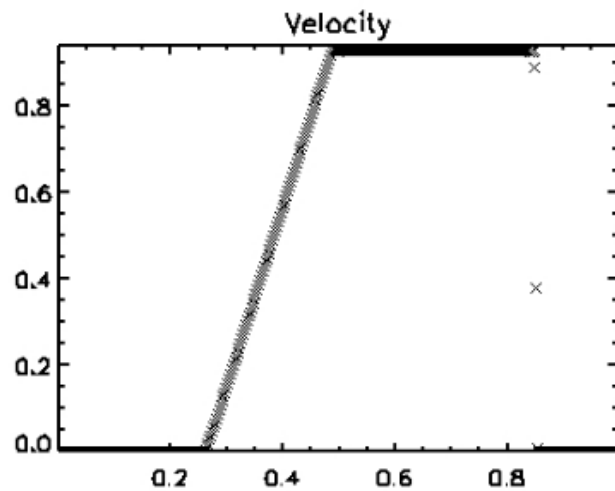
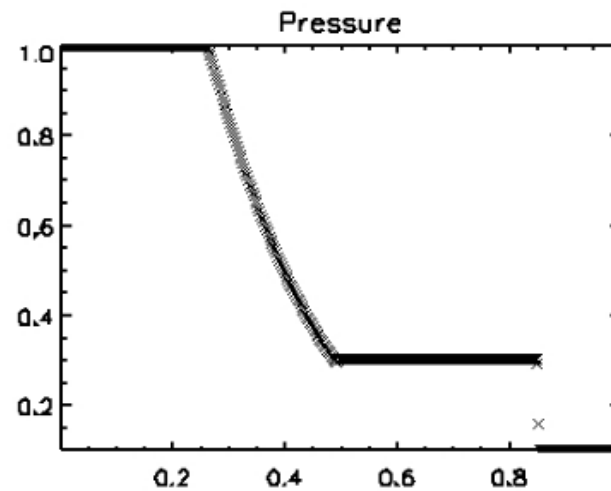
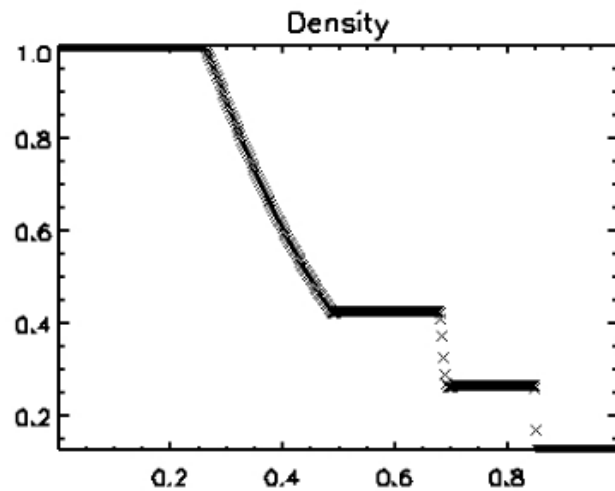
- We follow the “Quick start” from the beginning of the PLUTO/Doc/userguide.pdf
- Instead of every time after login repeating
export PLUTO_DIR=/home/user/PLUTO # in bash shell
Create in .bash_aliases (or .bashrc) a shell variable with path to PLUTO:
export PLUTO_DIR="/home/miki/PLUTO"
- Also, it is useful to create an alias (shortcut) for running the setup.py, e.g.:
alias pls='python \$PLUTO_DIR/setup.py'
 - For runs on multiple processors on laptop, useful is alias like:
alias plutorun6='mpirun -np 6 ./pluto'

Test problem: Sod shock tube in 1D

Detailed Description from **Doxygen file**: The Sod shock tube problem is one of the most used benchmark for shock-capturing schemes. It is a one-dimensional problem with initial condition given by a discontinuity separating two constant states:

$$\begin{aligned}(\rho, v_x, p)_L &= (1, 0, 1) && \text{for } x < 0.5 \\ (\rho, v_x, p)_R &= \left(\frac{1}{8}, 0, \frac{1}{10}\right) && \text{for } x > 0.5\end{aligned}$$

The evolved structured at $t=0.2$ is shown in the panels below and consists of a left-going rarefaction wave, a right-going contact discontinuity and a right-going shock wave. The results shown here were carried with PARABOLIC interpolation, CHARACTERISIC_TRACING time stepping and the two_shock Riemann solver on 400 zones (**configuration #04**).



Test problem: Sod shock tube in 1D-use #04 if problems with #01

0.2 Running a simple shock-tube problem

PLUTO can be quickly configured to run one of the several test problems provided with the distribution. Assuming that your system satisfies all the requirements described in the next chapter (i.e. C compiler, Python, etc..) you can quickly setup **PLUTO** in the following way:

1. Change directory to any of the test problems under **PLUTO/Test_Problems**, e.g.

```
~> cd $PLUTO_DIR/Test_Problems/HD/Sod
```

2. Copy the header and initialization files from a configuration of our choice (e.g. #01):

```
~/PLUTO/Test_Problems/HD/Sod> cp definitions_01.h definitions.h
~/PLUTO/Test_Problems/HD/Sod> cp pluto_01.ini pluto.ini
```

3. Run the Python script using

```
~/PLUTO/Test_Problems/HD/Sod> python $PLUTO_DIR/setup.py
```

and select “Setup problem” from the main menu, see Fig. [1.2](#) You can choose (by pressing Enter) or modify the default setting using the arrow keys.

4. Once you return to the main menu, select “Change makefile”, choose a suitable makefile (e.g. Linux.gcc.defs) and press enter.

All the information relevant to the specific problem should now be stored in the four files `init.c` (assigns initial condition and user-supplied boundary conditions), `pluto.ini` (sets the number of grid zones, Riemann solver, output frequency, etc.), `definitions.h` (specifies the geometry, number of dimensions, reconstruction, time stepping scheme, and so forth) and the makefile.

5. Exit from the main menu (“Quit” or press ‘q’) and type

```
~/PLUTO/Test_Problems/HD/Sod> make
```

to compile the code.

6. You can now run the code by typing

```
~/PLUTO/Test_Problems/HD/Sod> ./pluto
```

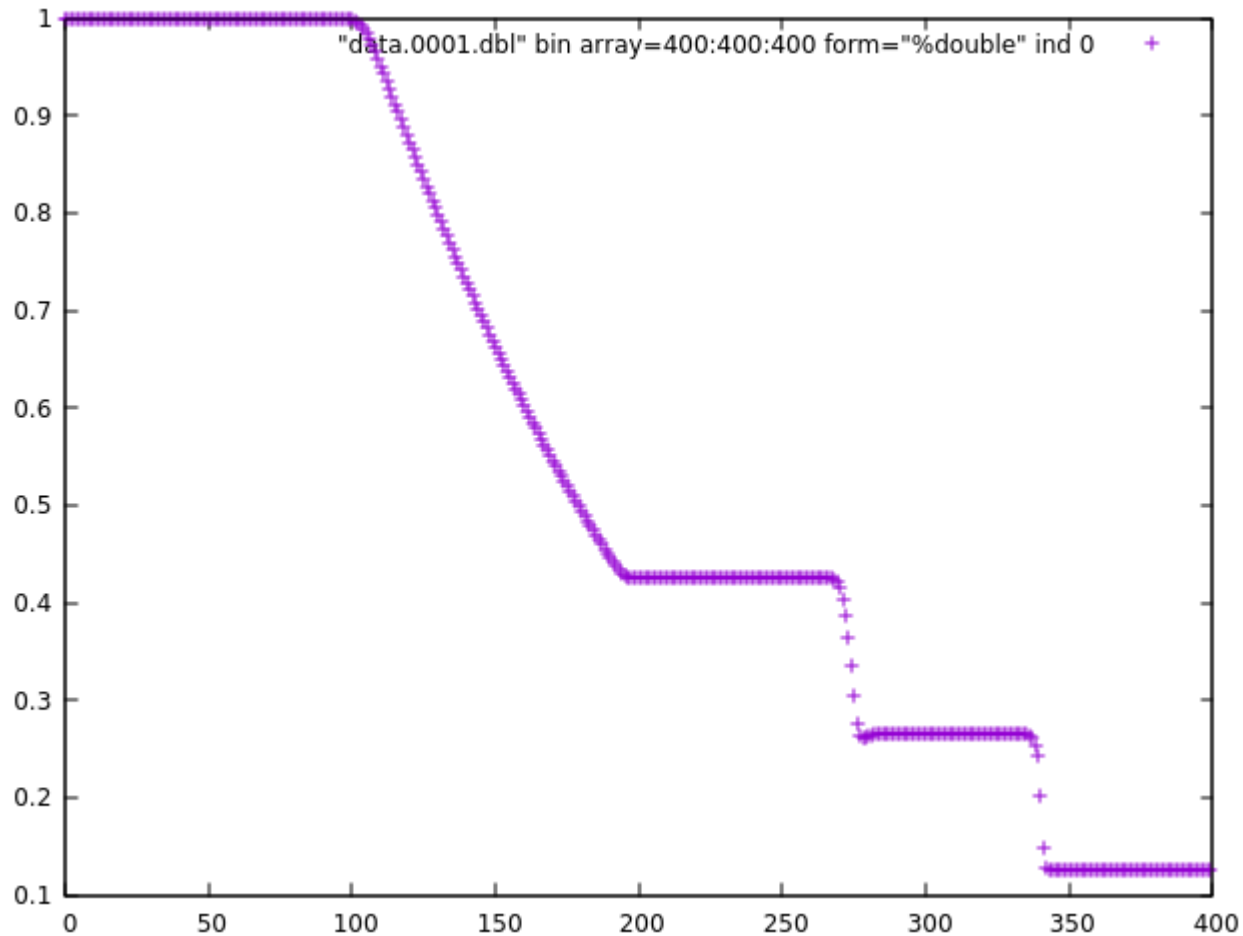
At this point, **PLUTO** reads the initialization file `pluto.ini` and starts integrating. The run should take a few seconds (or less) and the integration log should be dumped to screen.

Test problem: Sod shock tube in 1D

Typing in terminal:

```
gnuplot> plot "data.0001.dbl" bin array=400:400:400 form="%double" ind 0
```

You should obtain:



Summary of the Part I

- We got acquainted with PLUTO and its physics modules.
- I show where to find documentation, test problems and templates.
- Practical part: we went through the code installation.
- We tested the setup using the 1D Sod shock tube test provided with the code.
- We used visualization of the results with gnuplot.

Outline, Part II

- Hands-on introduction to setup files of PLUTO.
- Output formats & files which we should save for later analysis.
- Initial and boundary conditions setup for 2D Orszag-Tang test.
- Visualization of the results with Paraview and VisIt.
- Movies time: animation and saving of the movie.

Hands-on introduction to setup files of PLUTO

- Guided tour through the source files:
 - Python PLUTO/setup.py and PLUTO/Config/Linux.mpicc.defs files.
 - pluto.ini: grid, solver, choice of boundary conditions, output steps, parameters.
 - Definition of output formats.
 - Files to save in a Results folder together with output files.
 - definitions.h: macros setting.
 - init.c: physical setup, equations for initial and boundary conditions.

Hands-on introduction: PLUTO/setup.py

```
miki@petri: ~/Applics/PLUTO44
I A setup.py (python) import os Row 1 Col 1 9:35 Ctrl-K H
import os
import sys
import shutil

try:
    os.environ['PLUT044_DIR']
except KeyError:
    print('PLUT044_DIR not defined. Setting it to the Current Directory')
    pluto_directory = os.getcwd()
    pass
else:
    pluto_directory = os.environ['PLUT044_DIR']

sys.path.append(pluto_directory + '/Tools/Python/')
import menu
import configure
from make_problem import MakeProblem

def PlutoInterface(pluto_dir, do_auto_update = False):
    work_dir = os.getcwd()
    interface_optval = ''

** Joe's Own Editor v4.1 ** (utf-8) ** Copyright © 2015 **
```

Hands-on introduction: PLUTO/Config/Linux.mpicc.defs

```
miki@petri: ~/Applics/PLUTO44
IW Config/Linux.mpicc.defs Row 1 Col 1 9:
#####
# Configuration file for mpicc (parallel)
#
#####

CC      = mpicc
CFLAGS  = -c -O3 -Wundef
LDFLAGS = -lm

PARALLEL = TRUE
USE_HDF5  = FALSE
USE_PNG   = FALSE

#####
# MPI additional specifications
#####

ifeq ($(strip $(PARALLEL)), TRUE)
endif

#####
# HDF5 library options
#####

ifeq ($(strip $(USE_HDF5)), TRUE)
INCLUDE_DIRS += /usr/local/lib/HDF5-parallel/include
LDFLAGS += /usr/local/lib/HDF5-parallel/lib -lz -lm -lhdf5
endif

#####
# PNG library options
#####

** Joe's Own Editor v4.1 ** (utf-8) ** Copyright © 2015 **
```

Hands-on introduction: pluto.ini

```
miki@petri: ~/Pluto44/OTang2D
I pluto.ini (ini)
[Grid]
X1-grid 1 0.0 200 u 6.28318530717959
X2-grid 1 0.0 200 u 6.28318530717959
X3-grid 1 0.0 1 u 1.0

[Chombo Refinement]
Levels 4
Ref_ratio 2 2 2 2
Regrid_interval 2 2 2 2
Refine_thresh 0.3
Tag_buffer_size 3
Block_factor 4
Max_grid_size 32
Fill_ratio 0.75

[Time]
CFL 0.8
CFL_max_var 1.1
tstop 3.1
first_dt 1.e-4

[Solver]
Solver hll

[Boundary]
```

```
miki@petri: ~/Pluto44/OTang2D
I pluto.ini (ini)
[Boundary]
X1-beg periodic
X1-end periodic
X2-beg periodic
X2-end periodic
X3-beg periodic
X3-end periodic

[Static Grid Output]
uservar 0
dbl 10.31 -1 single_file
flt -1.0 -1 single_file
vtk 0.31 -1 single_file
tab -1.0 -1
ppm -1.0 -1
png -1.0 -1
log 1
analysis -1.0 -1

[Chombo HDF5 output]
Checkpoint_interval -1.0 0
Plot_interval 1.0 0

[Parameters]
SCRH 0
```

Output formats & files to save

- In pluto.ini we defined which output
- data.xxxx.dbl files – viewing, processing with Python, idl etc. packages.
-also, for restart, together with restart.out.
- Data.xxxxdbl.h5 - hdf5 files-need to set the flag in /PLUTO/Config/*.defs file
- data.xxxx.vtk files – for viewing with Paraview, VisIt.
- Together with data.xxxx.dbl and .vtk files, always save also, in the same directory with the results, pluto.ini, definitions.h. init.c, grid.out, dbl.out, restart.out, to know which setup produced the files, and also for some analysis and plotting packages, which might need them.
- grid.out files defines the geometry, dbl.out lists the output variables

```
miki@petri: ~/Pluto44/O_Tang2D
I A definitions.h (c) Row
#define PHYSICS MHD
#define DIMENSIONS 2
#define GEOMETRY CARTESIAN
#define BODY_FORCE NO
#define COOLING NO
#define RECONSTRUCTION LINEAR
#define TIME_STEPPING HANCOCK
#define NTRACER 0
#define PARTICLES NO
#define USER_DEF_PARAMETERS 1

/* -- physics dependent declarations -- */

#define EOS IDEAL
#define ENTROPY_SWITCH NO
#define DIVB_CONTROL EIGHT_WAVES
#define BACKGROUND_FIELD NO
#define AMBIPOLAR_DIFFUSION NO
#define RESISTIVITY NO
#define HALL_MHD NO
#define THERMAL_CONDUCTION NO
#define VISCOSITY NO
#define ROTATING_FRAME NO

/* -- user-defined parameters (labels) -- */

#define SCRH 0

/* [Beg] user-defined constants (do not change this line) */

#define LIMITER MC_LIM
```

Hands-on introduction: definitions.h

```

miki@petri: ~/Pluto44/OTang2D
I A init.c (c) void Init (double *us, double x1,
/* *****/
void Init (double *us, double x1, double x2, double x3)
/*
*
*****/
{
double x=x1, y=x2 ,z=x3;
g_gamma = 5./3.;//g_inputParam[GAMMA];

us[VX1] = -0.7* sin(y);
us[VX2] = 0.7* sin(x);
us[VX3] = 0.0;
us[BX1] = - sin(y);
us[BX2] = sin(2.0*x);
us[BX3] = 0.0;
us[RHO] = 1.;
#if EOS != ISOTHERMAL && EOS != BAROTROPIC
us[PRS] = 10.0;
#endif
us[TRC] = 1.0;

us[AX1] = 0.0;
us[AX2] = 0.0;
us[AX3] = cos(y) + 0.5 * cos(2.0*x);

#if DIMENSIONS == 3 && ROTATE == -1
{
double c0 = 0.8;
us[VX2] = - sin(z);
us[VX3] = sin(y);
us[VX1] = 0.0;

us[BX2] = c0*( - 2.0*sin(2.0*z) + sin(x));
us[BX3] = c0*( sin(x) + sin(y));
us[BX1] = c0*( sin(y) + sin(z));

```

Hands-on introduction: init.c

- Note that at the beginning of each file in PLUTO is a brief description about what it does. It is a very good practice, follow it.

```

miki@petri: ~/Pluto44/OTang2D
I A init.c (c) Row 27 Col 1 12:55 Ctrl-K H for
/* *****/
/*!
\file
\brief Orszag-Tang MHD vortex.

The Orszag Tang vortex system describes a doubly periodic fluid
configuration leading to two-dimensional supersonic MHD turbulence.
Although an analytical solution is not known, its simple and reproducible
set of initial conditions has made it a widespread benchmark for
inter-scheme comparison.

The computational domain is the periodic box  $[0, 2\pi]^D$  where
 $c$  D is the number of spatial dimensions.
In 2D, the initial condition is given by
\{
\vec{v} = \left(-\sin y, \sin x, 0\right) \,, \,\, \text{\texttt{qqquad}}
\vec{B} = \left(-\sin y, \sin 2x, 0\right) \,, \,\, \text{\texttt{qqquad}}
\rho = 25/9 \,, \,\, \text{\texttt{qqquad}}
p = 5/3
\}

This test problem does not have any input parameter.

A snapshot of the solution on a  $c$  512x512 grid is shown below.

\image html mhd_ot.02.jpg "Density at t=3.1 (configuration #02)."

\author A. Mignone (mignone@ph.unito.it)
\date April 13, 2014

\b References
- "Comparison of some Flux Corrected Transport and TVD numerical
schemes for hydrodynamic and magnetohydrodynamic problems",
Toth & Odstrcil, JCP (1996) 128, 82
- "High-order conservative finite difference GLM-MHD schemes for

```

2D test problem: Orszag-Tang

0.3 Running the Orszag-Tang MHD vortex test

1. Change directory to PLUTO/Test_Problems/MHD/Orszag_Tang.
2. Choose a configuration (e.g. #02) and copy the corresponding configuration files, i.e.,

```
~/PLUTO/Test_Problems/MHD/Orszag_Tang> cp definitions_02.h definitions.h
~/PLUTO/Test_Problems/MHD/Orszag_Tang> cp pluto_02.ini pluto.ini
```

3. Run the Python script:

```
~/PLUTO/Test_Problems/MHD/Orszag_Tang> python $PLUTO_DIR/setup.py
```

select "Setup problem" and choose the default setting by pressing enter;

4. Once you return to the main menu, select "Change makefile" and choose a suitable makefile (e.g. Linux.gcc.defs) and press enter.
5. Exit from the main menu ("Quit" or press 'q'). Edit pluto.ini and, under the *[Grid]* block, lower the resolution from 512 to 200 in both directions (X1-grid and X2-grid). Change *single_file*, in the "dbl" output under the *[Static Grid Output]* block, to *multiple_files*. Finally, edit *definitions.h* and change *PRINT_TO_FILE* from *YES* to *NO*.
6. Compile the code:

```
~/PLUTO/Test_Problems/MHD/Orszag_Tang> make
```

7. If compilation was successful, you can now run the code by typing

```
~/PLUTO/Test_Problems/MHD/Orszag_Tang> ./pluto
```

At this point, **PLUTO** reads the initialization file *pluto.ini* and starts integrating. The run should take a few minutes (depending on the machine you're running on) and the integration log should be dumped to screen.

Orszag-Tang I.C. & B.C.

