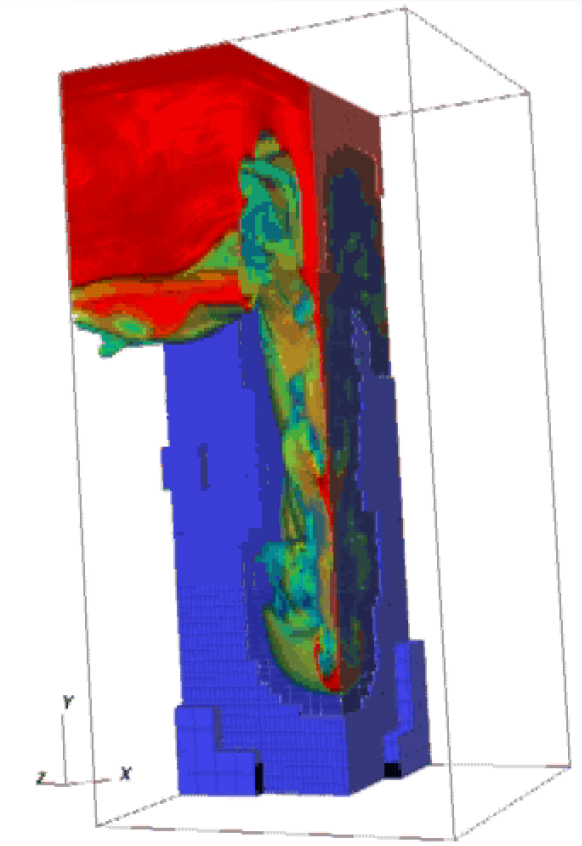
The PLUTO Code

- PLUTO[1,2] is a modular parallel code providing a *multi-physics* as well as a *multi-algorithm* framework for solving the equations of gas and plasma dynamics in astrophysics;



- *Target*: multidimensional *compressible* plasma with high Mach numbers:
  - Compressible Euler/Navier Stokes;
  - Newtonian (ideal/resistive) magnetohydrodynamics (MHD);
  - Special Relativistic hydro and MHD;
  - Heating/cooling processes, chemical network, …

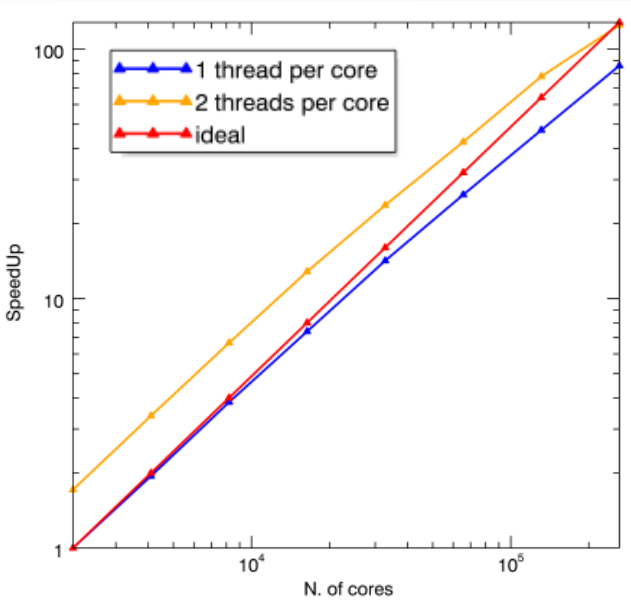
- Freely distributed at *http://plutocode.ph.unito.it* (v. 4.2)

[1]Mignone et al. ApJS (2007), 170, 228-242; [2]Mignone et al, ApJS (2012), 198, 7

Introducing the PLUTO Code

- PLUTO is written in C (~80,000 lines) and C++ (12,000 lines);

- Support multi-dimensional parallel (MPI) computations from single processor to a large number of cores (tested up to 262,144);



- Tested on several platforms (Linux/Mac OS/ SP6/Blue Gene Q,P, Cineca Tier-0 system, …);

- Computations may be performed on

  - Static grid : single fixed grid;
  - *Adaptive grid:* multiple refined, block-structured nested grids following and adapting to the solution

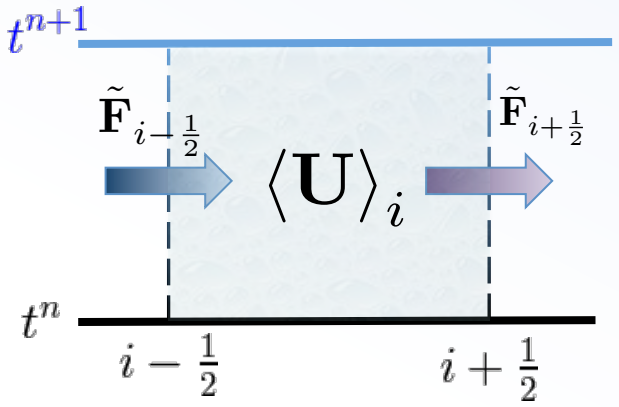
Underlying Phylosophy

- Suited for a system of conservation laws:

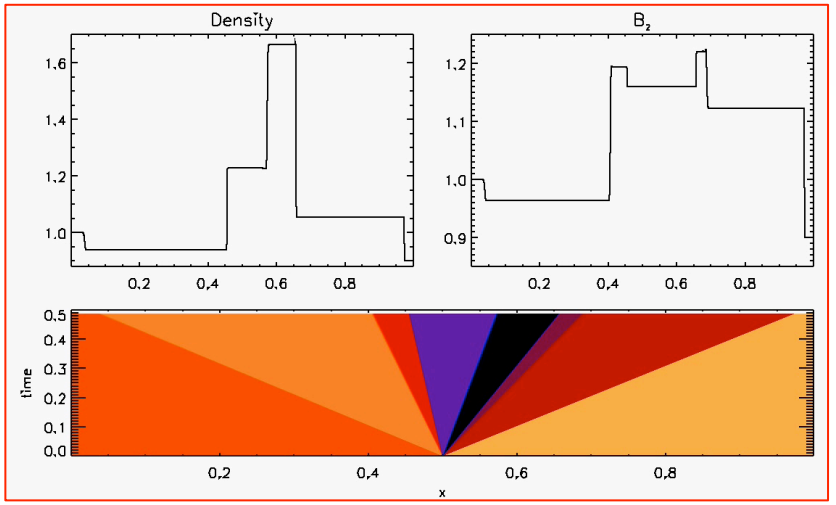
$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F} = \mathbf{S}$$

- Grid-based, finite-volume code

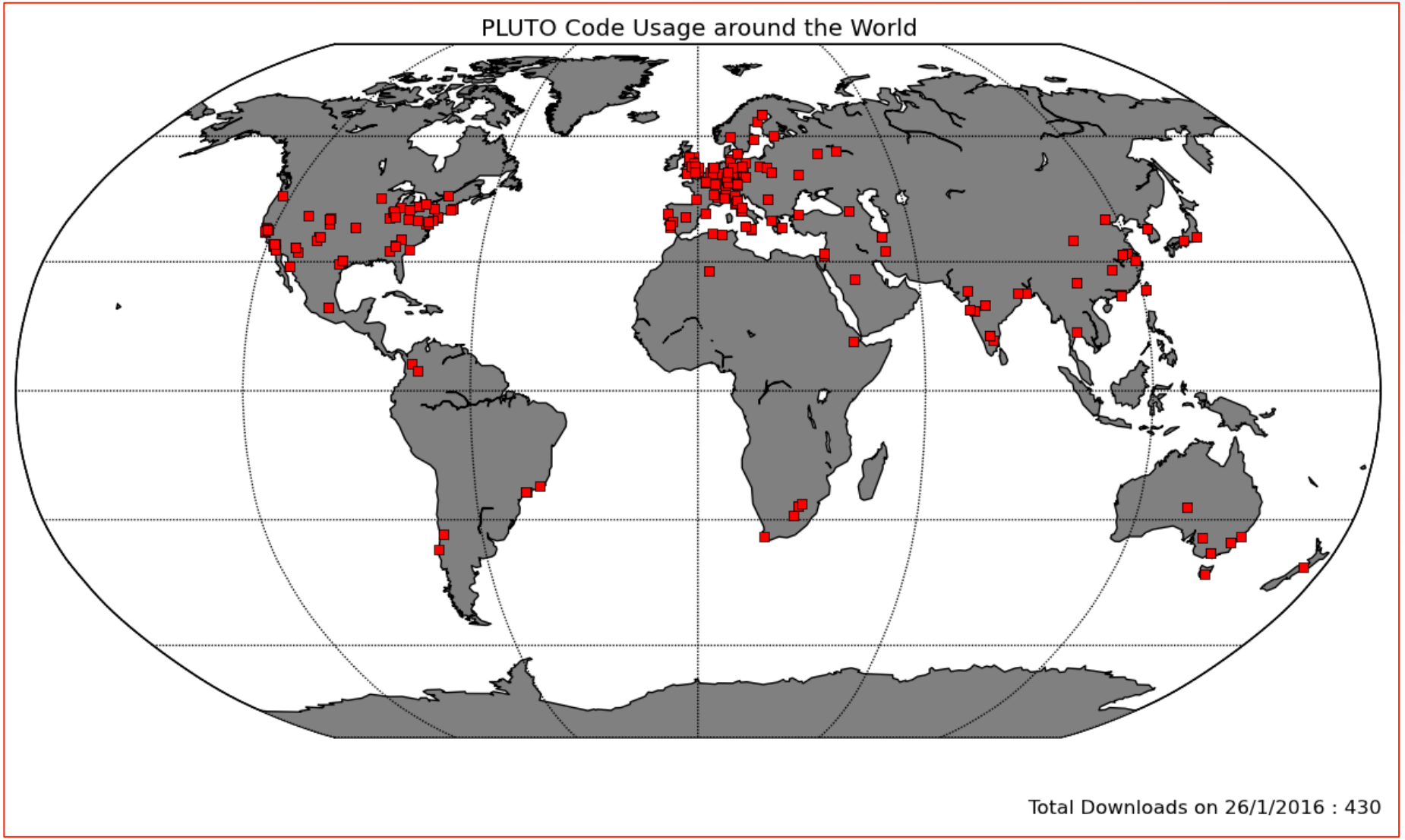
$$\langle U \rangle_i^{n+1} = \langle U \rangle_i^n - \frac{\Delta t}{\Delta x}\left(\tilde{F}_{i+\frac{1}{2}}^{n+\frac{1}{2}} - \tilde{F}_{i-\frac{1}{2}}^{n+\frac{1}{2}}\right)$$