I introduce some numerical simulations lingo here in ***bold***:

We set velocity and magnetic field in a 2D Cartesian coordinates, in a ***computational box*** with $(x,y,0)=(256 \times 256 \times 0)$ grid cells and a ***physical domain*** $x,y=[0,2\pi]$

Velocity components: $v = (-\sin y, \sin x, 0)$

Magnetic field: $B = (-\sin y, \sin 2x, 0)$

Density: $\rho = 25/9$

Pressure: $p = 5/3$

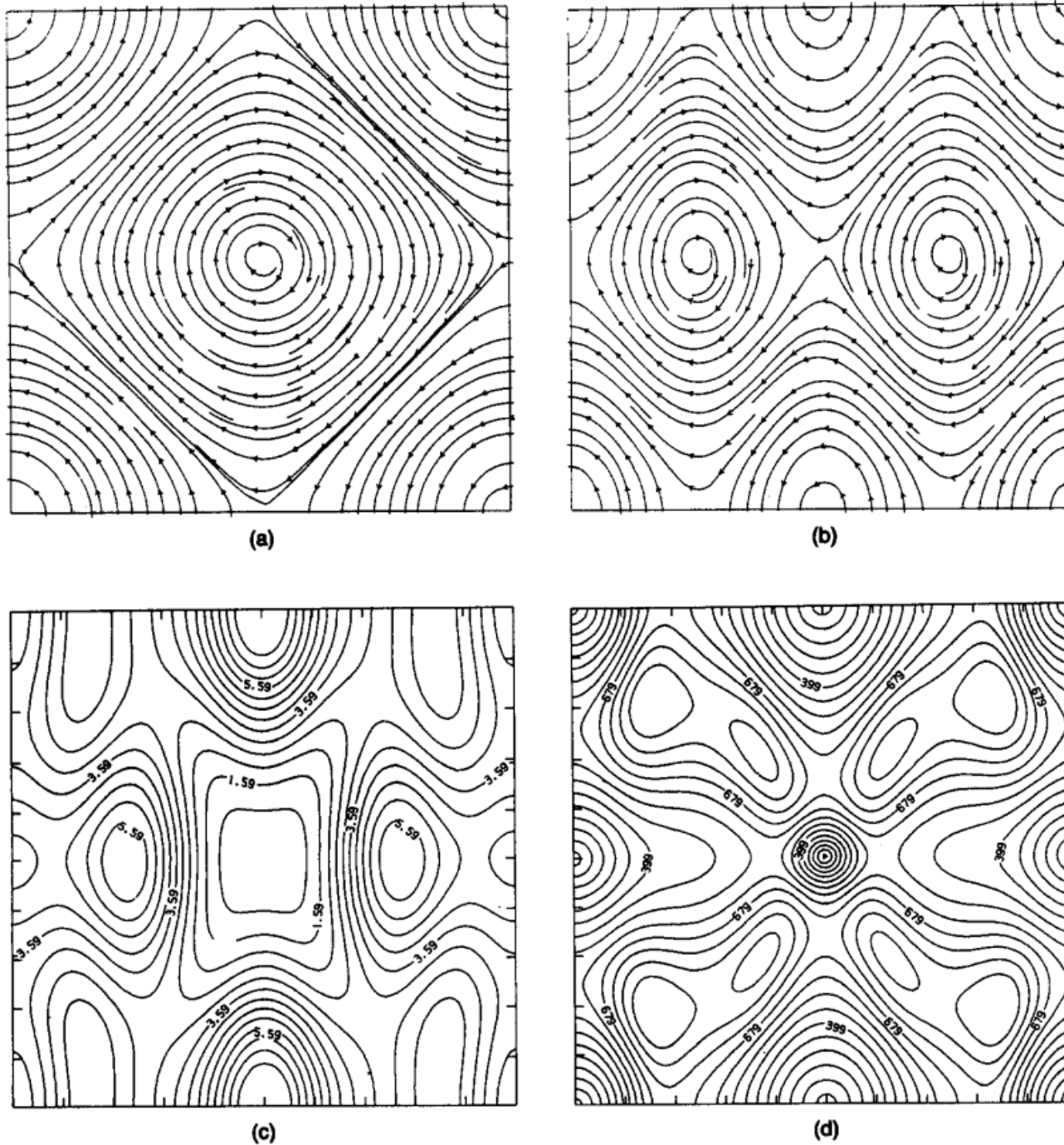
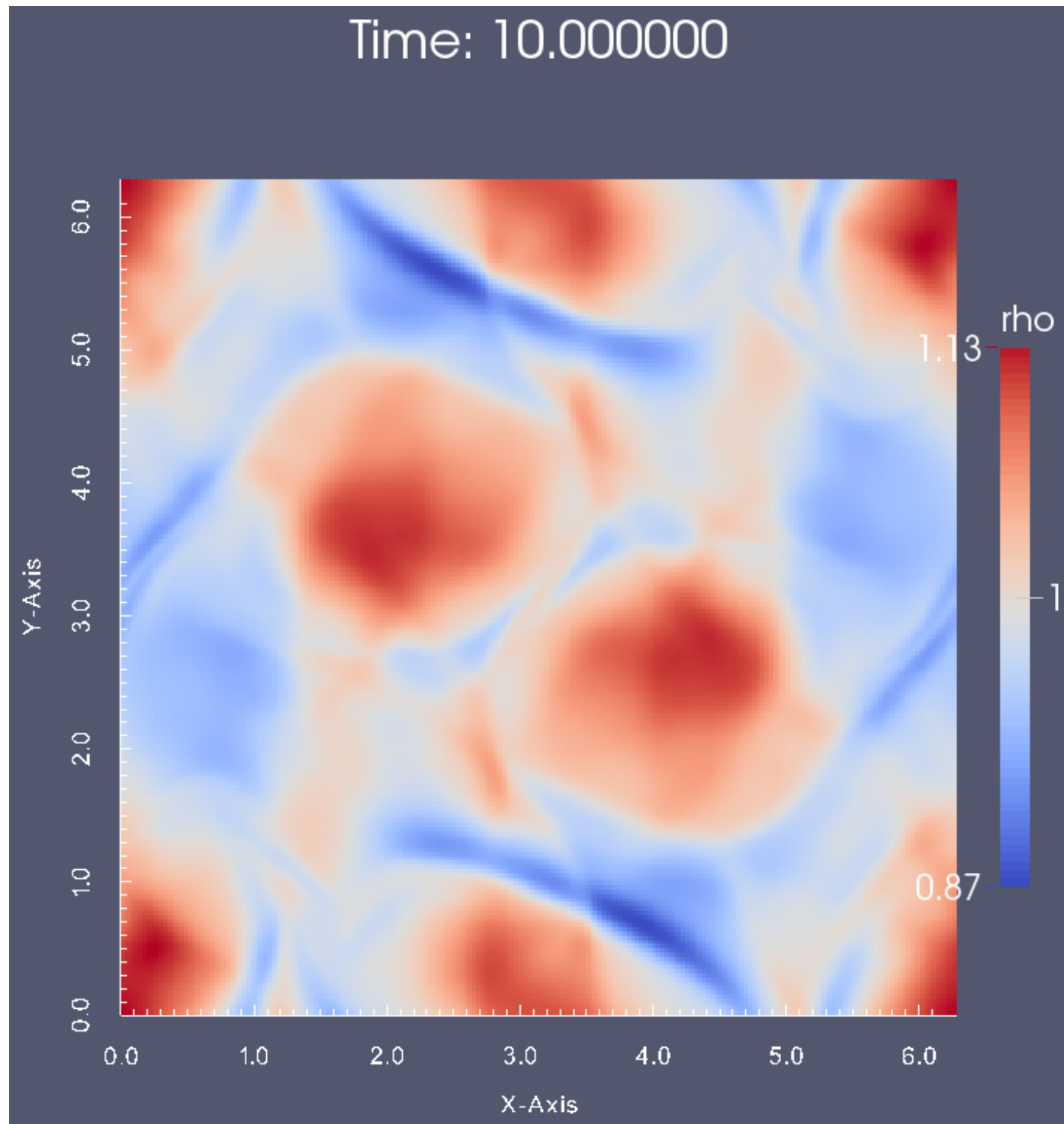
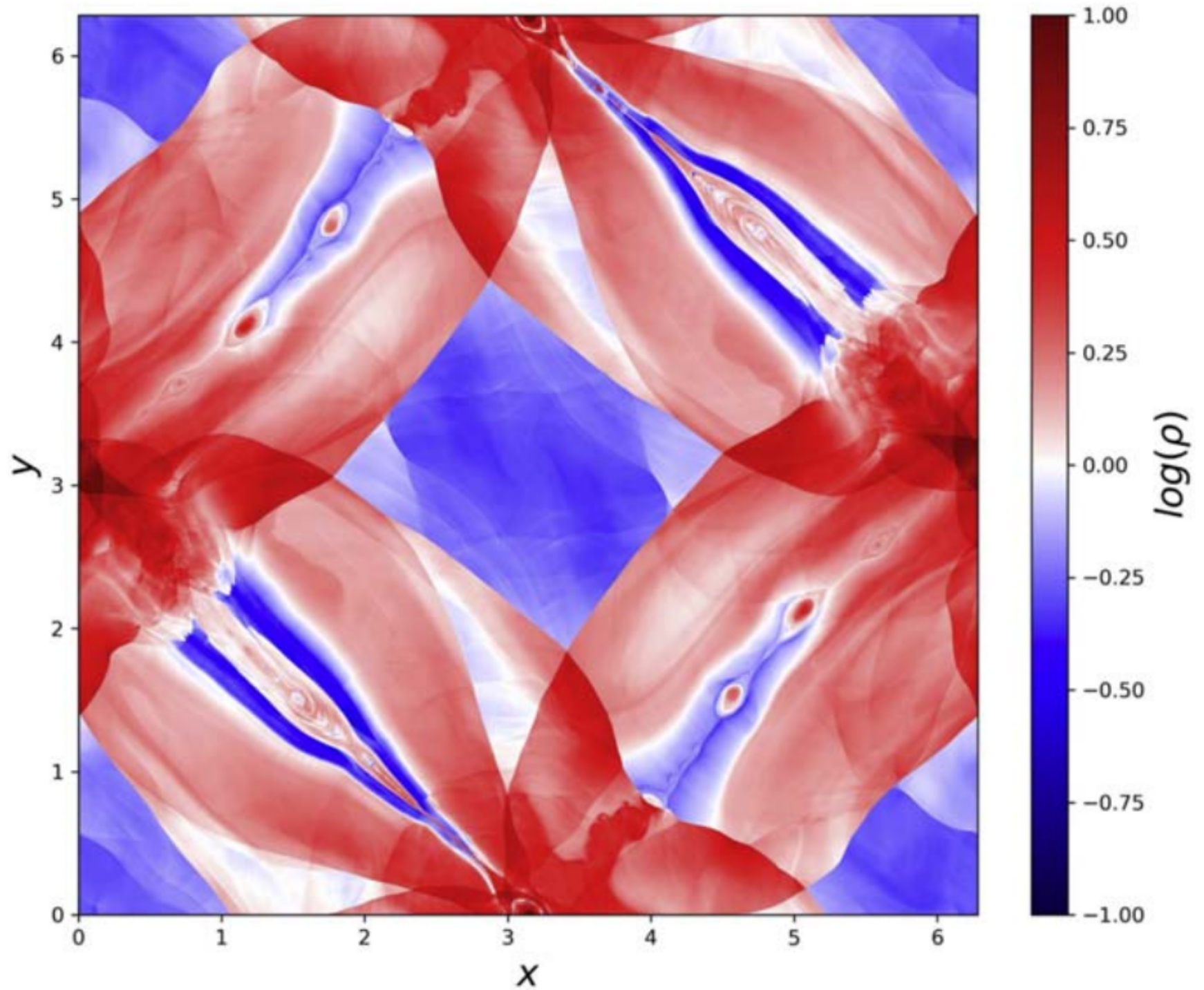


FIG. 1. Initial conditions: (a) velocity field; (b) magnetic field; initial conditions specific to the $M = 0.6$ case: (c) thermal pressure field; (d) local Mach number. Minimum and maximum values are 0.73 and 7.9 for thermal pressure and 0.0 and 0.97 for local Mach number.

Visualization and animation with Paraview, VisIt



OT, standard test but not so standard results



OT, standard test but not so standard results

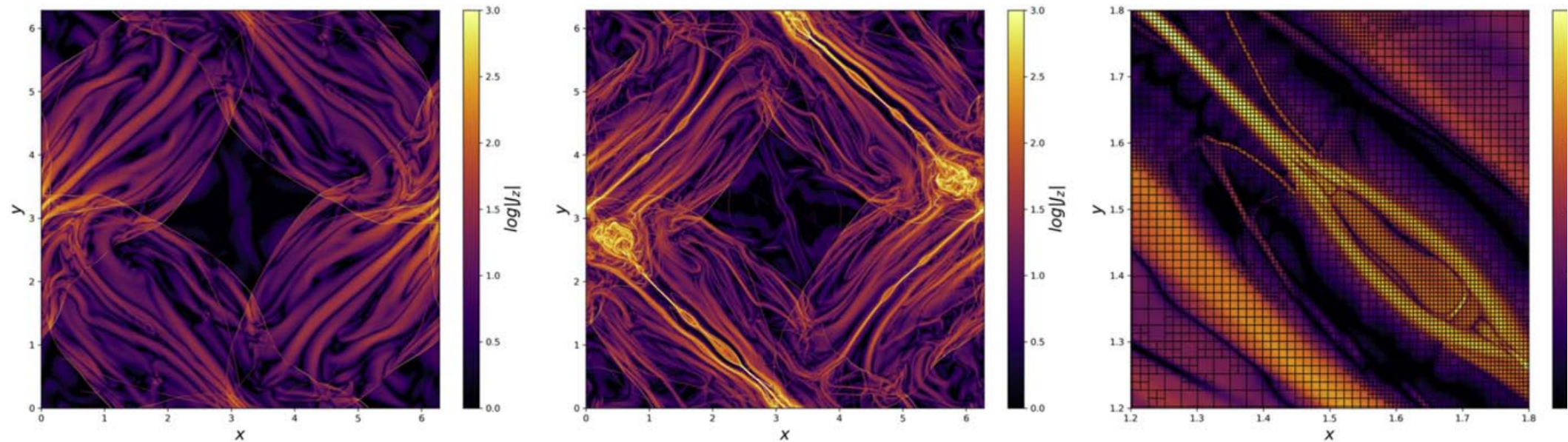


Figure 2. Logarithm of the out-of-plane current density magnitude $|J_z|$ at $t = 10t_c$ for $S = 2 \times 10^3$ (left), $S = 2 \times 10^4$ (middle), and a zoom into the current sheet $S = 2 \times 10^4$ showing the AMR grid blocks (right), each consisting of 8×8 cells, in black. Both cases have an effective resolution of 8192^2 cells in the domain. The plasmoid-unstable current sheet on the right is captured by more than 10 cells over its width.

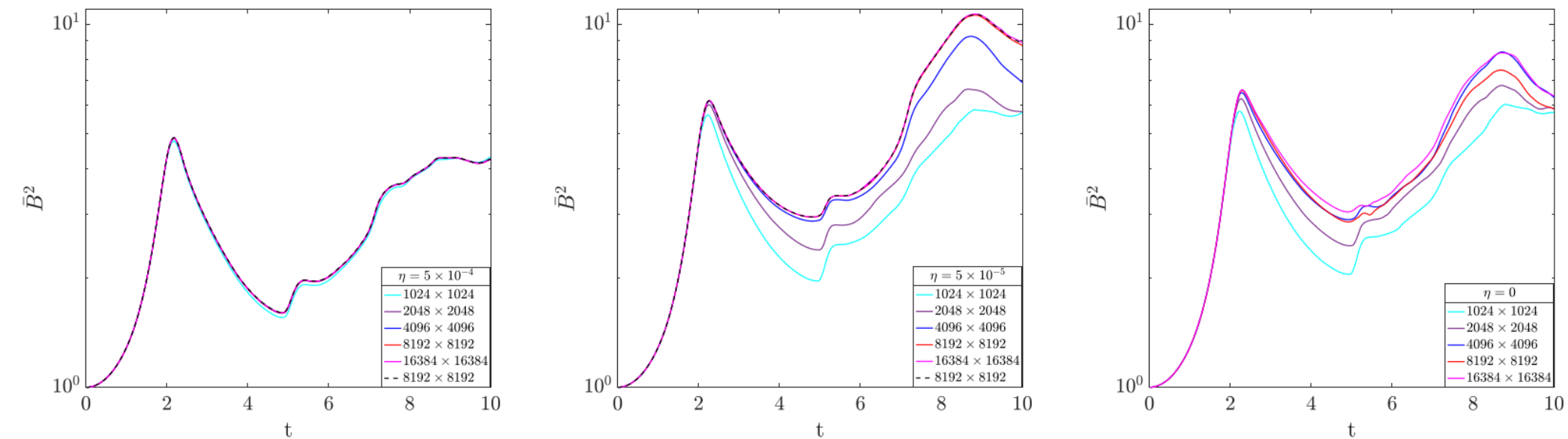


Figure 3. Evolution of the domain-averaged magnetic energy density \bar{B}^2 for $S = 2 \times 10^3$ (left), $S = 2 \times 10^4$ (middle), and the ideal case $\eta = 0$ (right) for all resolutions considered.

Summary of the Part II

- I described the setup files of PLUTO.
- We learned which are the output formats.
- We did the 2D Orszag-Tang test.
- We learned to use Paraview and VisIt.
- We learned how to animate the results with Paraview.

Outline, Part III

- A very brief introduction to history of accretion disk numerics.
- Introduction to HD accretion disk simulation.
- Setup of axisymmetric 2D(=2.5D) HD disk.
- Introduction to star-disk magnetospheric interaction simulations.
- Setup of axisymmetric 2D setup with dipole magnetic field.
- Visualization of magnetic field lines with Paraview.

Introduction of accretion disk simulations

- First numerical solution of (HD) accretion disk was by Prendergast & Burbidge (1968) (no pics!)
- Analytical solution was given by Shakura & Sunyaev (1973), with alpha-viscosity prescription.
- From that time to the 1990-ies many developments and models, with different approximations.
- In Kluźniak & Kita (2000) was given a solution of the HD disk in the full 3D. It was obtained by the method of asymptotic approximation (Taylor expansion).
- In 2009, numerical simulations of star-disk magnetospheric interaction were done in 2D-axisymmetric simulations, by Romanova et al. (2009, 2013, with non-public code), Zanni & Ferreira (2009, 2013, with modification of a publicly available code PLUTO v. 3.0).
- Development of disk simulations in the direction of MRI in the disk (Flock et al.), but nothing much in star-disk magnetospheric simulations for a decade.
- In Čemeljić et al. (2017) and Čemeljić (2019, “Atlas” paper), the first repeating of Zanni et al. (2009, 2013) axi-symmetric viscous & resistive MHD simulations in 2D with PLUTO code, v.4.1. The results are similar to Romanova et al., obtained with their (non-public) code with entropy conservation, not energy.
- In “Atlas” paper I performed a parameter study to investigate the influence of different parameters. This publication, especially its Appendix, is used here as a guide through setup and numerical methods.

Numerical methods in PLUTO for star-disk interaction

•I will show you later in the code where is definition of each of the next entries in the setup, this is an example of what you can usually read in the description of a setup:

- Simulations were performed using the second-order piecewise linear reconstruction.
- Van Leer limiter, which is more diffusive and enhances stability, is used in density and magnetic field and a minmod (monotonized central differences) limiter in pressure and velocity.
- An approximate Roe solver (hlld in the pluto options) was used, with a modification in the flag_shock subroutine: flags were set to switch to more diffusive hll solver when the internal energy was lower than 1% of the total energy, instead of switching in the presence of shocks.
- The second-order time-stepping (RK2) was employed.
- $\nabla \cdot \mathbf{B} = 0$ was maintained by the constrained transport method.
- The magnetic field was evolved with the split-field option, so that only changes from the initial stellar magnetic field were evolved in time. This means that in the final results, one has to add magnetic dipole (or other initial field) to the solution from the code.
- The power-law cooling is introduced to account for the disk dissipative heating. Physically it is good enough to represent bremsstrahlung. There are other cooling functions in PLUTO, one can also import a table, not necessarily a function.

Normalization in the code

Code works in the normalized units. For translating into other units, one needs to multiply with the corresponding scaling factors. When using Cooling, one needs to define units in definitions.h subroutine.

Table A.1. Typical values and scaling for different central objects.

	YSOs	WDs	NSs
$M_{\star}(M_{\odot})$	0.5	1	1.4
R_{\star}	$2R_{\odot}$	5000 km	10 km
P_{\star}	4.6 d	6.1 s	0.46 ms
B_{\star} (G)	500	5×10^5	10^8
ρ_{d0} (g cm^{-3})	1.2×10^{-10}	9.4×10^{-9}	4.6×10^{-6}
v_0 (km s^{-1})	218	5150	136 000
$\dot{M}_0(M_{\odot} \text{ yr}^{-1})$	5.7×10^{-7}	1.9×10^{-9}	10^{-9}
B_0 (G)	200	5×10^4	2.93×10^7

Equations solved by the code

- Code I used in “Atlas” is PLUTO (v.4.1) by Mignone et al. (2007, 2012). We will do it in v.4.4
- We will perform simulations of a thin accretion disk.
- If the disk is to reach the quasi-stationary state, the Ohmic and viscous heating in the energy equation should be balanced with the source term. In such a case, we effectively neglect the dissipative terms in the energy equation—this is equal to the assumption that all the heat is radiated away from the disk.
- Viscosity and resistivity are still affecting the solution, in the equation of motion and in the induction equation.

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

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot \left[\rho \mathbf{v} \mathbf{v} + \left(P + \frac{\mathbf{B} \cdot \mathbf{B}}{8\pi} \right) \mathbf{I} - \frac{\mathbf{B} \mathbf{B}}{4\pi} - \boldsymbol{\tau} \right] = \rho \mathbf{g}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot \left[\left(E + P + \frac{\mathbf{B} \cdot \mathbf{B}}{8\pi} \right) \mathbf{v} - \frac{(\mathbf{v} \cdot \mathbf{B}) \mathbf{B}}{4\pi} \right] + \nabla \cdot \left[\underbrace{\eta_m \mathbf{J} \times \mathbf{B} / 4\pi - \mathbf{v} \cdot \boldsymbol{\tau}}_{\text{heating terms}} \right] = \rho \mathbf{g} \cdot \mathbf{v} - \underbrace{\Lambda}_{\text{cooling}}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \times (\mathbf{B} \times \mathbf{v} + \eta_m \mathbf{J}) = 0$$

the gravity acceleration is $\mathbf{g} = -\nabla \Phi_g$, where the gravitational potential of the star with mass M_\star is equal to $\Phi_g = -GM_\star/R$. Then $g_R = -1.0/R^2$ in the code units.

$$\frac{\partial E}{\partial t} + \nabla \cdot \left[\left(E + P + \frac{\mathbf{B} \cdot \mathbf{B}}{8\pi} \right) \mathbf{u} - \frac{(\mathbf{u} \cdot \mathbf{B}) \mathbf{B}}{4\pi} \right] + \nabla \cdot [\eta_m \mathbf{J} \times \mathbf{B} / 4\pi - \mathbf{u} \cdot \boldsymbol{\tau}] = \rho \mathbf{g} \cdot \mathbf{u} - \Lambda_{\text{cool}}$$

Practical part: init.c file, Hydro-dynamical star-disk setup

The initial disk was set with the initial density ρ a self-similar profile and the aspect ratio ϵ :

$$\begin{aligned}\rho_d &= \rho_{d0} \left\{ \frac{\gamma - 1}{\gamma \epsilon^2} \left[\frac{R_\star}{R} - \left(1 - \frac{\gamma \epsilon^2}{\gamma - 1} \right) \frac{R_\star}{R \sin \theta} \right] \right\}^{1/(\gamma-1)} \\ &= \rho_{d0} \left\{ \frac{2}{5 \epsilon^2} \left[\frac{R_\star}{R} - \left(1 - \frac{5}{2} \epsilon^2 \right) \frac{R_\star}{R \sin \theta} \right] \right\}^{3/2}.\end{aligned}$$

The pressure was

$$\begin{aligned}P_d &= \epsilon^2 \rho_{d0} v_{K\star}^2 \left(\frac{\rho_d}{\rho_{d0}} \right)^\gamma = \\ &= \frac{\rho_{d0} v_{K\star}^2}{\epsilon^3} \left\{ \frac{2}{5} \left[\frac{R_\star}{R} - \left(1 - \frac{5 \epsilon^2}{2} \right) \frac{R_\star}{R \sin \theta} \right] \right\}^{5/2}\end{aligned}$$

The initial disk velocity profile is, following KK00:

$$\begin{aligned}v_{Rd} &= -\alpha_v \epsilon^2 \left[10 - \frac{32}{3} \Lambda \alpha_v^2 - \Lambda \left(5 - \frac{1}{\epsilon^2 \tan^2 \theta} \right) \right] \sqrt{\frac{GM_\star}{R \sin^3 \theta}}, \\ v_{R\phi} &= \left[\sqrt{1 - \frac{5 \epsilon^2}{2}} + \frac{2}{3} \epsilon^2 \alpha_v^2 \Lambda \left(1 - \frac{6}{5 \epsilon^2 \tan^2 \theta} \right) \right] \sqrt{\frac{GM_\star}{R \sin \theta}}\end{aligned}\quad (\text{A.10})$$

where

$$\Lambda = \frac{11}{5} / \left(1 + \frac{64}{25} \alpha_v^2 \right).\quad (\text{A.11})$$

In the corona, density and pressure are:

$$\begin{aligned}\rho_c &= \rho_{c0} (R_\star / R)^{1/(\gamma-1)}, \\ P_c &= \rho_{c0} \frac{\gamma - 1}{\gamma} \frac{GM_\star}{R_\star} \left(\frac{R_\star}{R} \right)^{\gamma/(\gamma-1)}\end{aligned}$$

Viscosity tensor is given by Landau-Lifshitz def:

$$\tau = \eta_v \left[(\nabla \mathbf{v}) + (\nabla \mathbf{v})^T - \frac{2}{3} (\nabla \cdot \mathbf{v}) \mathbf{I} \right],\quad (\text{A.7})$$

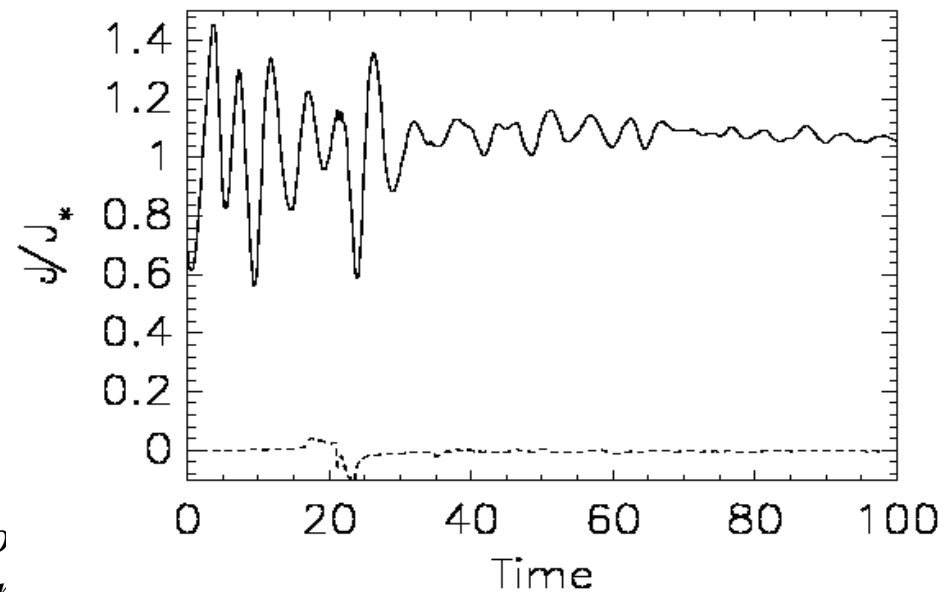
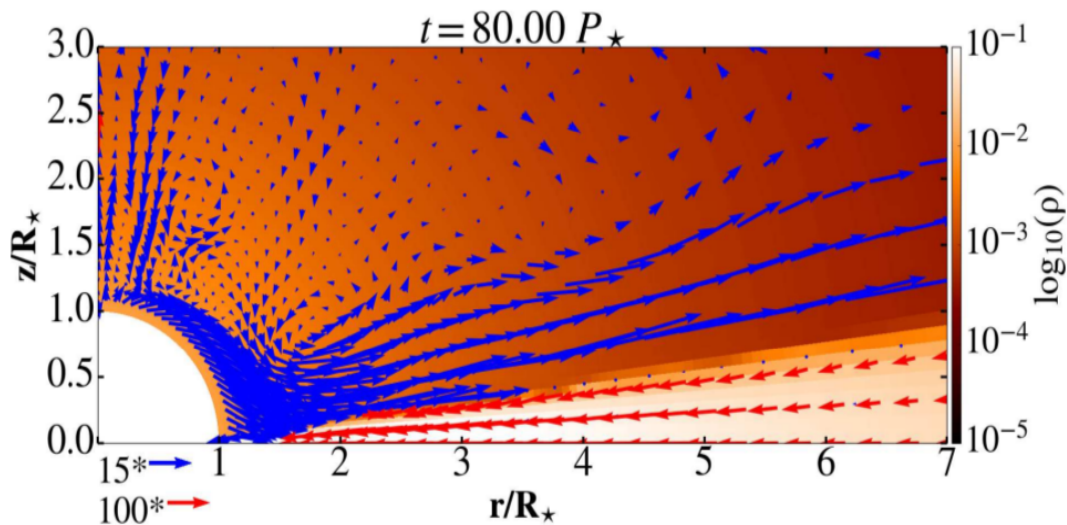
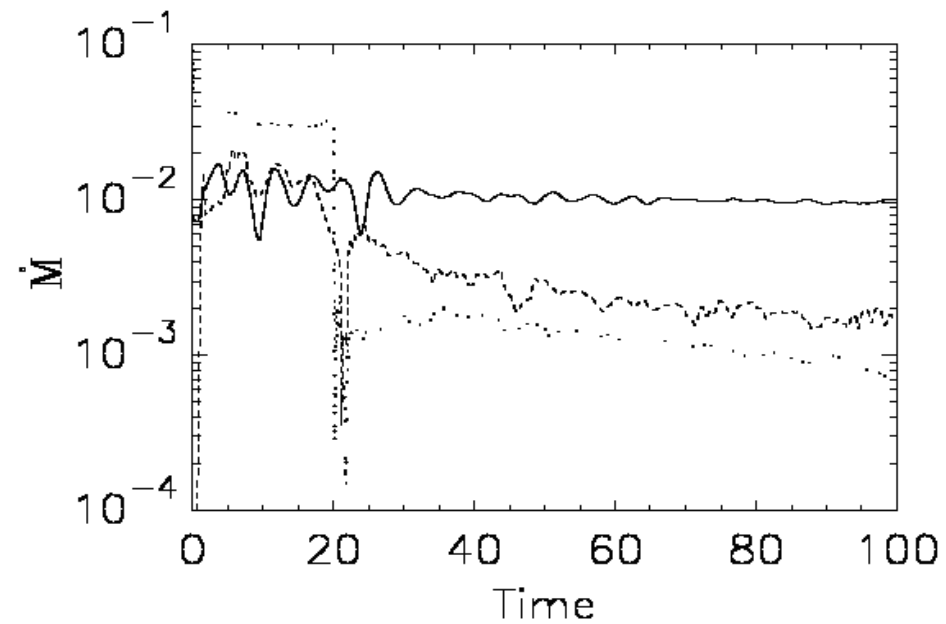
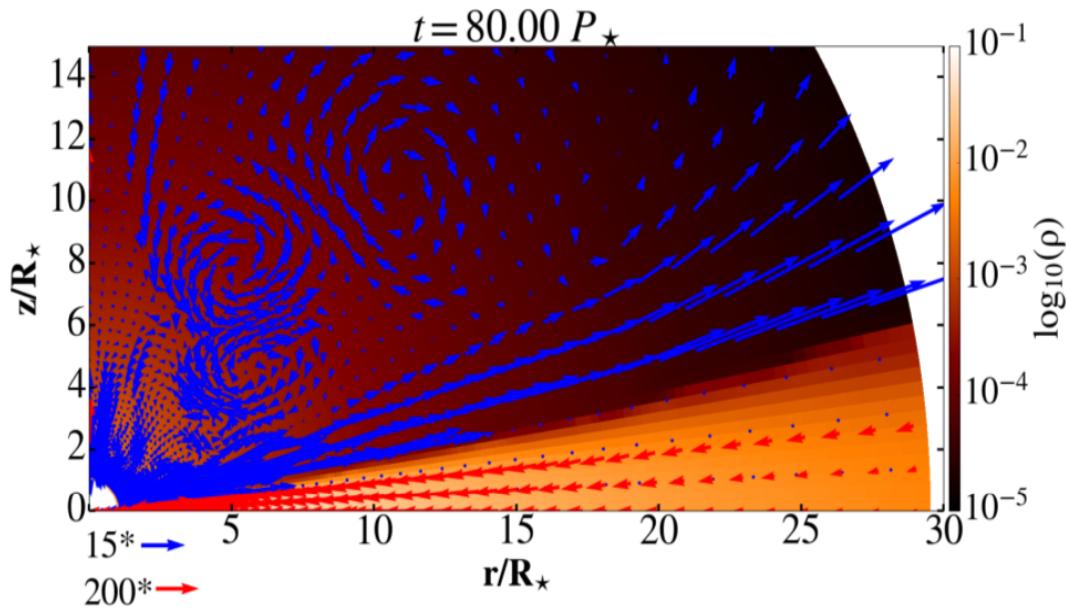
with the dynamic viscosity $\eta_v = \rho \nu_v$ given with

$$\eta_v = \frac{2}{3} \rho \alpha_v \left[c_s^2(r)|_{z=0} + \frac{2}{5} \left(\frac{GM_\star}{R} - \frac{GM_\star}{r} \right) \right] \sqrt{\frac{r^3}{GM_\star}},\quad (\text{A.8})$$

where ν_v is the kinematic viscosity. The magnetic diffusivity was assumed proportional to the viscosity, with the free parameter

$$\nu_m = \frac{3}{2} \alpha_m \frac{\nu_v}{\alpha_v},$$

Practical part: results in Hydro-dynamical star-disk simulations

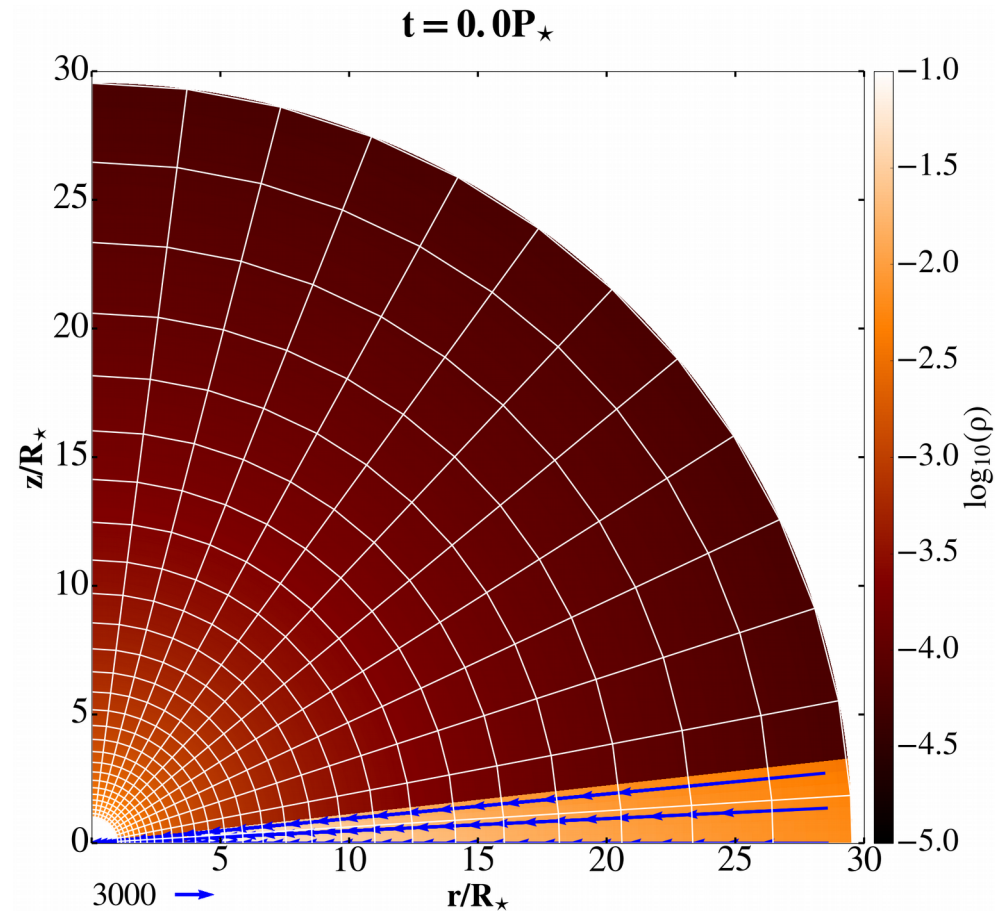


Ideally, this is what we wish to obtain. Computational box and a zoom closer to the star after 80 stellar rotations. In color is shown the density, and vectors show velocity, with the different normalization in the disk and stellar wind.

Time dependence of the mass and angular momentum fluxes in the various components in our simulations.

Star-disk simulations setup

- The Kluźniak-Kita (2000) analytical solution is purely HD.
- We will start with the HD disk, and then add the magnetic field.
- Stellar surface is a rotating boundary around the origin of the spherical computational domain. In the non-magnetic setup, it is just a simple setup with “absorption” of the flow atop the stellar surface—we assume the star to be a (differentially) rotating magnetized rotator.
- The initial corona is a non-rotating corona in a hydrostatic balance with the 100 times denser disk below it. After short relaxation, lasting for the few stellar rotations, corona starts rotating, following the disk.



Logarithmic grid in a star-disk simulations setup

Pluto44usgui.pdf

31 (32 of 151) 202.599

Index	x
Quick Start	6
Introduction	11
Problem Header Fi...	17
Makefile Selection...	27
Runtime initializat...	29
The [Grid] block	30
The [Chombo Refi...	32
The [Time] Block	33
The [Solver] Block	34
The [Boundary] Bl...	35
The [Static Grid O...	36
The [Chombo HD...	38
The [Particles] Bl...	38
The [Parameters] ...	39
Initial and Boundar...	40
Initial Conditions: ...	40
Units and Dimen...	42
Specifying Temp...	43
Initial Conditions:...	45
Assigning Initial ...	47
User-defined Bou...	48

In practice, the mesh spacing in the $l+$ grid is obtained by reversing the spacing in the $l-$ grid.

Note: The interval should not include the origin when using a logarithmic grid.