- Flux computed by solving a Riemann problem at cell interfaces: the decay of initially discontinuous data →

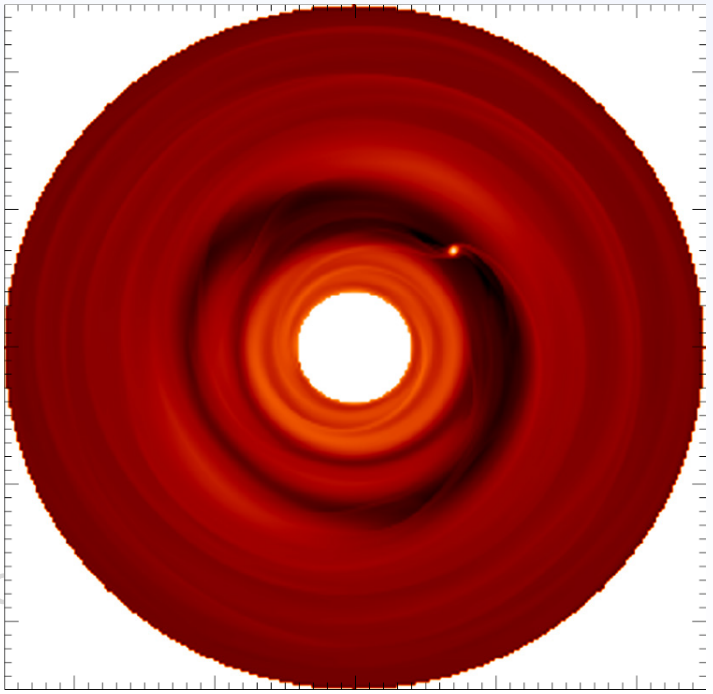


PLUTO Worldwide Distribution



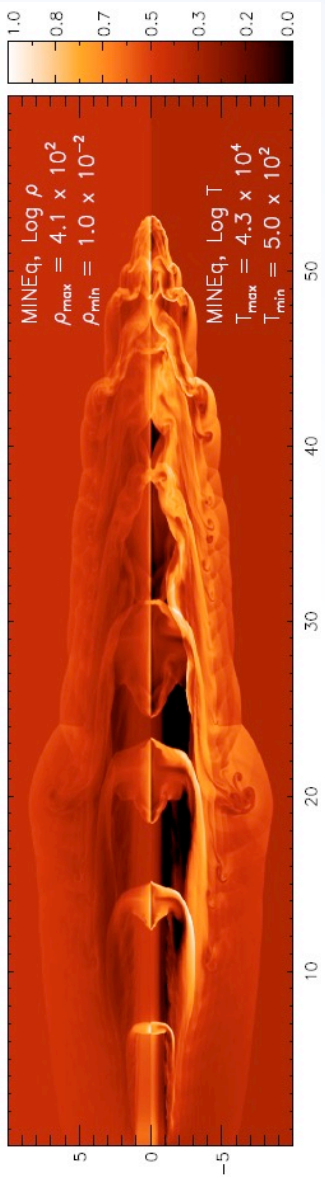
PLUTO Application Gallery

- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction
- Magneto-rotational instability (MRI) & ac
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc…