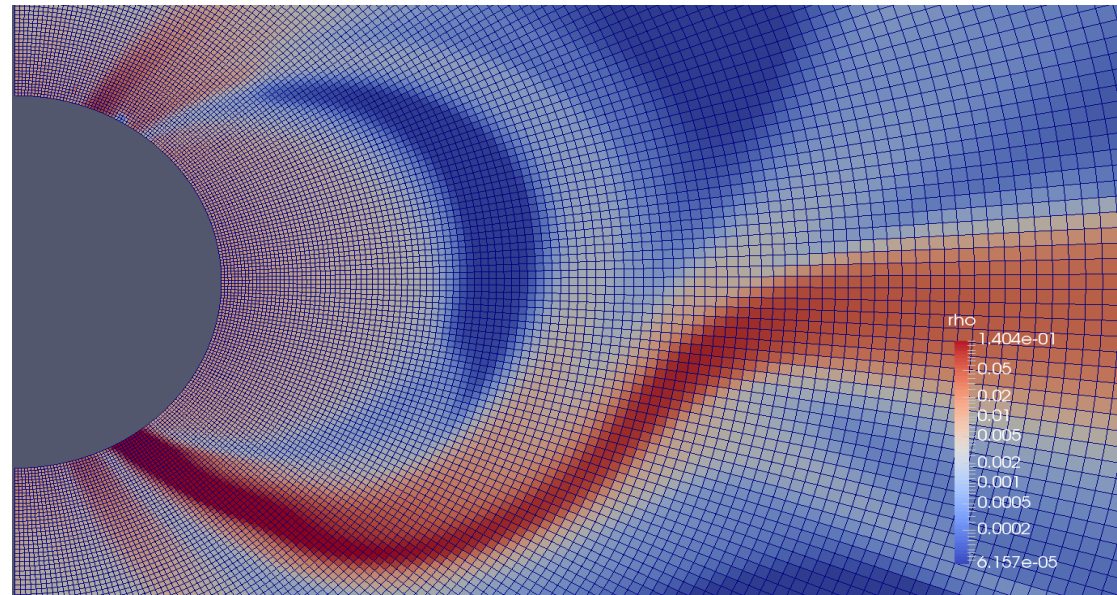
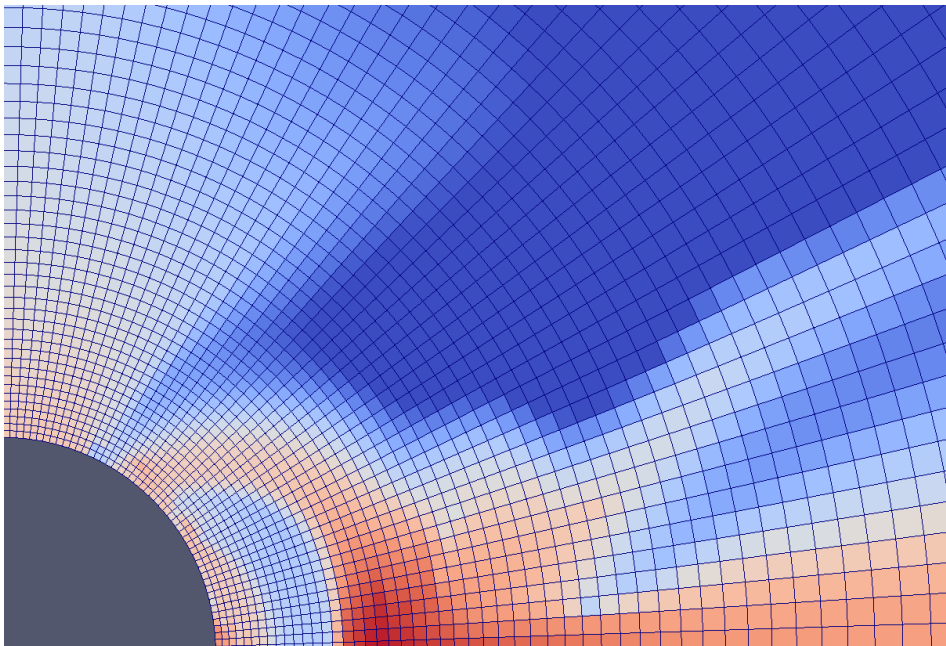
In *CYLINDRICAL* or *SPHERICAL* coordinates, a radial logarithmic grid has the advantage of preserving the cell aspect ratio at any distance from the origin. In addition, the condition to obtain approximately squared cells (aspect ratio ≈ 1) is $\Delta r_1 \approx r_1 \Delta \phi$ where $\Delta r_1 = r_L (e^{\Delta \xi} - 1)$ is the radial spacing of the first active computational zone. This condition can be used to determine either the number of points in the radial direction or the endpoint:

$$\log_{10} \frac{r_R}{r_L} = N_r \log_{10} \frac{2 + \Delta \phi}{2 - \Delta \phi}.$$

Beware that non-uniform grids may introduce extra dissipation in the algorithm. Changes in the grid spacing are correctly accounted for when `RECONSTRUCTION` is set to either *LINEAR*, *PARABOLIC* or *WENO3*.

Resolution needed in a setup

- Resolution in my production runs is $R \times \vartheta = [217 \times 100]$ grid cells in $\vartheta = [0, \pi/2]$, with a logarithmic grid spacing in the radial direction. For testing I also use $R \times \vartheta = [109 \times 50]$ grid cells, which gives qualitatively correct results.
- The accretion column is well resolved if a rule of thumb is satisfied that there is at least that many grid cells how many there is independent variables (5 in HD case, 8 in MHD case in our setups: density, pressure, 3 components of \mathbf{v} and \mathbf{B}).
- Star rotates at about 1/10 of the breakup rotational velocity.
- I did also $\vartheta = [0, \pi]$ cases in $R \times \vartheta = [217 \times 200]$ grid cells, as well as $R \times \vartheta = [109 \times 100]$ grid cells. Now there is no need to define the equatorial boundary condition, the simulation self-consistently computes across the domain. This case usually leads to an asymmetry with respect to the equatorial plane-I will show more on this in the last lecture.



Star-disk: pluto.ini

```
I pluto.ini (ini) Row 22 Col 1 2:51 Ctrl-K H for help
[Grid]
X1-grid 1 1.0 109 l+ 30.
X2-grid 1 0.0 50 u 1.570796327
X3-grid 1 0.0 1 u 1.0

[Chombo Refinement]
Levels 4
Ref_ratio 2 2 2 2 2
Regrid_interval 2 2 2 2
Refine_thresh 0.3
Tag_buffer_size 3
Block_factor 8
Max_grid_size 64
Fill_ratio 0.75

[Time]
CFL 0.4
CFL_max_var 1.2
tstop 628.0
first_dt 1.e-6

[Solver]
Solver roe

[Boundary]
X1-beg userdef
X1-end userdef
X2-beg axisymmetric
** Joe's Own Editor v4.1 ** (utf-8) ** Copyright © 2015 **
```

```
I pluto.ini (ini) Row 57 Col 1 2:52 Ctrl-K H for help
X2-end eqtsymmetric
X3-beg outflow
X3-end outflow

[Static Grid Output]
uservar 3 nu num Te
dbl 6.279 -1 single_file
flt -1.0 -1 single_file
vtk 6.279 -1 single_file
tab -1.0 -1
dbl.h5 -6.279 -1
ppm -1.0 -1
png -1.0 -1
log 100
analysis -1.0 -1

[Chombo HDF5 output]
Checkpoint_interval -1
Plot_interval 1.0

[Parameters]
ALPHAM 0.4
MU 0.1
TEMPF 300.0
RHOC 0.01
RD 1.7
EPS 0.1
OMG 0.1
ALPHAV 1.0
DFLOOR 5.e-7
```

Star-disk: definitions.h

```
I A definitions.h (c) Row 3 Col 1 2:53 Ctrl-K H for help
#define PHYSICS HD
#define DIMENSIONS 2
#define COMPONENTS 3
#define GEOMETRY SPHERICAL
#define BODY_FORCE VECTOR
#define FORCED_TURB NO
#define COOLING NO
#define RECONSTRUCTION LINEAR
#define TIME_STEPPING RK2
#define DIMENSIONAL_SPLITTING NO
#define NTRACER 1
#define USER_DEF_PARAMETERS 9

/* -- physics dependent declarations -- */

#define DUST_FLUID NO
#define EOS IDEAL
#define ENTROPY_SWITCH NO
#define THERMAL_CONDUCTION NO
#define VISCOSITY EXPLICIT
#define ROTATING_FRAME NO

/* -- user-defined parameters (labels) -- */

#define ALPHAM 0
#define MU 1
#define TEMPF 2
#define RHOC 3
#define RD 4
#define EPS 5
#define OMG 6
#define ALPHAV 7
#define DFLOOR 8
** Joe's Own Editor v4.1 ** (utf-8) ** Copyright © 2015 **
```

```
I A definitions.h (c) Row 18 Col 1 2:53 Ctrl-K H for help
#define DUST_FLUID NO
#define EOS IDEAL
#define ENTROPY_SWITCH NO
#define THERMAL_CONDUCTION NO
#define VISCOSITY EXPLICIT
#define ROTATING_FRAME NO

/* -- user-defined parameters (labels) -- */

#define ALPHAM 0
#define MU 1
#define TEMPF 2
#define RHOC 3
#define RD 4
#define EPS 5
#define OMG 6
#define ALPHAV 7
#define DFLOOR 8

/* [Beg] user-defined constants (do not change this line) */

#define WARNING_MESSAGES NO
#define INTERNAL_BOUNDARY YES
#define SHOCK_FLATTENING MULTID
#define CT_EN_CORRECTION YES
#define UNIT_DENSITY 8.5e-11
#define UNIT_LENGTH 1.392e11
#define UNIT_VELOCITY 2.1839e7
#define VTK_VECTOR_DUMP YES
#define CT_EMF_AVERAGE ARITHMETIC
#define LIMITER VANLEER_LIM

/* [End] user-defined constants (do not change this line) */
```

Star-disk: init.c

```
I A init.c (c) Row 1 Col 1 2:55 Ctrl-K H for help
/* //////////////////////////////////////////////////////////////////// */
/*
 \file
 \brief Contains basic functions for problem initialization.

The init.c file collects most of the user-supplied functions useful
for problem configuration.
It is automatically searched for by the makefile.

 \author A. Mignone (mignone@ph.unito.it)
 \date Sep 10, 2012

 \modified by M. Cemeljic (miki@camk.edu.pl)
 \date July 2020
Below is given Miki's stripped, minimal version of the setup in ideal MHD,
for teaching purpose in SHA0. It is not meant to produce quasi-stationary
or any publication-worthy result, but to provide a platform for teaching.
It is based on the setup described in Appendix of "Atlas" paper,
Cemeljic, 2019, A&A, 624, A31.
*/
/* //////////////////////////////////////////////////////////////////// */
#include "pluto.h"

/* ***** */
void Init (double *v, double x1, double x2, double x3)
/*
***** */
{
double coeff, eps2, pc, rcyl;
double br, bth;
double lambda;
double xhi2, Rco;

I A init.c (c) void Init (double *v, double x1, double x2, double x3) Row 36 Col 1 2:56 Ctrl-K H for help
rcyl=x1*sin(x2);
eps2=g_inputParam[EPS]*g_inputParam[EPS];
coeff=2./5./eps2*(1./x1-(1.-5./2.*eps2)/rcyl);
lambda=11./5./(1.+64./25.*g_inputParam[ALPHAV]*g_inputParam[ALPHAV]);

/* initial non-rotating adiabatic corona in hydrostatic equilibrium */
v[RHO] = g_inputParam[RHOC]*pow(x1,-3./2.);
v[PRS] = 2./5.*g_inputParam[RHOC]*pow(x1,-5./2.);

pc=v[PRS];

v[VX1] = 0.0;
v[VX2] = 0.0;
v[VX3] = 0.0;

/* Keplerian adiabatic disk in vertical pressure equilibrium with the
adiabatic corona, as given by Kluzniak & Kita (2000) */

v[PRS]=eps2*pow(coeff,5./2.);

if (v[PRS] >= pc && rcyl > g_inputParam[RD])
{v[RHO] = pow(coeff,3./2.);
v[VX1] = -g_inputParam[ALPHAV]/sin(x2)*eps2*(10.-32./3.
*lambda*g_inputParam[ALPHAV]*g_inputParam[ALPHAV]
-lambda*(5.-1./(eps2*tan(x2)*tan(x2)))/sqrt(rcyl);
v[VX3] = (sqrt(1.-5./2.*eps2)+2./3.*eps2
*g_inputParam[ALPHAV]*g_inputParam[ALPHAV]
*lambda*(1.-6./(5.*eps2*tan(x2)*tan(x2)))/sqrt(rcyl);
v[TRC] = 1.0; /* Track the disc material */
}
else
{
v[PRS]=2./5.*g_inputParam[RHOC]*pow(x1,-5./2.);
v[TRC] = 0.0; /* Track the corona */
}
}
}
```

Summary of the Part III

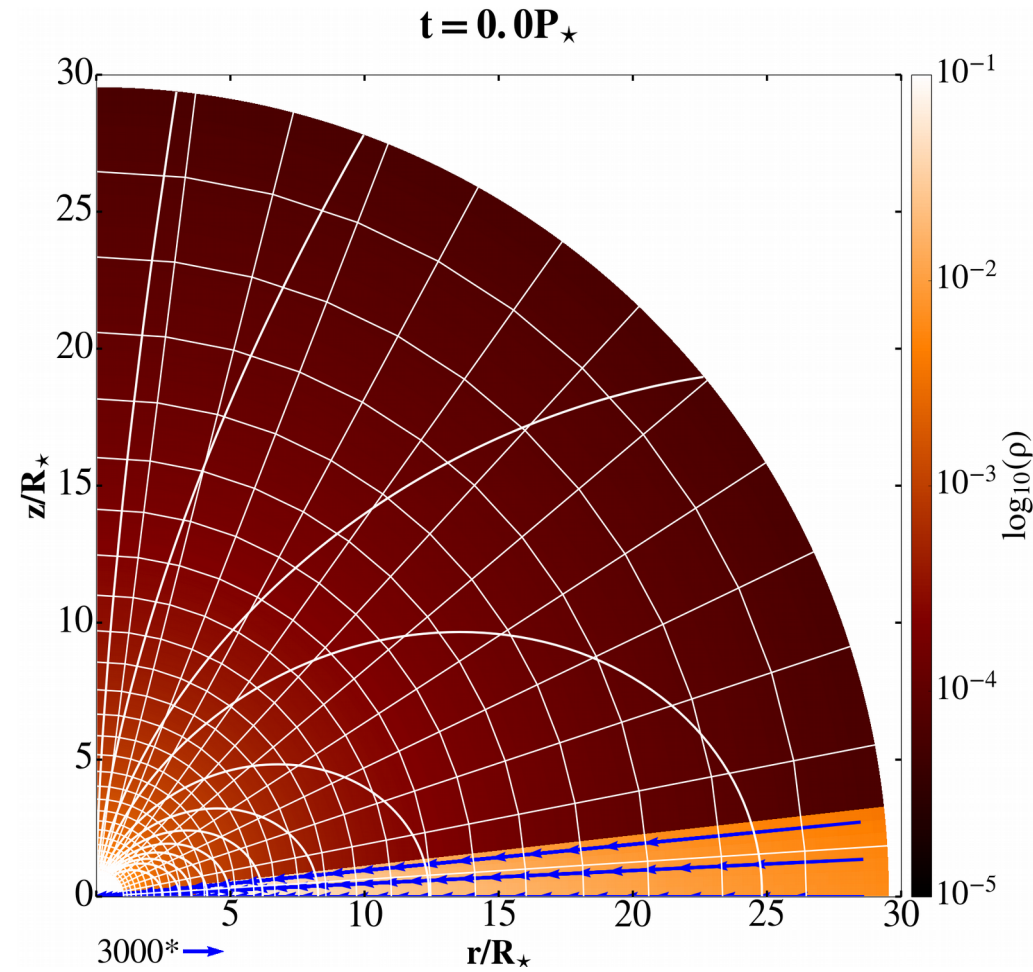
- A very brief introduction to history of accretion disk numerics.
- Introduction to HD accretion disk simulation.
- Setup of axisymmetric 2D(=2.5D) HD disk.
- Introduction to star-disk magnetospheric interaction simulations.
- Setup of axisymmetric 2D setup with dipole magnetic field.
- Visualization of magnetic field lines with Paraview.

Outline, Part IV

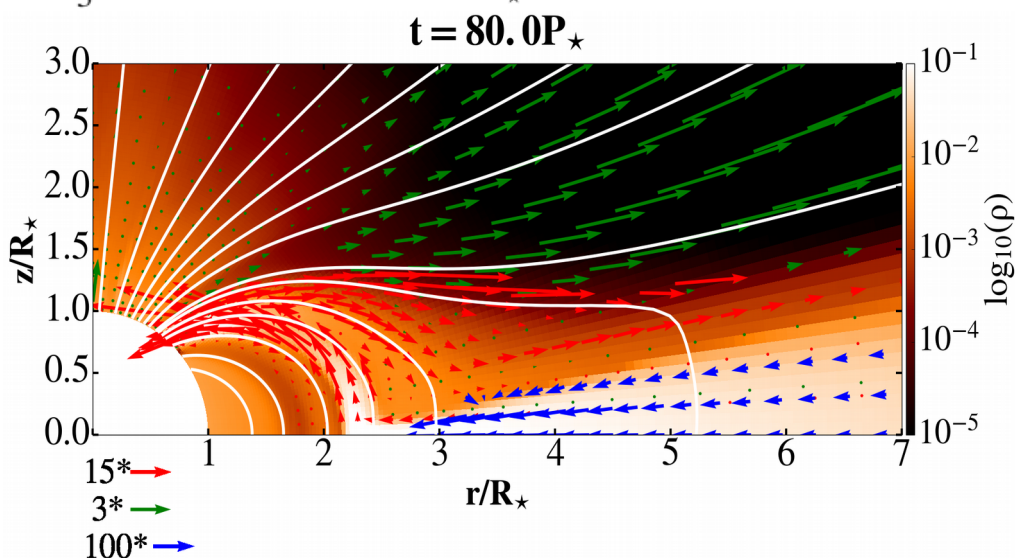
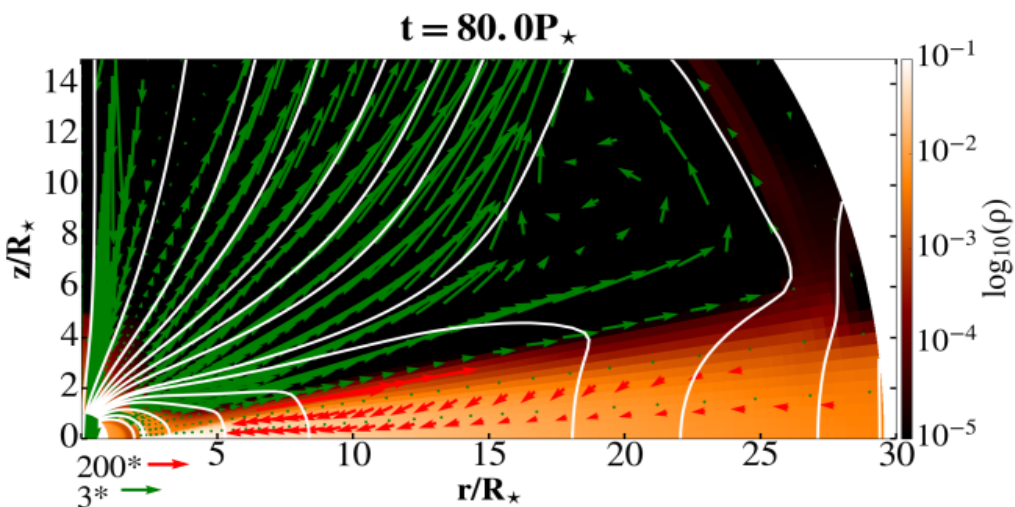
- Introduction to star-disk magnetospheric interaction simulations.
- Setup of axisymmetric 2D setup with dipole magnetic field.
- Visualization of magnetic field lines with Paraview.
- Simulations in 2.5D with an axial outflow.
- Python scripts for visualization.
- Setup of full 3D HD disk.
- Setup of full 3D magnetic disk with dipole magnetic field.
- Restart and initialization from (modified) file.
- Visualization of 3D solutions with Paraview

Setup of magnetic star-disk interaction

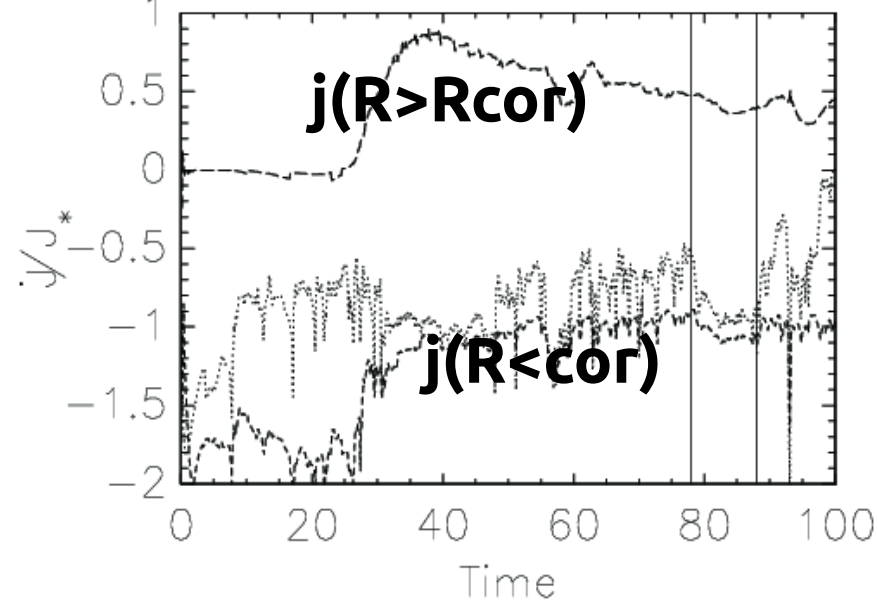
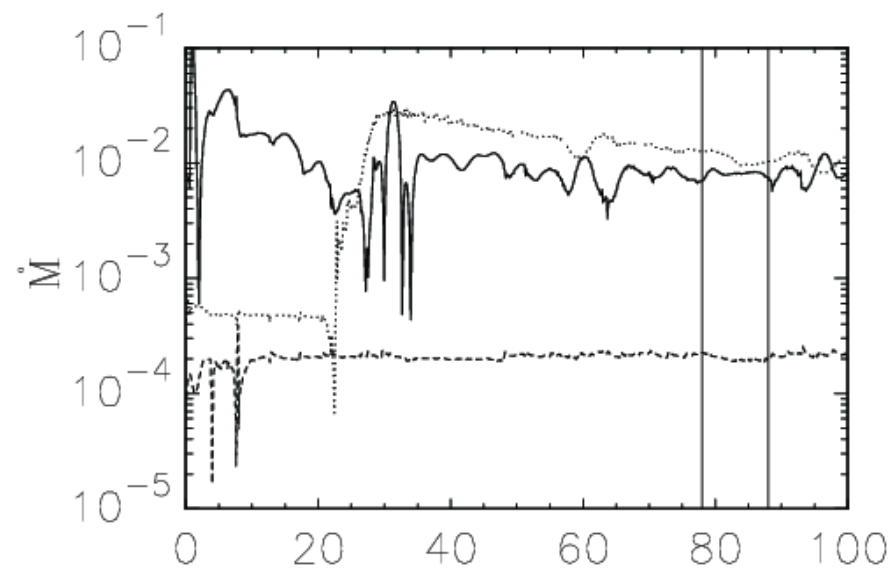
- The Kluźniak-Kita (2000) analytical solution is purely HD. We now add the stellar dipole magnetic field, with the axis matching the stellar rotation axis, so that the axi-symmetry would be preserved. Later we will set the simulations with quadrupole, octupole and mixed, multipole field.
- Stellar surface is a rotating boundary around the origin of the spherical computational domain. In the magnetic setup, it is **not** a simple setup with “absorption” of the flow atop the stellar surface—we assume the star to be a (differentially) rotating magnetized rotator. The initial corona is a non-rotating corona in a hydrostatic balance. After short relaxation, lasting for the few stellar rotations, corona starts rotating, following the disk.