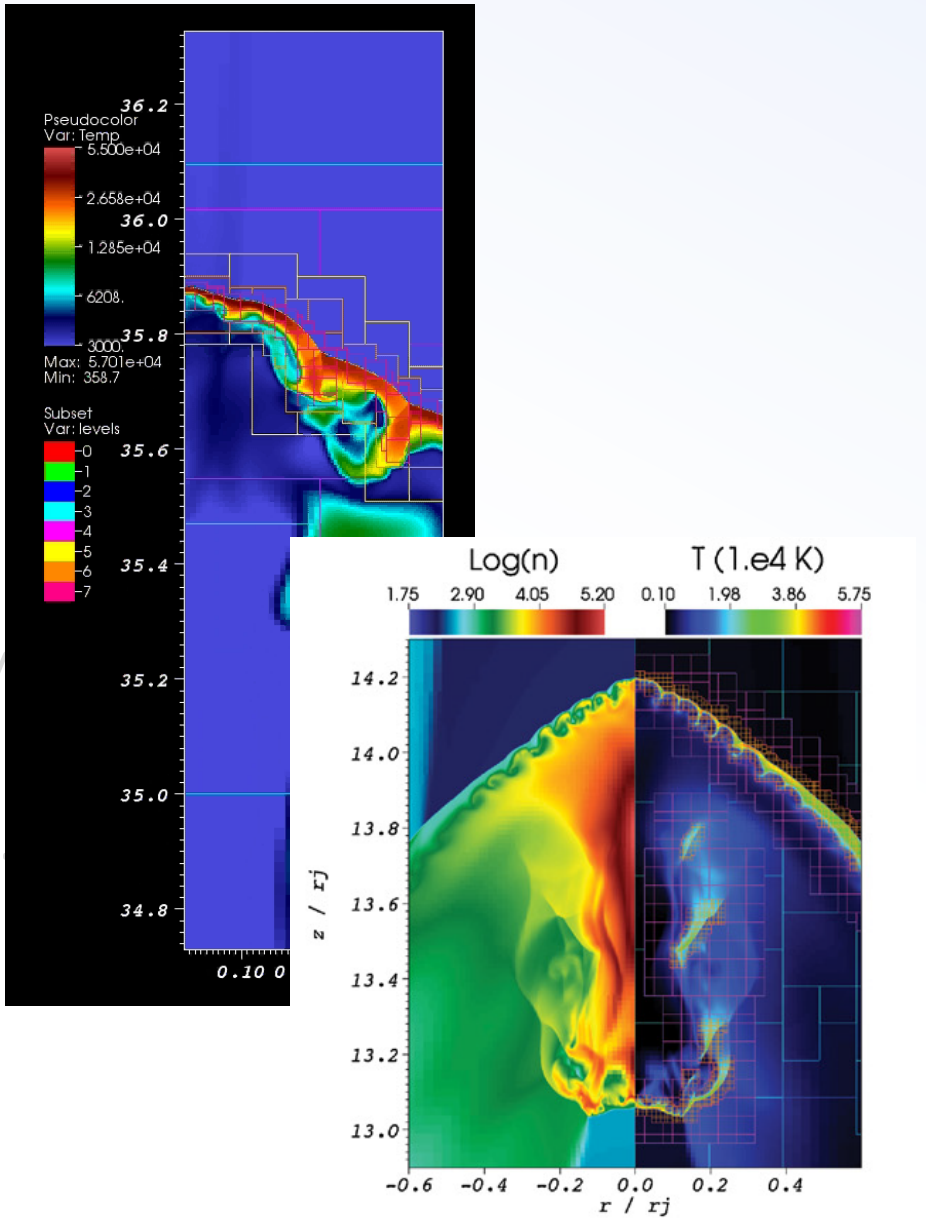
PLUTO Application Gallery

- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction
- Magneto-rotational instability (MRI) & acc
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc…



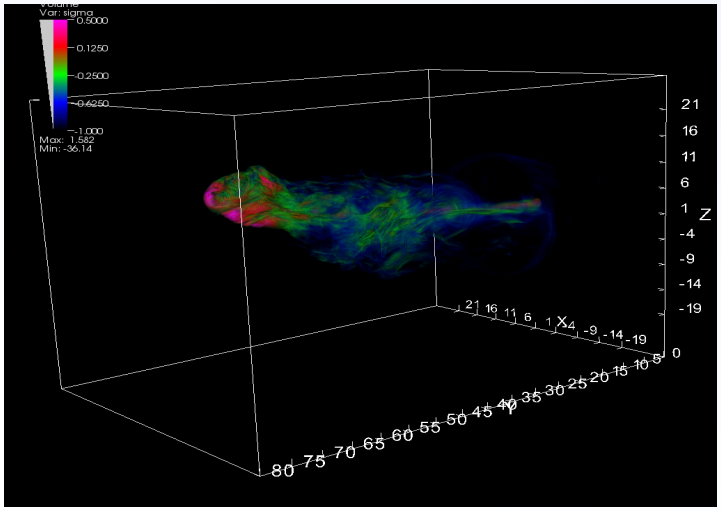
PLUTO Application Gallery

- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction
- Magneto-rotational instability (M
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc.

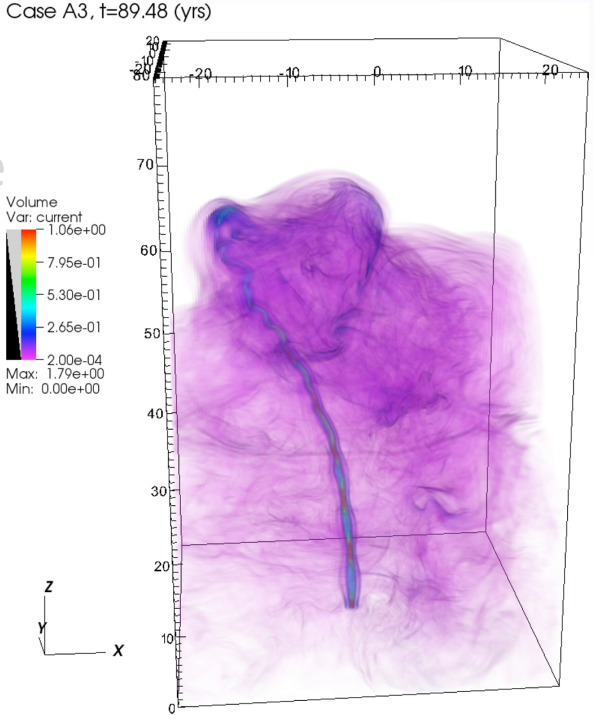


PLUTO Application Gallery

- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction
- Magneto-rotational instability (MRI) & accre
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc…

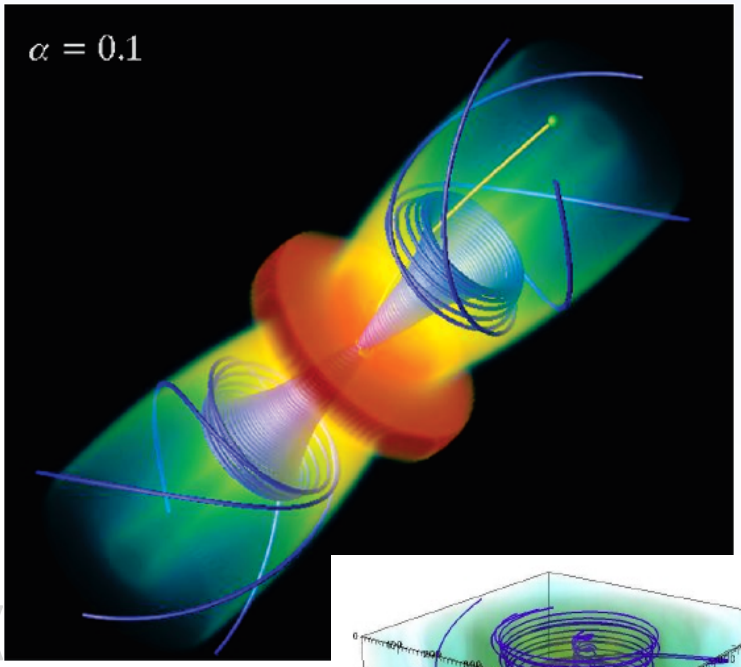


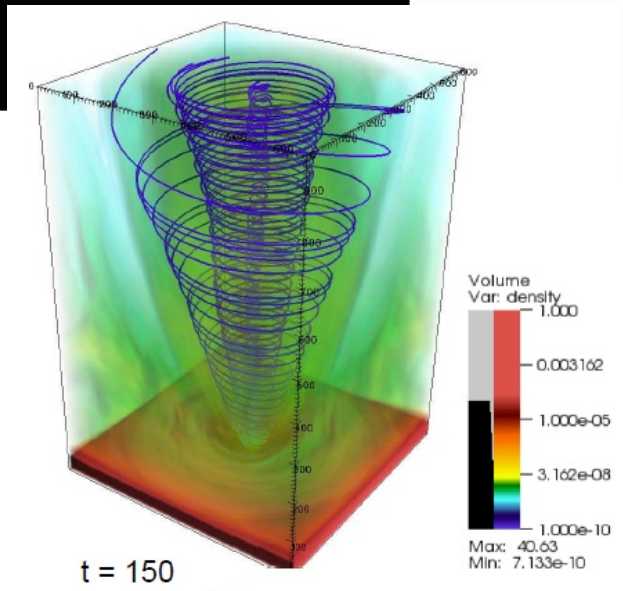
Case A3, t=89.48 (yrs)



PLUTO Application Gallery

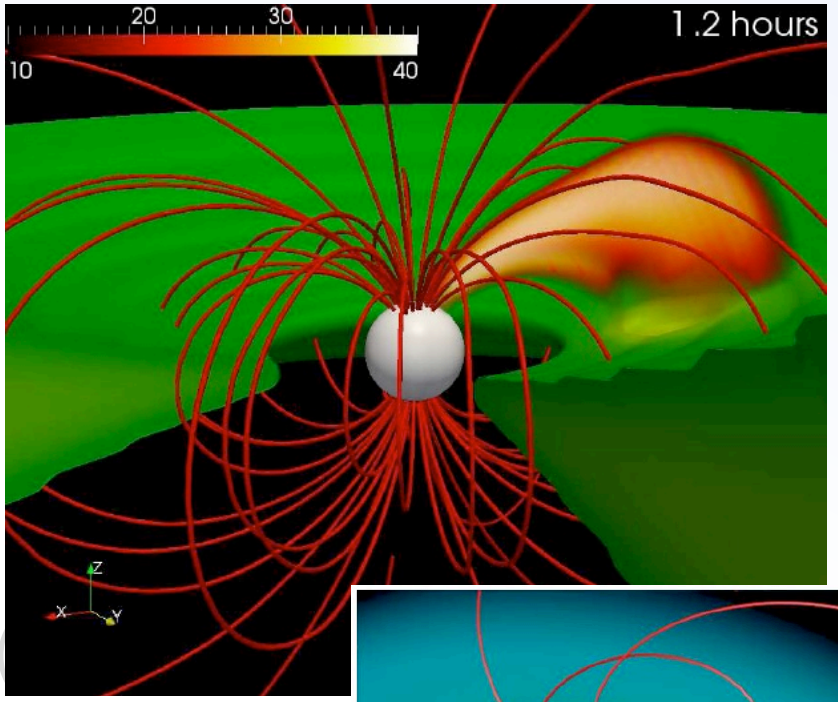
- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction
- Magneto-rotational instability (
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc…