Results in star-disk magnetospheric interaction simulations

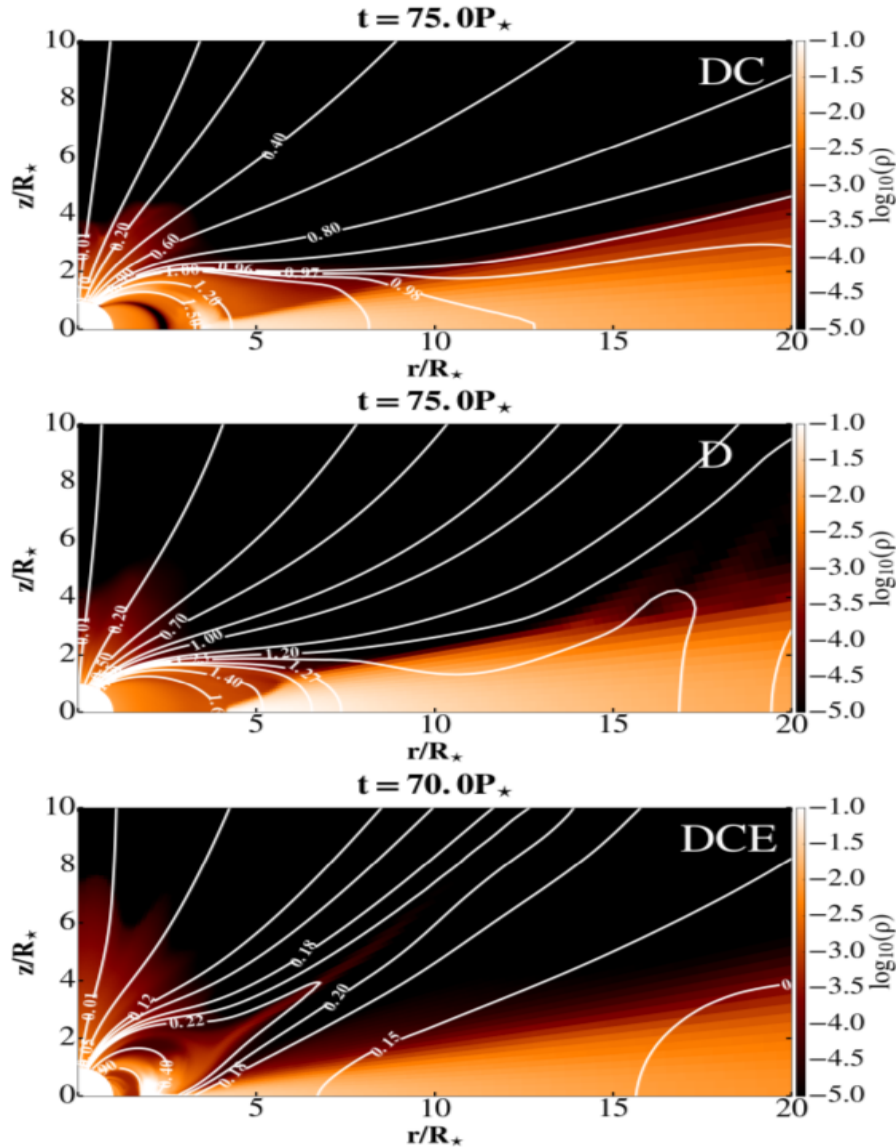


Computational box and a zoom closer to the star after 80 stellar rotations, to visualize the accretion column and the magnetic field lines (white solid lines) connected to the disk beyond the corotation radius $R_{\text{cor}}=2.92 R_{\text{s}}$. In color is shown the density, and vectors show velocity, with the different normalization in the disk, column and stellar wind.



Time dependence of the mass and angular momentum fluxes in the various components in our simulations with marked time interval in which the average value during the quasi-stationarity is computed.

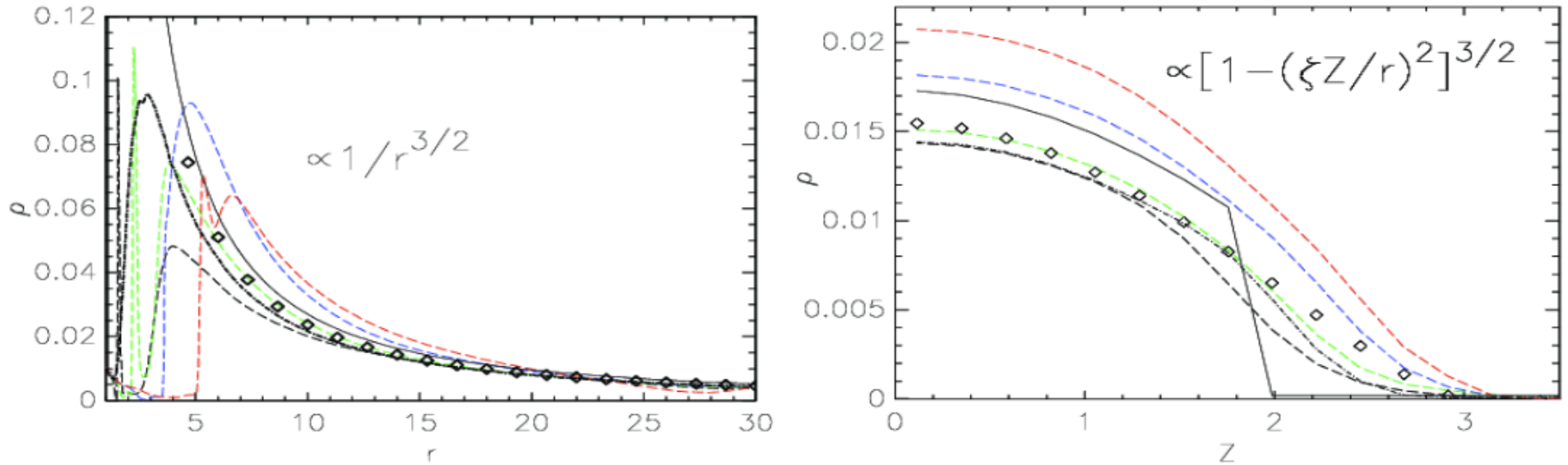
Types of solutions for slowly rotating stars in “Atlas”



Three different cases of geometry in the results. In the top and middle panels are shown $B=1 \text{ kG}$ and the resistivity $\alpha_m=1$, in the cases with $\Omega_s=0.1$ (top panel) and $\Omega_s=0.15$ (middle panel). Faster stellar rotation prevents the accretion column formation. In the bottom panel is shown the third case, with $B=0.5 \text{ kG}$, resistivity $\alpha_m=0.1$ and $\Omega_s=0.1$, where a conical outflow is formed.

$\alpha_m =$	0.1	0.4	0.7	1
$\Omega_*/\Omega_{\text{br}}$				
$B_* = 250 \text{ G}$				
0.05	DCE	DC	DC	DC
0.1	DCE	DC	DC	DC
0.15	DCE	DC	DC	DC
0.2	DCE	DC	DC	DC
$B_* = 500 \text{ G}$				
0.05	DCE	DC	DC	DC
0.1	DCE	DC	DC	DC
0.15	DCE	DC	DC	DC
0.2	DCE	DC	DC	DC
$B_* = 750 \text{ G}$				
0.05	DCE	DC	DC	DC
0.1	DCE	DC	DC	DC
0.15	DCE	DC	DC	DC
0.2	DCE	DC	DC	DC
$B_* = 1000 \text{ G}$				
0.05	DCE	DC	DC	DC
0.1	DCE	DC	DC	DC
0.15	DCE	D	D	D
0.2	DCE	D	D	D

Comparison of results with increasing mag. Field strength



Matter density in the initial set-up (thin solid line) with the quasi-stationary solutions in numerical simulations in the HD (dot-dashed line) and the MHD (long-dashed line) cases. In black, green blue and red colors are the results in the MHD cases with the stellar magnetic field strength 0.25, 0.5, 0.75 and 1.0 kG, respectively. The closest match to the 0.5 kG case is depicted with the diamond symbols. In the left panel is shown the radial dependence nearby the disk equatorial plane, and in the right panel the profiles along the vertical line at $r=15$.

Similar plots for other quantities reveal trends on the solutions with the increasing magnetic field, stellar rotation rate or dissipation coefficients.

Trends in the “Atlas” results

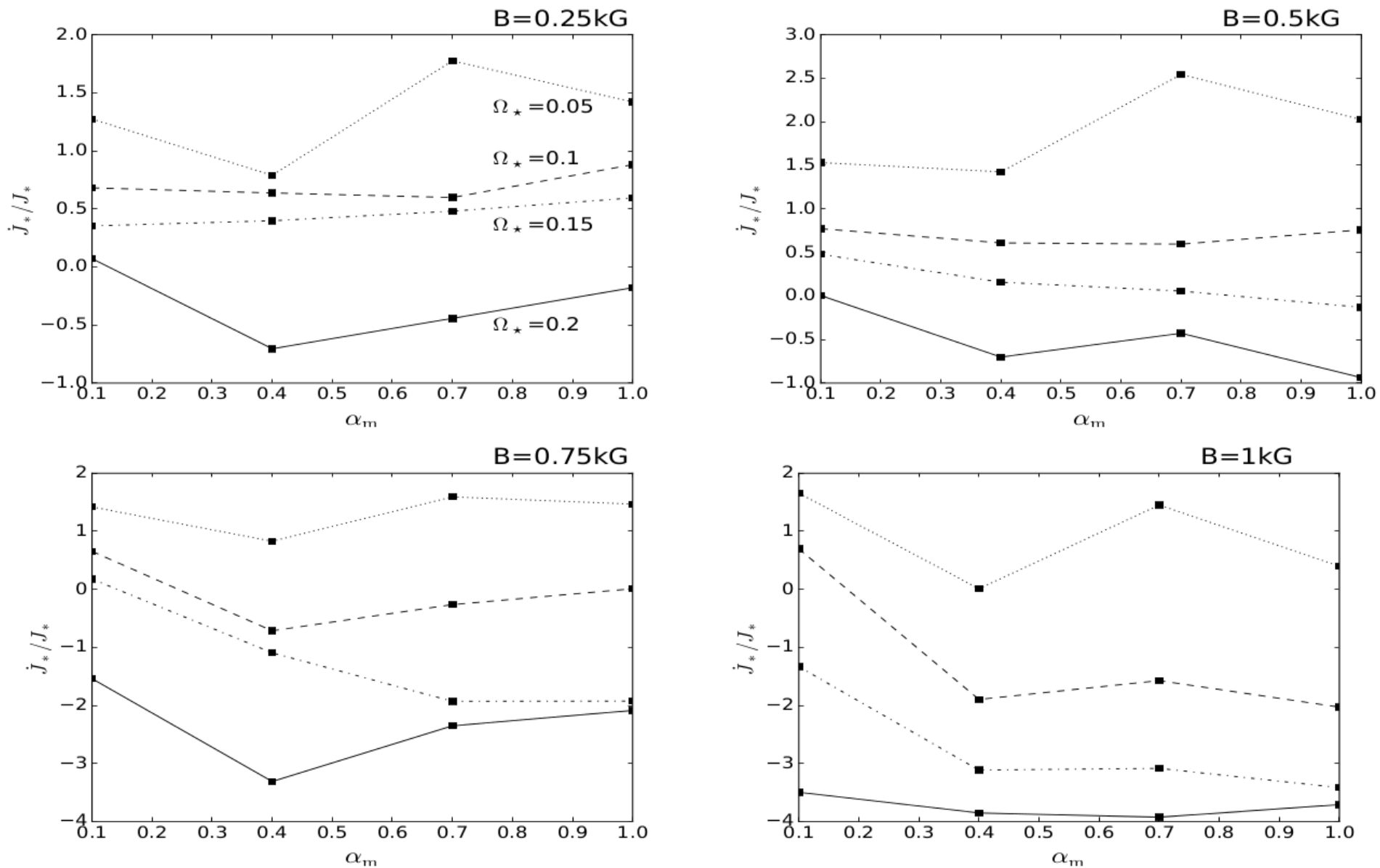


Fig. 4. Average angular momentum flux transported onto the stellar surface by the matter in-falling from the disk onto the star through the accretion column. Each panel shows a set of solutions with one stellar magnetic field strength and varying stellar rotation rate and resistivity. Results with $\Omega_*/\Omega_{\text{br}} = 0.05$ (dotted), 0.1 (dashed), 0.15 (dash-dot-dotted), and 0.2 (solid) are shown in units of stellar angular momentum $J_* = k^2 M_* R_*^2 \Omega_*$ (with $k^2 = 0.2$ for the typical normalized gyration radius of a fully convective star). A positive flux spins the star up, a negative flux slows it down. With the increase in stellar rotation rate, spin-up of the star by the infalling matter decreases and eventually switches to spin-down.

Trends in the “Atlas” results

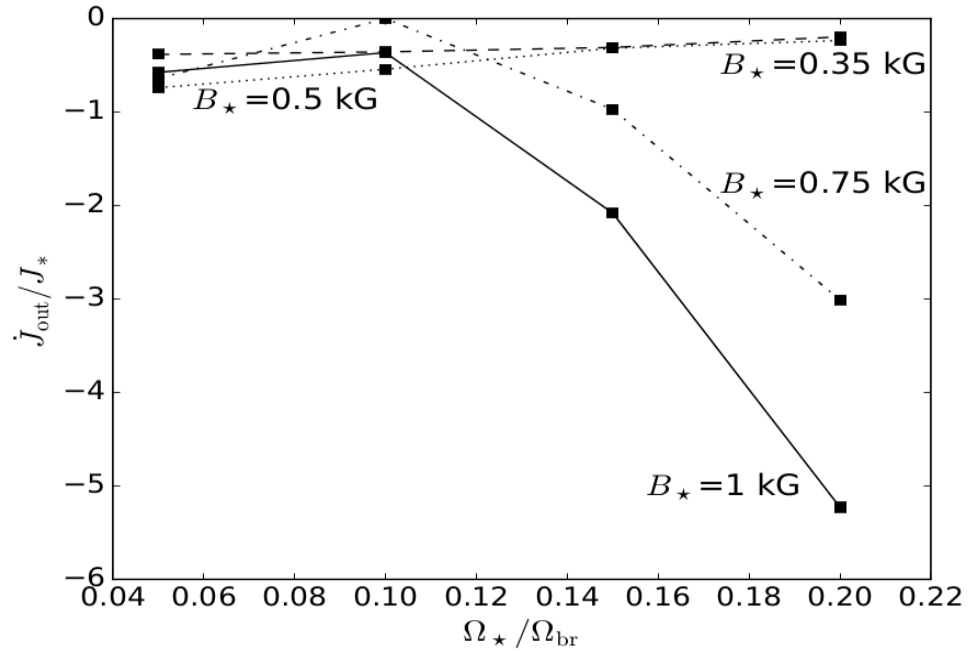
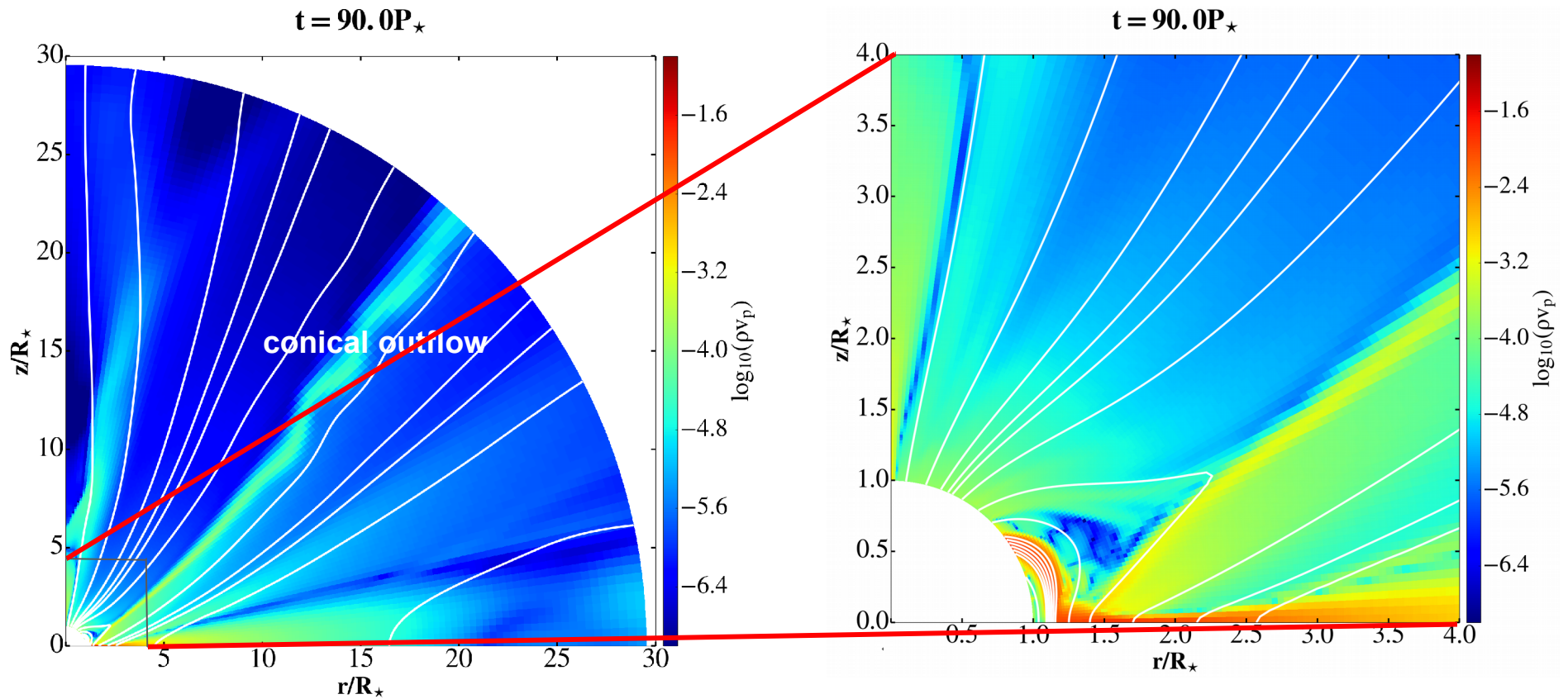


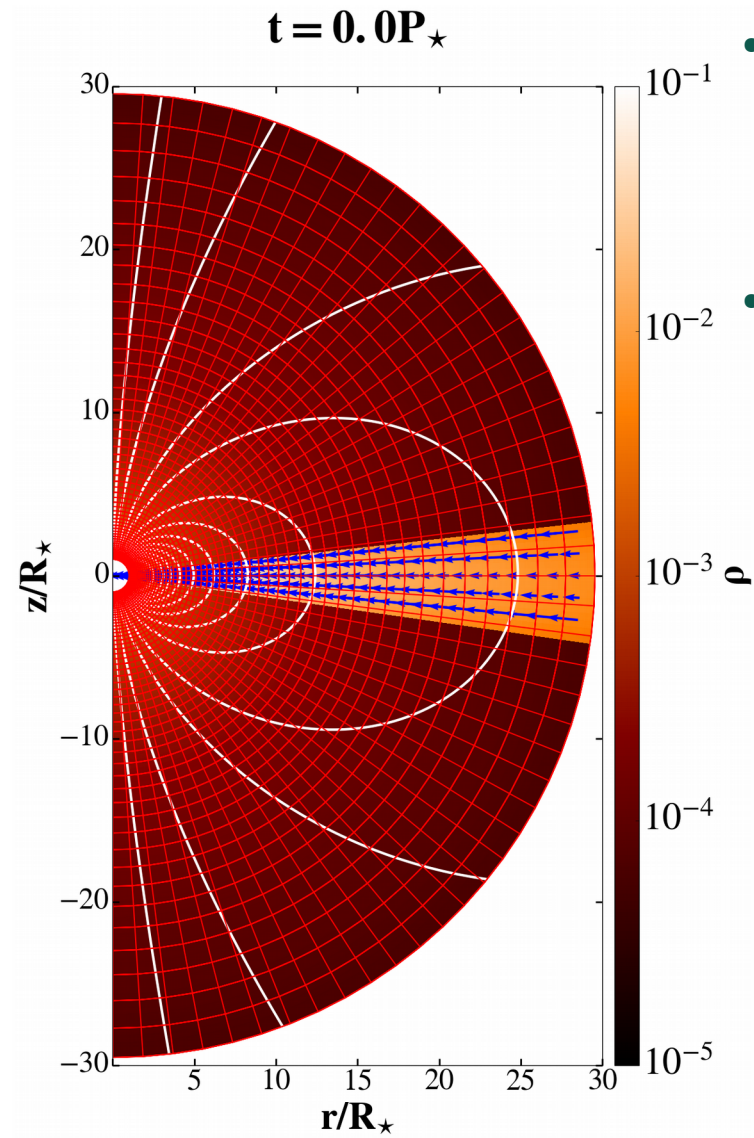
Fig. 5. Average angular momentum flux in the outflow that forms for $\alpha_m = 0.1$. It is computed at $R = 12R_\star$ for different stellar rotation rates. Normalization is the same as in Fig. 4. Fluxes in $B_\star = 0.25$ (dotted), 0.5 (dash-dotted), 0.75 (dashed) and 1 kG (solid) are shown.