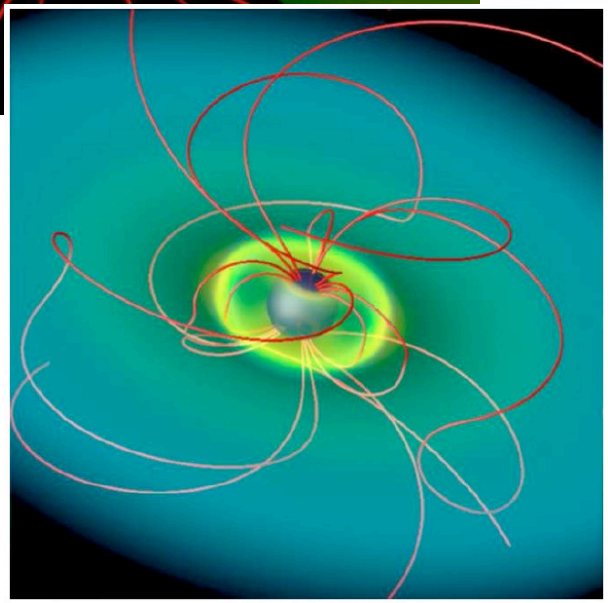




PLUTO Application Gallery

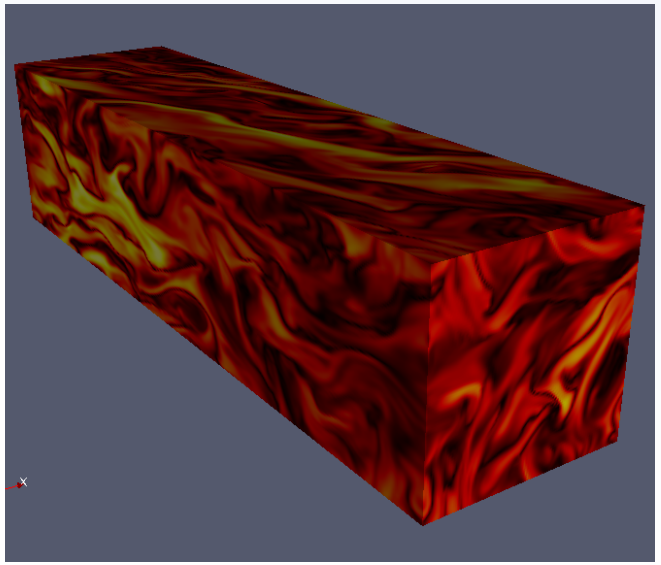
- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction
- Magneto-rotational instability (
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc…



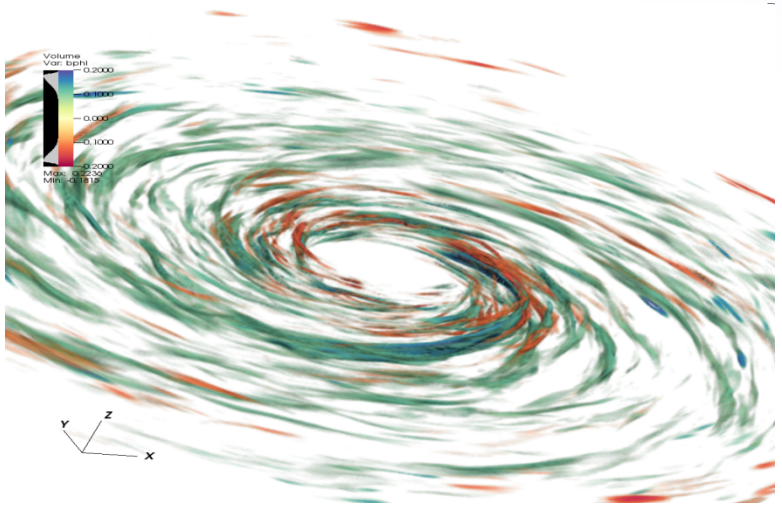


PLUTO Application Gallery

- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction

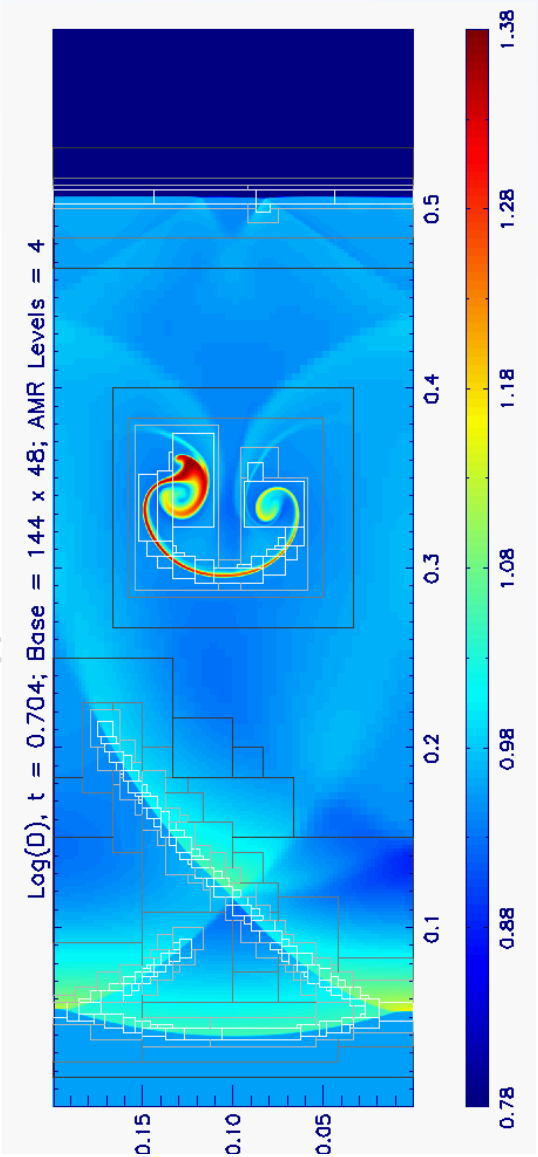


- Magneto-rotational instability (MRI) & accretion disks
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc…



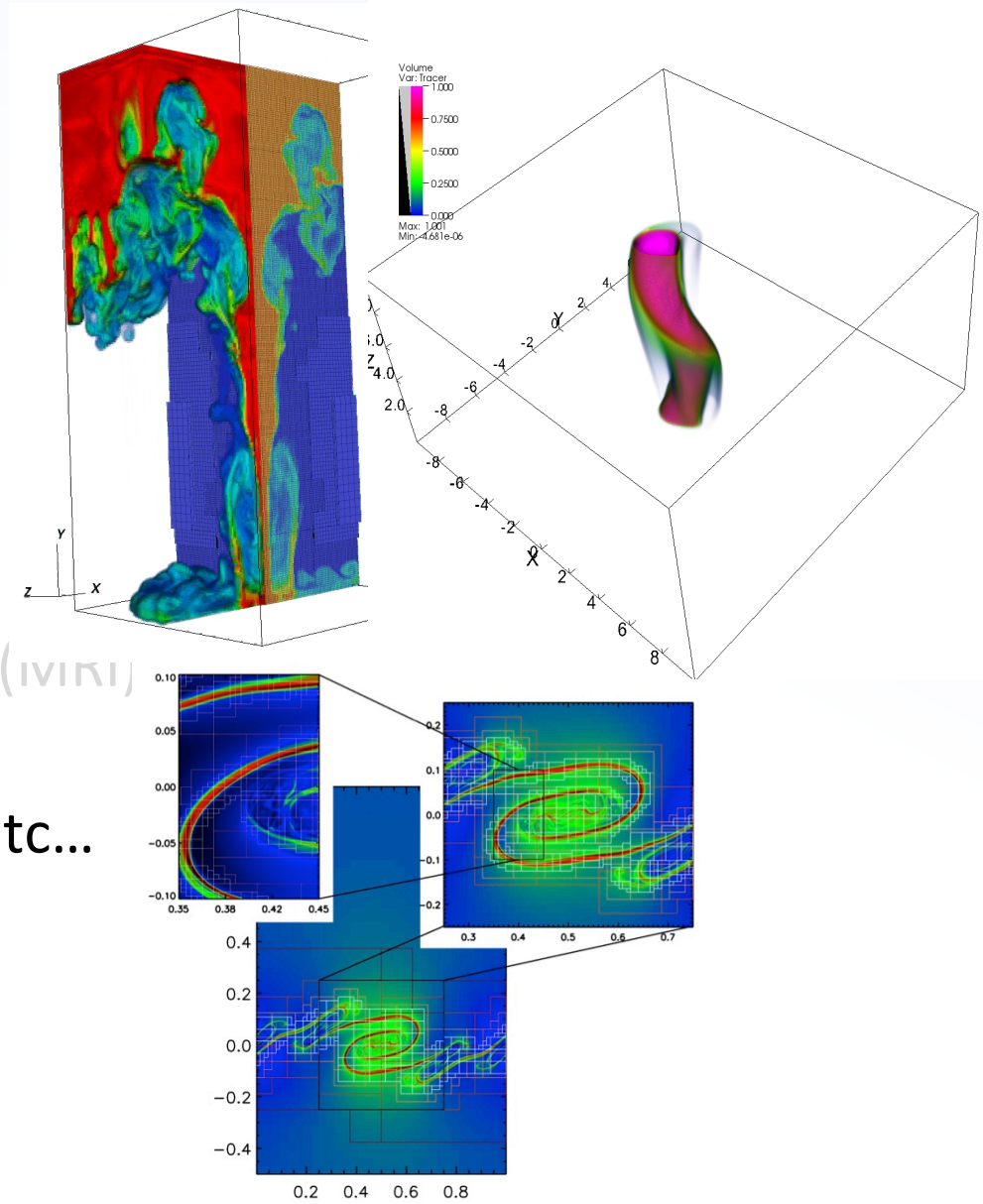
PLUTO Application Gallery

- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction
- Magneto-rotational instability (MRI) & acc
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc…