Magnetospheric launching of conical outflow



Color shows **momentum** in the MHD simulation in the young stellar object with conical outflow, lines are the magnetic field lines. Right panel is a zoom into the left panel close to the star, to show the closest vicinity of the star.

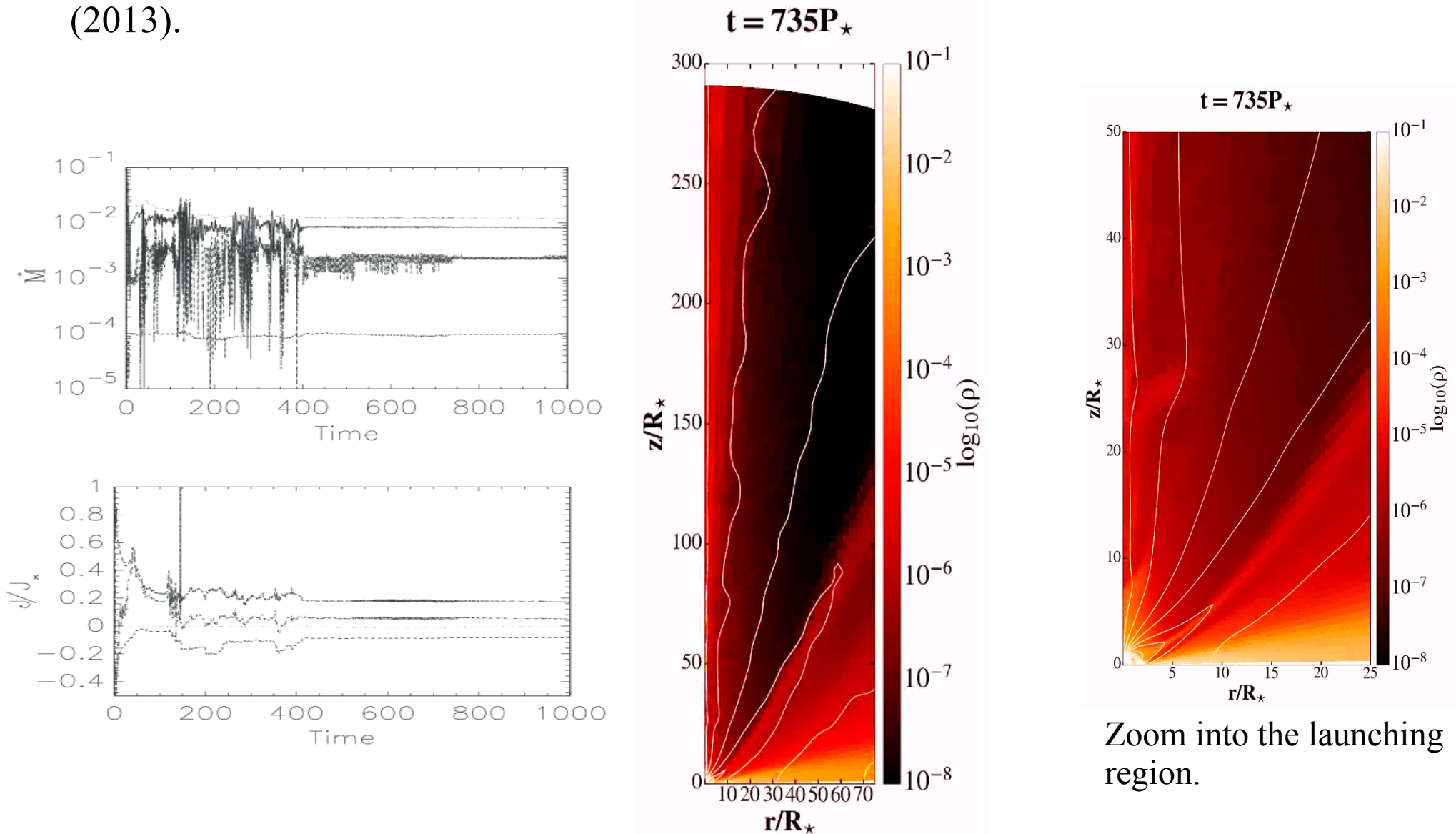
Setup in a complete co-latitudinal plane



- I did also $R \times \vartheta = [217 \times 200]$ grid cells in a complete co-latitudinal plane $\vartheta = [0, \pi]$, here is shown the case with dipole stellar field.
- Now there is no meridional plane boundary condition, so that the flow is not constrained at that plane.

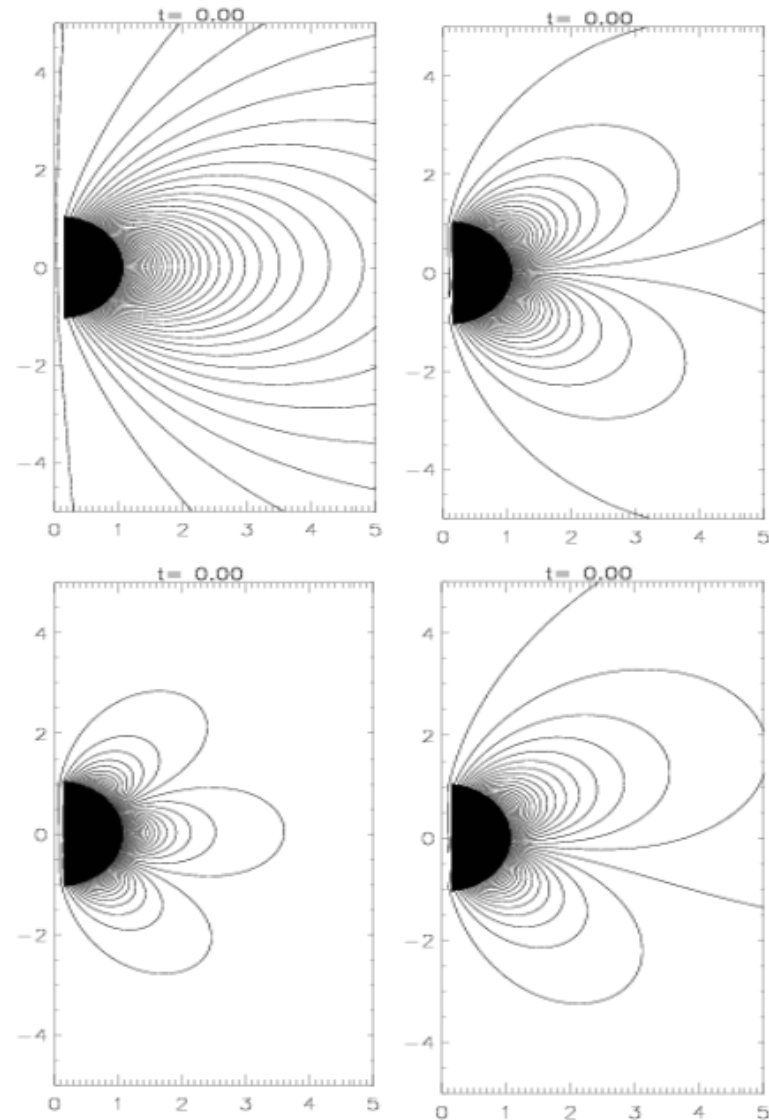
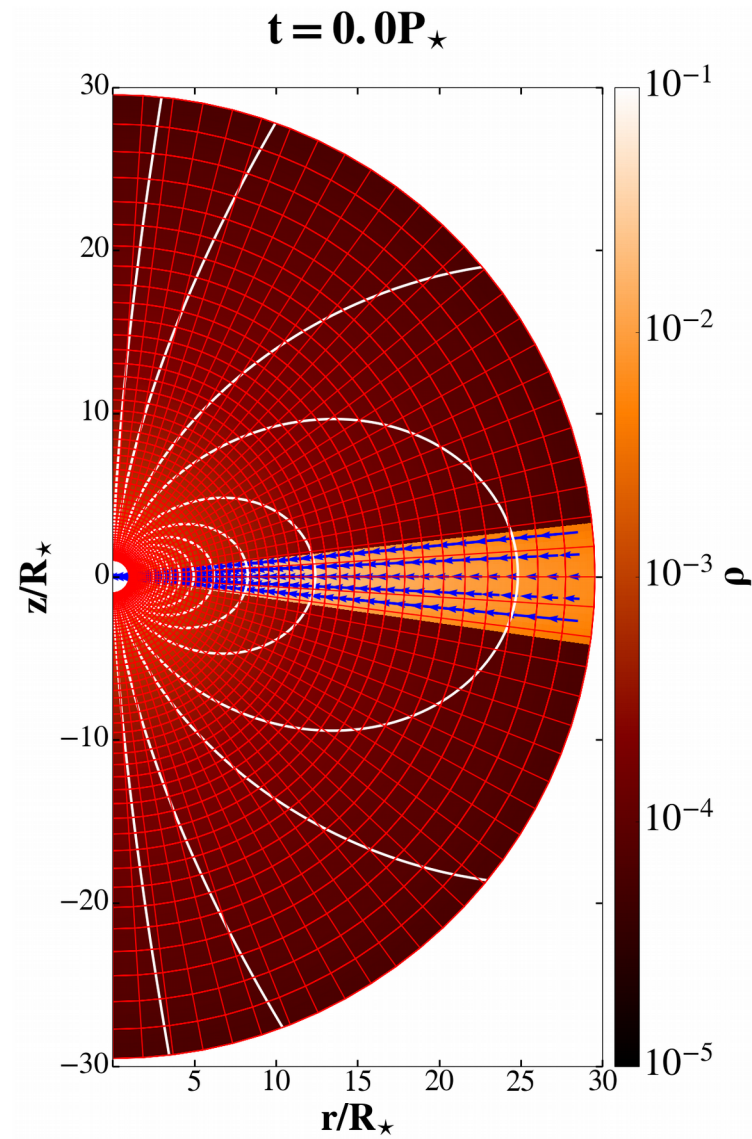
The axial outflow (“jet”) launching

- In a part of the parameter space, there is a continuous launching of an axial outflow from the star-disk magnetosphere. In my simulations, it showed that one has to wait until few hundreds of rotations of the star.
- The axial and the conical outflow are launched after the relaxation from the initial conditions. They are similar to the results in Romanova et al. (2009) and Zanni & Ferreira (2013).



Setup with different geometries of initial magnetic field

- I did also setups with different initial geometries of magnetic field than dipole: a quadrupole, octupole and mixed multipole.



Results with different geometries of initial magnetic field

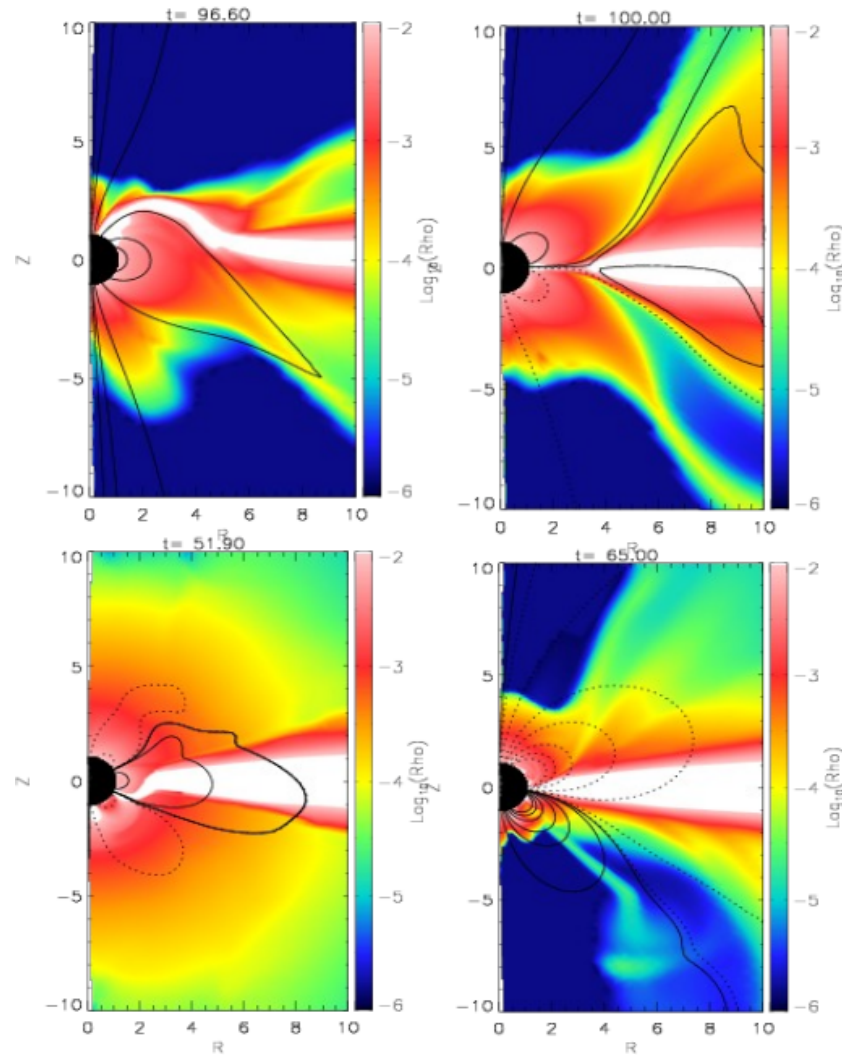


Figure 2: Stable solutions in our SDI simulations, for the different geometries of the magnetic field, with time t given in stellar rotations. The matter density distribution is shown in logarithmic color grading, with the poloidal magnetic field shown in solid and dashed lines, depending on the direction of the field. We sampled the magnetic field lines to leave the density distribution close to the star visible. The results for dipole, quadrupole, octupole and mixed multipole cases are shown from left to right, top to bottom, respectively.

Setup of magnetic runs-changes in pluto.ini and definitions.h

```
I pluto.ini (Modified)(ini) Row 27 Col 19 1:29 Ctrl-K H for help
Solver      hll
[Boundary]
X1-beg      userdef
X1-end      userdef
X2-beg      axisymmetric
X2-end      eqtsymmetric
X3-beg      outflow
X3-end      outflow
[Static Grid Output]
uservar     3 nu num Te
dbl         6.279 -1 single_file
flt         -1.0 -1 single_file
vtk         6.279 -1 single_file
tab         -1.0 -1
dbl.h5      -6.279 -1
ppm         -1.0 -1
png         -1.0 -1
log         100
analysis    -1.0 -1
[Chombo HDF5 output]
Checkpoint_interval -1
Plot_interval  1.0
[Parameters]
ALPHAM      0.4
MU          0.07
```

```
I A definitions.h (c) Row 1 Col 1 1:30 Ctrl-K H for help
#define PHYSICS MHD
#define DIMENSIONS 2
#define COMPONENTS 3
#define GEOMETRY SPHERICAL
#define BODY_FORCE VECTOR
#define FORCED_TURB NO
#define COOLING NO
#define RECONSTRUCTION LINEAR
#define TIME_STEPPING RK2
#define DIMENSIONAL_SPLITTING NO
#define NTRACER 1
#define USER_DEF_PARAMETERS 9

/* -- physics dependent declarations -- */
#define EOS IDEAL
#define ENTROPY_SWITCH NO
#define DIVB_CONTROL CONSTRAINED_TRANSPORT
#define BACKGROUND_FIELD YES
#define AMBIPOLAR_DIFFUSION NO
#define RESISTIVITY NO
#define HALL_MHD NO
#define THERMAL_CONDUCTION NO
#define VISCOSITY EXPLICIT
#define ROTATING_FRAME NO

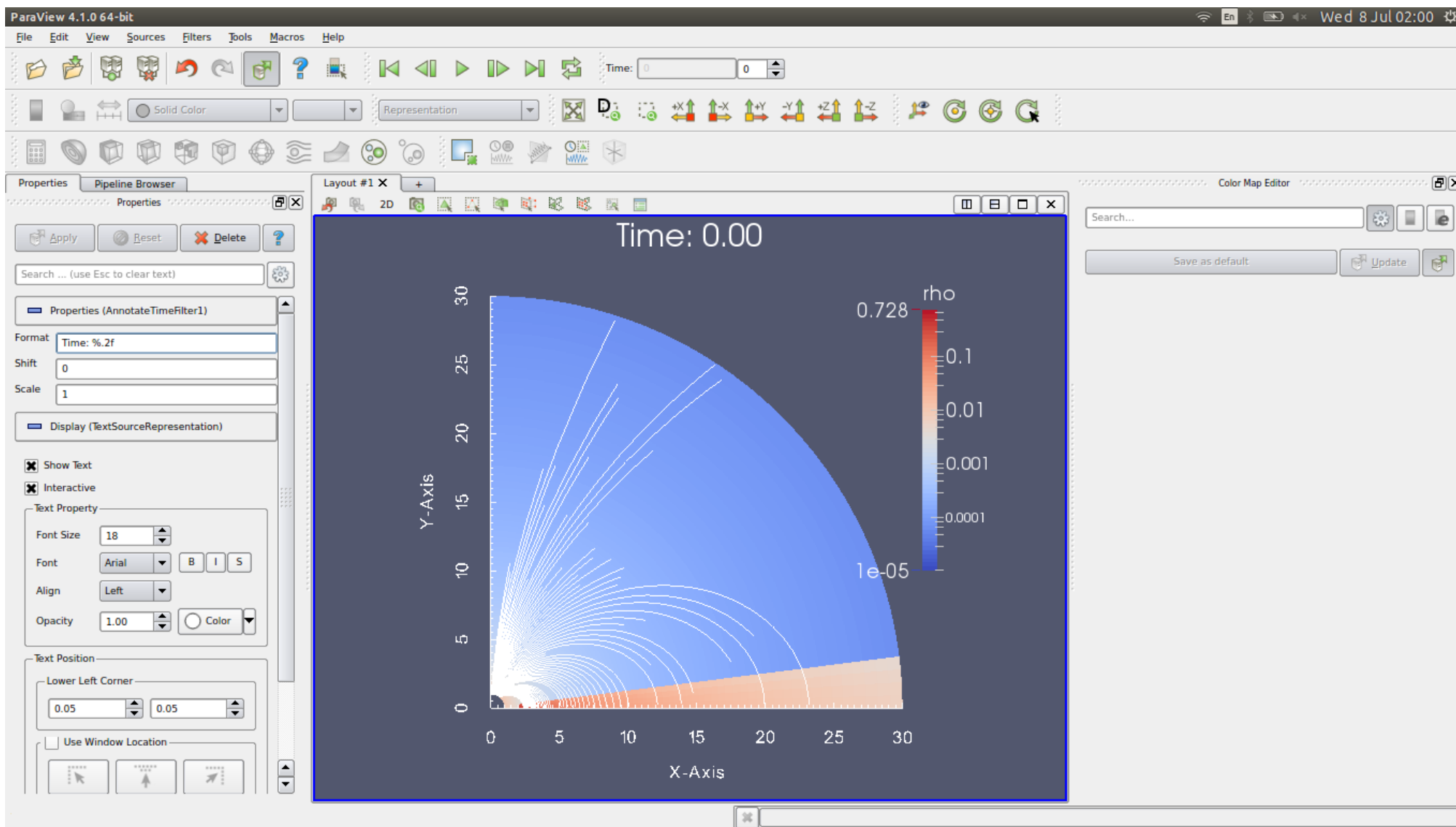
/* -- user-defined parameters (labels) -- */
#define ALPHAM 0
#define MU 1
#define TEMPF 2
#define RHOC 3
#define RD 4
#define EPS 5
```