PLUTO Application Gallery

- Planet Formation
- Stellar Jets
- Radiative shocks
- Extragalactic Jets
- Jet Launching
- Magnetospheric accretion & star-disk Interaction
- Magneto-rotational instability (MRI)
- Relativistic Shock dynamics
- Fluid instabilities CD, KH, RT, etc…



Setting up PLUTO

- Download latest PLUTO release from:

  → http://personalpages.to.infn.it/~mignone/PLUTO/

- Unpack using the *tar xzvf* command

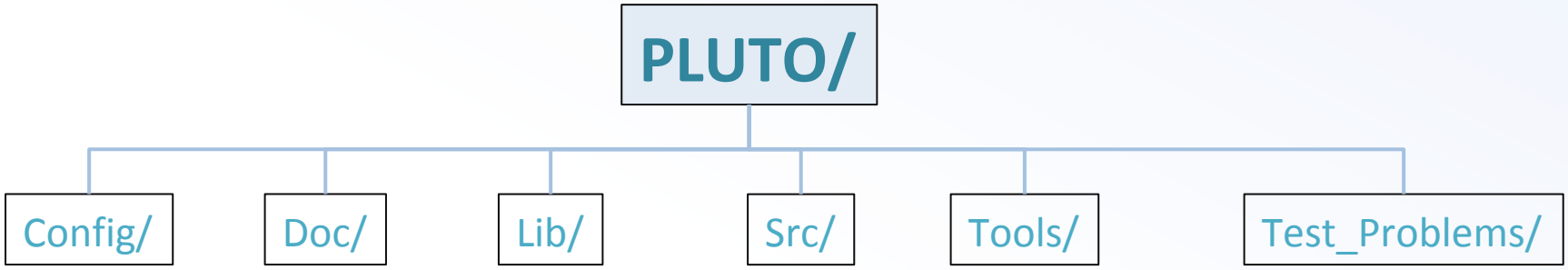
```
> tar xzvf pluto-4.3-beta-Nov2017.tar.gz
```

- This will create, at the level where you unpack, the main directory PLUTO/

```
> cd PLUTO/
```

```
> ls -l
total 96
-rw-r--r--   1 mignone  staff  18056 Jun 23 09:50 CHANGES
-rw-r--r--   1 mignone  staff  17982 Apr 18  2011 COPYING
drwxr-xr-x  10 mignone  staff    320 Nov 13 15:31 Config
drwxr-xr-x   9 mignone  staff    288 Nov 13 15:33 Doc
drwxr-xr-x   3 mignone  staff     96 Nov 13 15:31 Lib
-rw-r--r--   1 mignone  staff   2162 May 30 12:56 README
drwxr-xr-x  77 mignone  staff   2464 Nov 13 15:33 Src
drwxr-xr-x   7 mignone  staff    224 Nov 13 15:31 Test_Problems
drwxr-xr-x   7 mignone  staff    224 Nov 13 15:31 Tools
-rwxr-xr-x   1 mignone  staff   3919 Jun 19 18:03 setup.py
```

Directory Structure

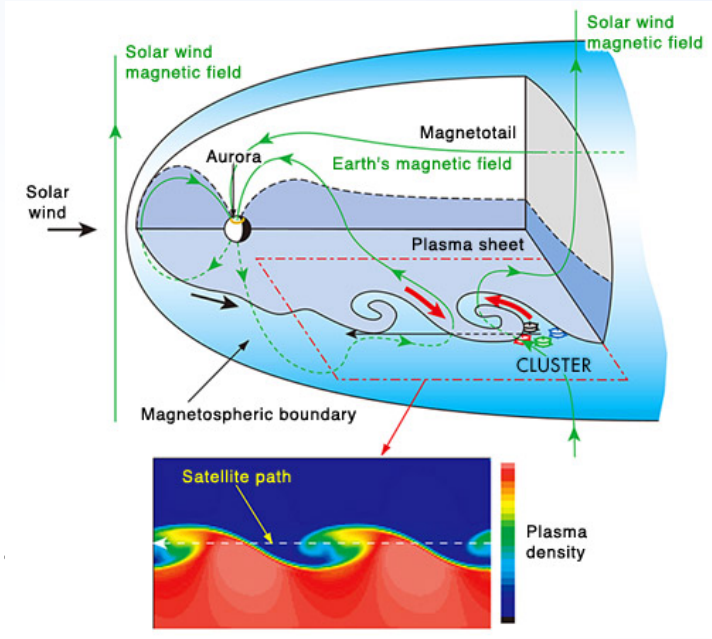