Setup of magnetic dipole and multipoles in init.c

```
I A init.c (c) void UserDefBoundary (const Data *d, RBox *box, int side, Row 146 Col 1 4:22 Ctrl-K H for help)
#if PHYSICS == MHD
/* ***** */
void BackgroundField (double x1, double x2, double x3, double *B0)
/*
 *
 * PURPOSE
 *
 * Define the component of a static, curl-free background
 * magnetic field.
 *
 *
 * ARGUMENTS
 *
 * x1, x2, x3 (IN) coordinates
 *
 * B0 (OUT) vector component of the background field.
 *
 *
 * ***** */
{
/* dipole */
B0[0] = 2.*g_inputParam[MU]*cos(x2)/(x1*x1*x1);
B0[1] = g_inputParam[MU]*sin(x2)/(x1*x1*x1);
B0[2] = 0.0;
}
#endif
```

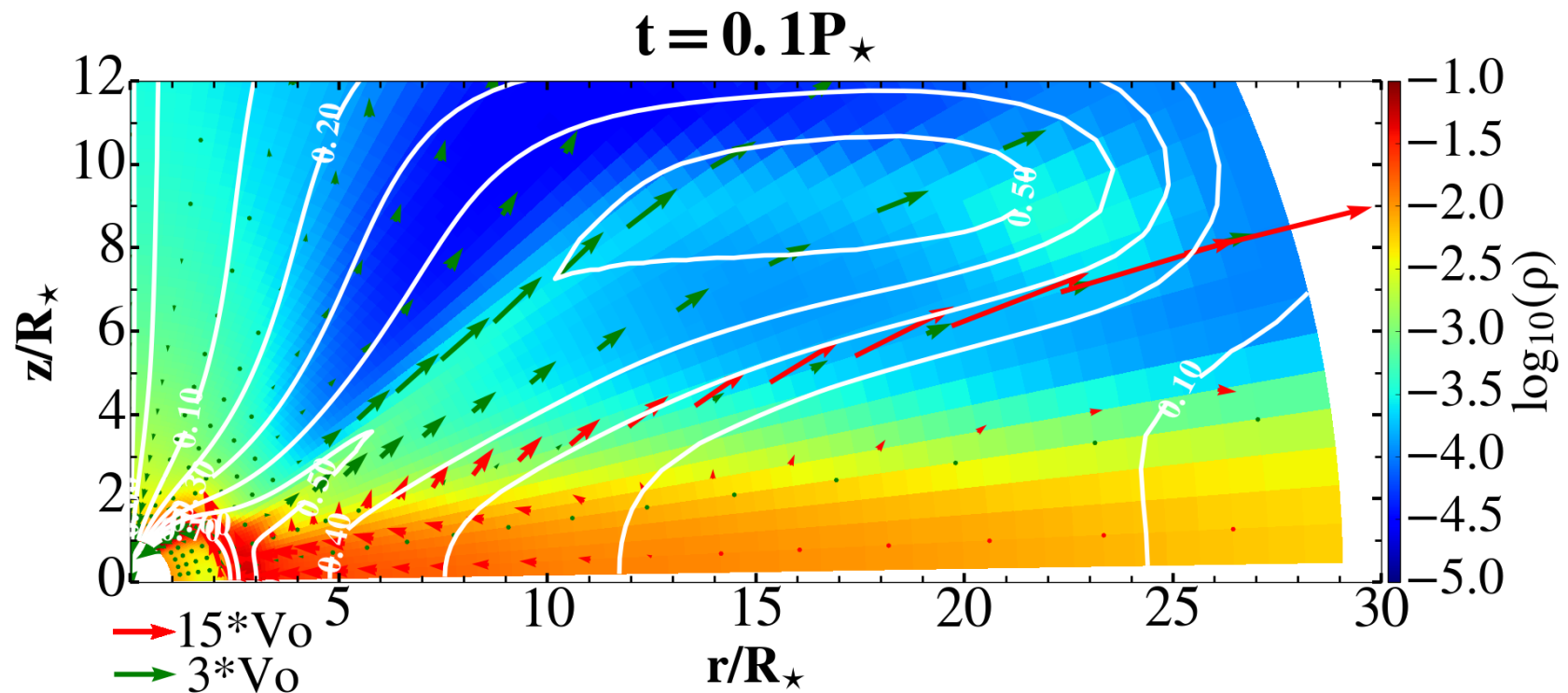
```
{
/* dipole */
B0[0] = 2.*g_inputParam[MU]*cos(x2)/(x1*x1*x1);
B0[1] = g_inputParam[MU]*sin(x2)/(x1*x1*x1);
B0[2] = 0.0;
/* */
/* quadrupole
B0[0] = 3.0/2.0*g_inputParam[MU]*(3.0*cos(x2)*cos(x2)-1.0)/(x1*x1*x1*x1);
B0[1] = 3.0*g_inputParam[MU]*cos(x2)*sin(x2)/(x1*x1*x1*x1);
B0[2] = 0.0;
*/
/* octupole
B0[0] = 2.0*g_inputParam[MU]*(5.0*cos(x2)*cos(x2)*cos(x2)-3.0*cos(x2))/(x1*x1*x1*x1*x1);
B0[1] = 0.5*g_inputParam[MU]*(15.0*cos(x2)*cos(x2)*sin(x2)-3.0*sin(x2))/(x1*x1*x1*x1*x1);
B0[2] = 0.0;
*/
}
```

Visualization of Bp lines with Paraview



Python script for visualization

- mc_veloclect.py , python ver.3.5 and newer.
- Density in logarithmic color grading.
- Vectors of velocity. Remember to show normalization, I normalize to Keplerian velocity at the stellar equator ($R=1$).
- Magnetic field lines-plots with A_ϕ isocontours-they are parallel to B_p field lines. Magnetic field lines could be plotted also with B_p streamlines, but I did not do it here.



Summary of the Part IV

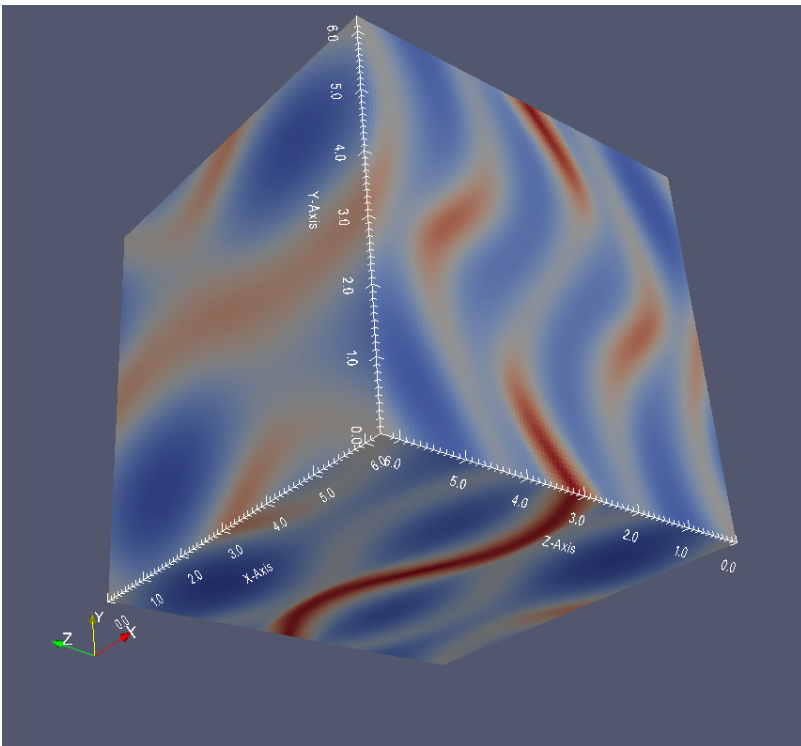
- I introduced star-disk magnetospheric interaction simulations.
- We learned setup of axisymmetric 2D setup with dipole magnetic field.
- Visualization of magnetic field lines with Paraview.
- Python script for visualization of mag. field lines and velocity vectors.

Outline, Part V

- Setup of 3D simulation: Orszag-Tang test in 3D.
-plotting of the 3D results in Paraview.
- Setup of HD disk in full 3D.
- How about MHD? 3D MHD disk with the dipole magnetic field.
- Restart and initialization from (modified) file.
- Examples of the code uses:
 - comparison with observations: light curve
 - post-processing (PLUTO + DUSTER): influence of the radiative pressure on the motion of dust grain particles
- Concluding remarks

Setup of 3D simulation, Orszag-Tang test in 3D

- Simple, just perform a setup.py in terminal (made a pls alias?) to set a 3D run, and add z-direction in pluto.ini:
- Visualisation with Paraview in 3D:



```
miki@petri: ~/Pluto44/OTang3D
I pluto.ini (ini) Row 20 Col 1
[Grid]
X1-grid 1 0.0 100 u 6.28318530717959
X2-grid 1 0.0 100 u 6.28318530717959
X3-grid 1 0.0 100 u 6.28318530717959

[Chombo Refinement]
Levels 4
Ref_ratio 2 2 2 2 2
Regrid_interval 2 2 2 2
Refine_thresh 0.3
Tag_buffer_size 3
Block_factor 4
Max_grid_size 32
Fill_ratio 0.75

[Time]
CFL 0.8
CFL_max_var 1.1
tstop 3.1
** Joe's Own Editor v4.1 ** (utf-8) ** Copyright © 20
```

Setup of 3D simulation of accretion disk in HD case

- Should be simple, just define the whole Kluźniak-Kita solution, which is HD and axisymmetric, so we just set the same in each azimuthal plane.
- We need $\theta=[0,\pi]$ version of the setup now, so boundaries will be different.

```
miki@petri: ~/Pluto44/Pluto3dHDdisk
I pluto.ini (Modified)(ini) Row :
[[Grid]]
X1-grid 1 1.0 109 l+ 30.
X2-grid 1 0.0 50 u 3.14159
X3-grid 1 0.0 16 u 6.283185

[[Chombo Refinement]]
Levels 4
Ref_ratio 2 2 2 2 2
Regrid_interval 2 2 2 2
Refine_thresh 0.3
Tag_buffer_size 3
Block_factor 8
Max_grid_size 64
Fill_ratio 0.75

[[Time]]
```

Restart of simulation from a (modified) file

- Setup of 3D run: change to 3D in definitions.h (or through python setup) and change the pluto ini grid accordingly in pluto.ini file, change the boundary condition (usually “periodic”).
- Restart from a file is one of the options in PLUTO.
- One can start completely new simulation from a (modified) file. Previous result, with eventual modifications, is now used as an initial condition for new simulation.
- You can create a 3D initial condition from the result of 2D simulation.

```
I A init.c (c) }
```

```
void InitDomain (Data *d, Grid *grid)
/*!
 * Assign initial condition by looping over the computational domain.
 * Called after the usual Init() function to assign initial conditions
 * on primitive variables.
 * Value assigned here will overwrite those prescribed during Init().
 *
 *
 */
*****
{
int i,j,k,id;
double *x1 = grid->x[IDIR];
double *x2 = grid->x[JDIR];
double *x3 = grid->x[KDIR];
double r;
double coeff, eps2, pc, rcyl;
double br,bth;
double lambda;
double xhi2, Rco;

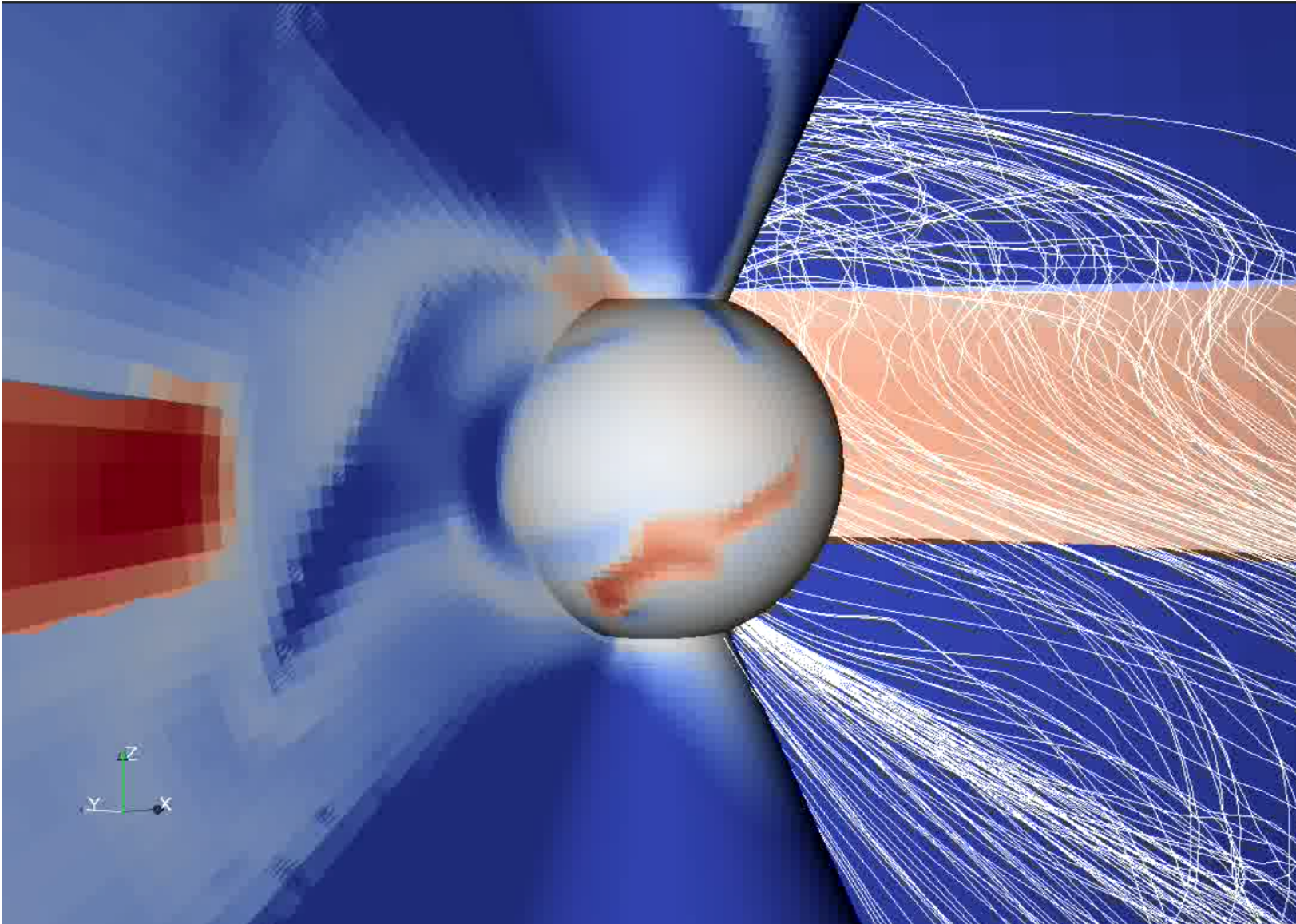
rcyl=x1[i]*sin(x2[j]);
eps2=g_inputParam[EPS]*g_inputParam[EPS];
coeff=2./5./eps2*(1./x1[i]-(1.-5./2.*eps2)/rcyl);
lambda=11./5./(1.+64./25.*g_inputParam[ALPHAV]*g_inputParam[ALPHAV]);

id = InputDataOpen ("./Adata.0045.dbl", "grid.out", " ", 0, CENTER);
TOT_LOOP(k,j,i){
d->Vc[RHO][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[PRS][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[VX1][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[VX2][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[VX3][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[BX1][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
}
```

```
I A init.c (c) void InitDomain (Data *d, Grid *grid)
d->Vc[RHO][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[PRS][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[VX1][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[VX2][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[VX3][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[BX1][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[BX2][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[BX3][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
d->Vc[TRC][k][j][i] = InputDataInterpolate(id,x1[i],x2[j],x3[k]);
}
InputDataClose(id);
/**/
TOT_LOOP(k,j,i){
// pc=2./5.*g_inputParam[RHOC]*pow(x1[i],-3./2.);
if (d->Vc[RHO][k][j][i] >= pc){
// if (d->Vc[TRC][k][j][i] >= 0.5) {
}
else
{
d->Vc[RHO][k][j][i] = g_inputParam[RHOC]*pow(x1[i],-3./2.);
d->Vc[PRS][k][j][i] = 2./5.*g_inputParam[RHOC]*pow(x1[i],-5./2.);
d->Vc[VX1][k][j][i] = 0.0;
if (x2[j] < CONST_PI/2./20.) d->Vc[VX1][k][j][i] = 0.001;
d->Vc[VX2][k][j][i] = 0.0;
d->Vc[VX3][k][j][i] = 0.0;
d->Vc[TRC][k][j][i] = 0.0; // Track the corona
}
} /**/
}
/* ***** */
void Analysis (const Data *d, Grid *grid)
/*
```

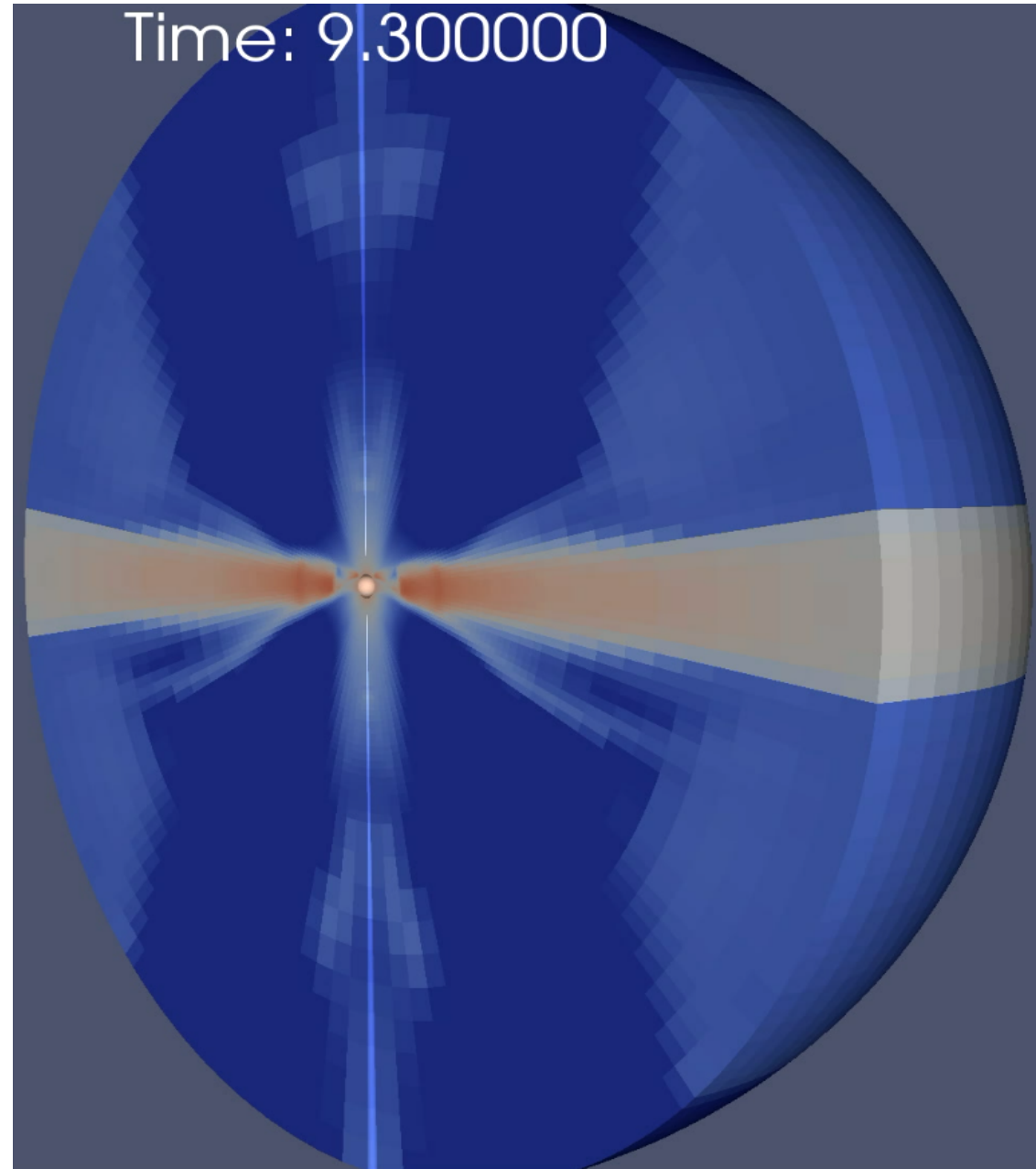
3D simulations, tilted magnetic field case

- The magnetic field in this case is not aligned with the rotation axis.



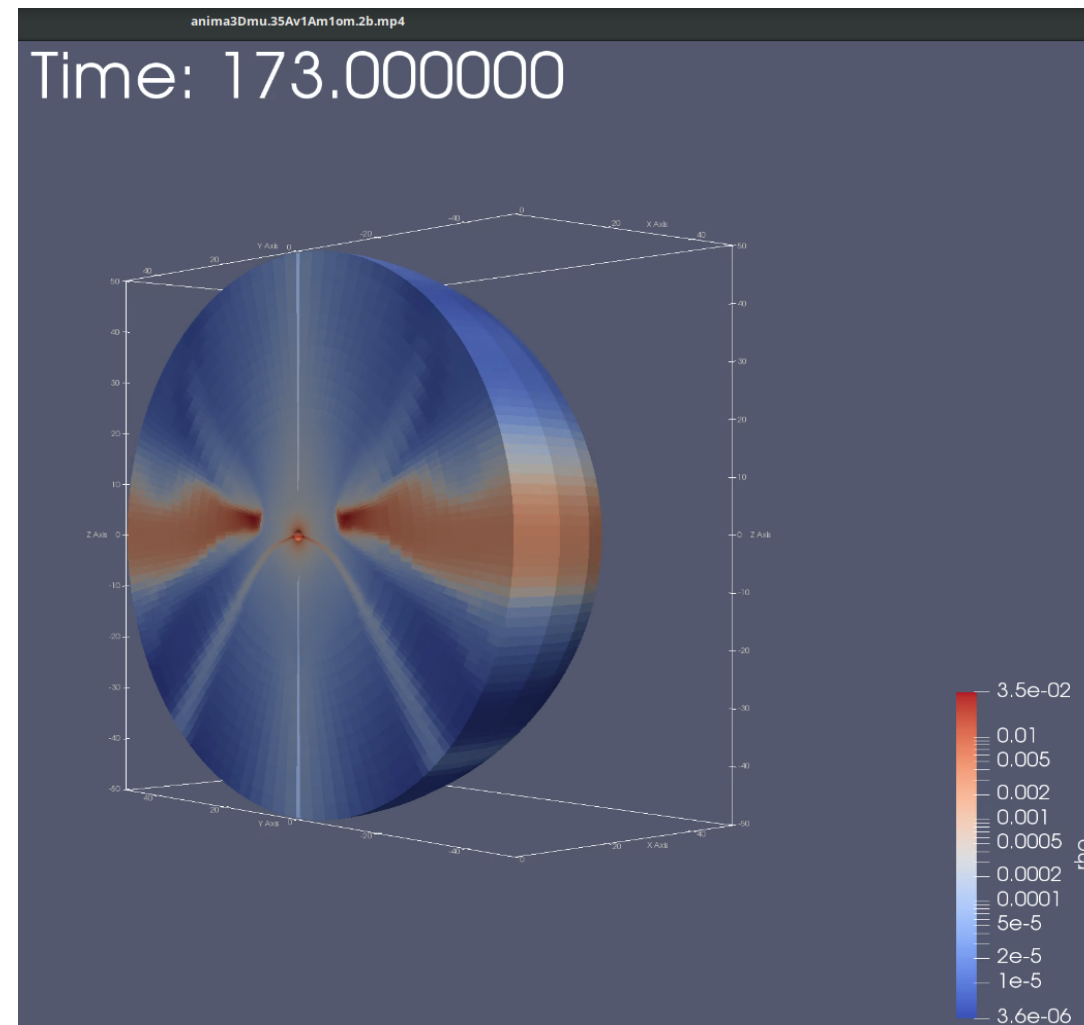
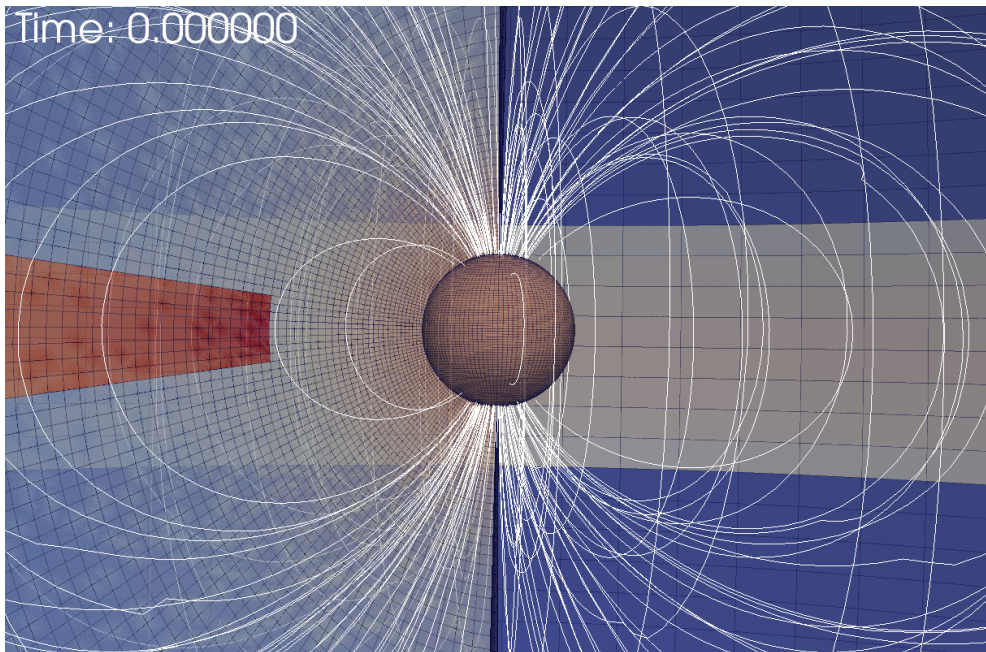
The asymmetric jet launching in 3D simulations

- In the full 3D simulations I obtain asymmetric jets.



3D simulations, case with magnetic field aligned with rotation

- The first step here is to perform the axisymmetric 3D simulation. I use the spherical grid. The magnetic field in this case is aligned with the rotation axis.
- Zoom into the vicinity of the star=inner boundary condition at $T=0$ and at $T=173$



Comparison with observations, “hiccups” in light curves

- A curious case with switching of the hemisphere to which the column is attached (Čemeljić & Siwak, 2020).

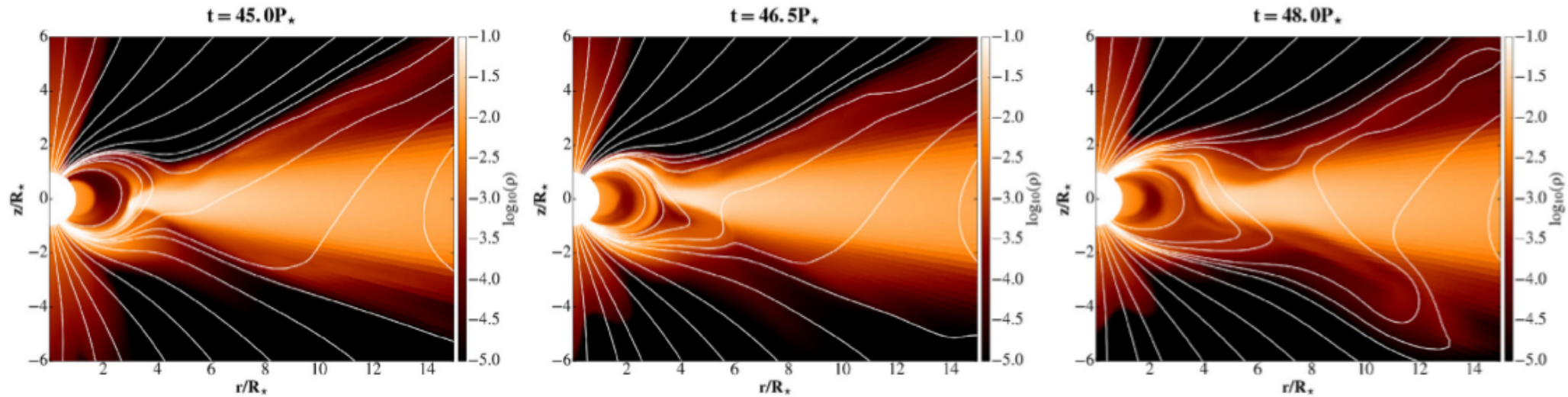


Figure 4. A sequence of snapshots from the results in the interval when switching of the accretion column from the Southern to the Northern hemisphere occurs.

Comparison with observations, “hiccups” in light curves

- Such switching could explain some of the cases of aperiodicity in the Young Stellar Objects.

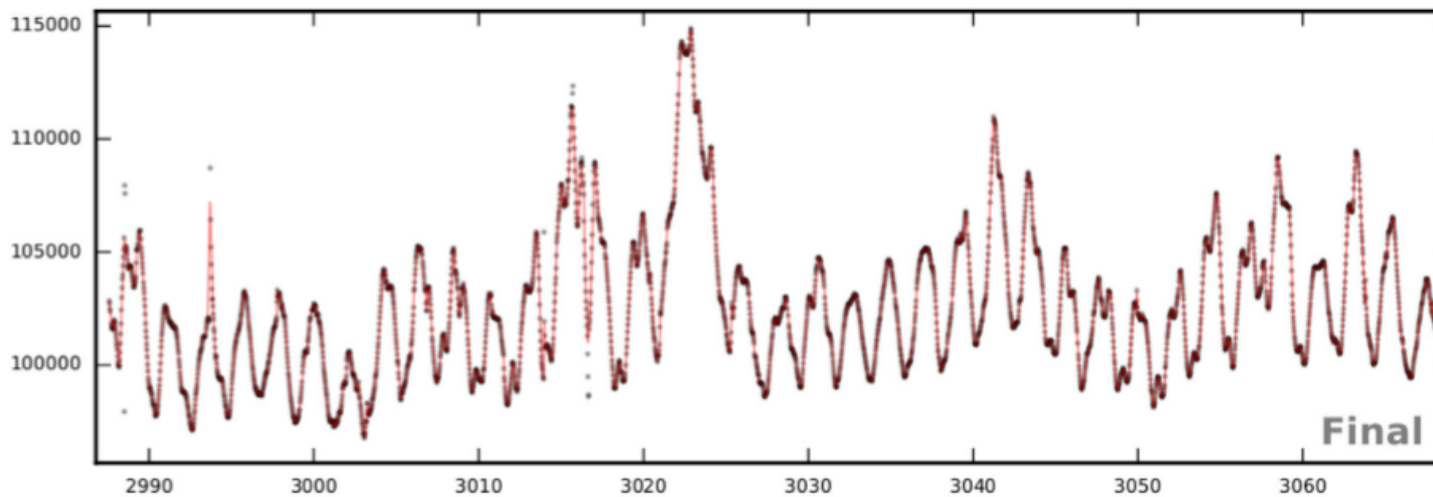
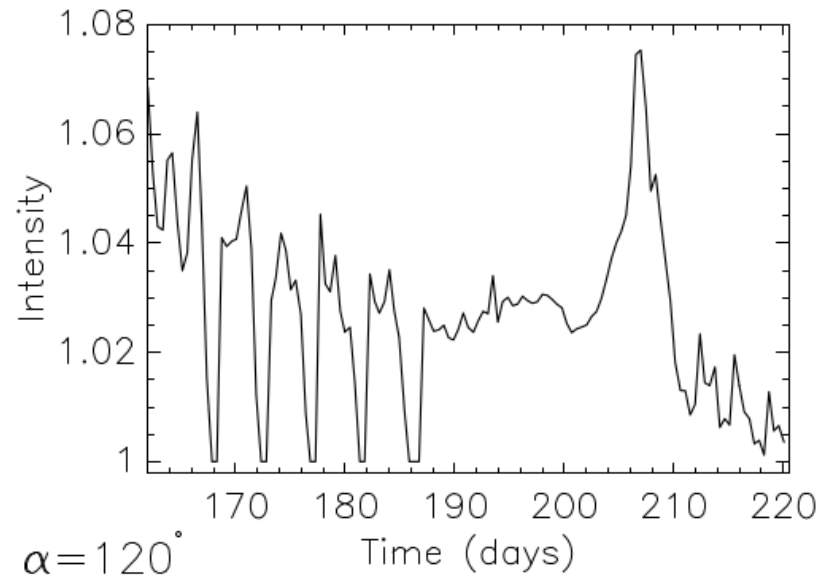
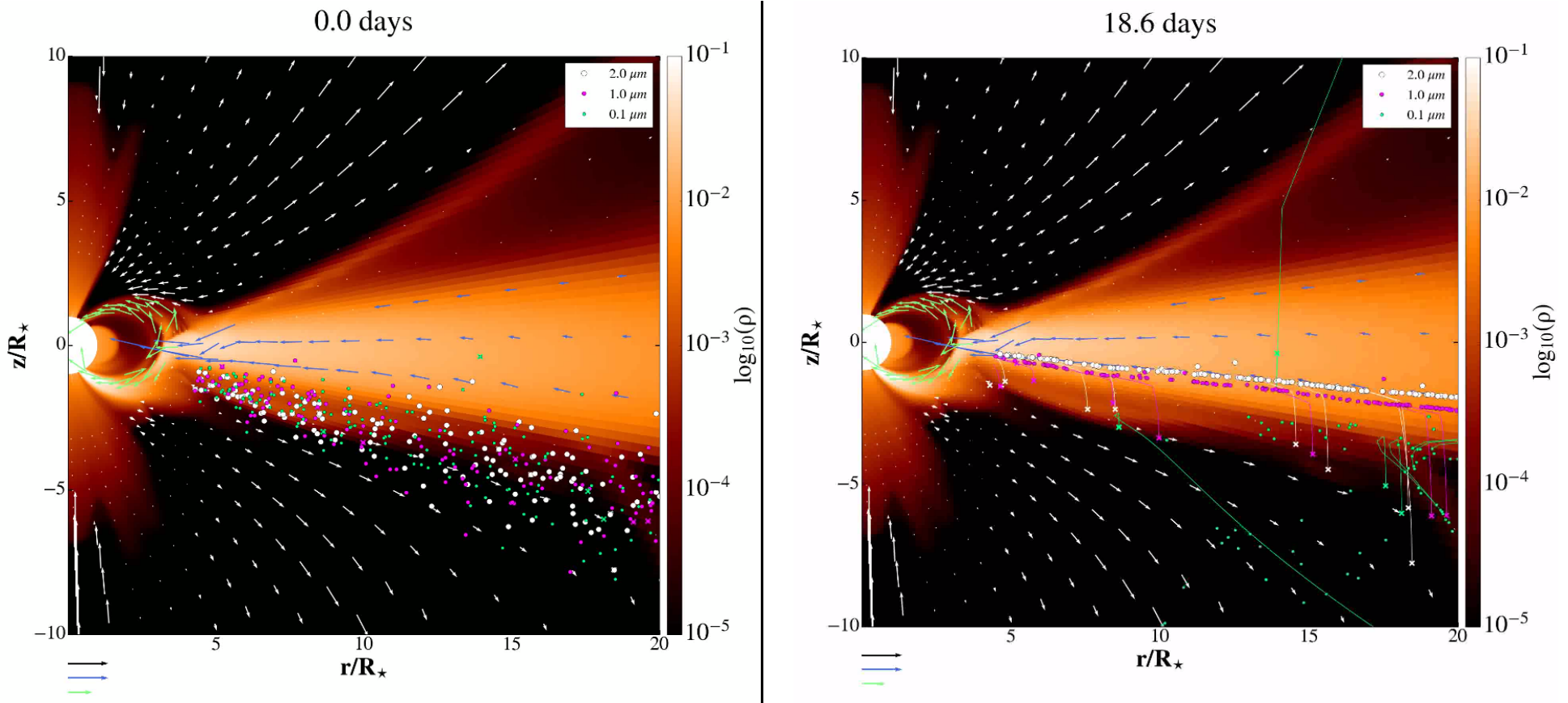


Figure 8. Top panel: intensity in a 3D model made from the longer time sequence than shown in Fig. 7. Shifts in the phase of the observed light (‘hiccups’) occur during the switch of the accretion column from the southern to northern stellar hemisphere between the 190 and 210 d. Bottom panel: light curve from the *Kepler* observation of V1000 Tau in which part of the contribution could occur because of the column switching. Time in the abscissa is annotated in Julian days.

“STARDUST”: Dusty disk in Young Stellar Objects

During a Summer Student Program, C. Turski wrote the Python script “DUSTER” for post-processing of the quasi-stationary results in my star-disk solutions. He added the dust particles and computed their movement in the disk-corona solution as a background. The results are used to model dust distribution in the disk.



$$\ddot{\vec{r}} = -G \frac{M_{\star}}{r^3} \vec{r} - \frac{\rho_{gas}}{\rho_{gr}} \frac{c_s}{a} (\dot{\vec{r}} - \vec{v}_{gas}) + \beta G \frac{M_{\star}}{r^2}$$

$$\beta = 0.4 \frac{L_{\star}}{L_{\odot}} \frac{M_{\odot}}{M_{\star}} \frac{3000 \frac{kg}{m^3}}{\rho_{gr}} \frac{\mu m}{a}, \quad R_{in} = 0.0344 \Psi \left(\frac{1500K}{T_{gr}} \right)^2 \sqrt{\frac{L_{tot}}{L_{\odot}}} [AU],$$

Summary & Concluding Remarks

-In 5 lectures, I presented the PLUTO code, in hands-on approach. I chose PLUTO because it is well used (=checked and updated) in the astrophysics community, is user-friendly (good documentation) and very well written (good tool for hands-on learning of C programming language).

- We started with general description and followed with a standard Orszag-Tang test. This brought us to the thin disk setup in HD. With addition of the stellar dipole magnetic field we obtained MHD setup for a star-disk magnetospheric interaction for a thin accretion disk in 2D axisymmetric simulations.
- Along the way, we learned different ways of visualization: Paraview, VisIt, Python
- I detailed the 3D setup of magnetic star-disk simulations, ab initio and from the already evolved setup, restarting/initializing simulations from a (modified) previous result file.
- I described the use of results in post-processing, to create a light curve, compute the dust settling or radiative pressure effects on the dust.

Thank you and Good Luck!

This was a schnell kurs in PLUTO. I hope it will be of use to you and your students. For me, PLUTO is 15 years in places like my island in Croatia and the cities, Athens, Warsaw, Taipei and Paris:

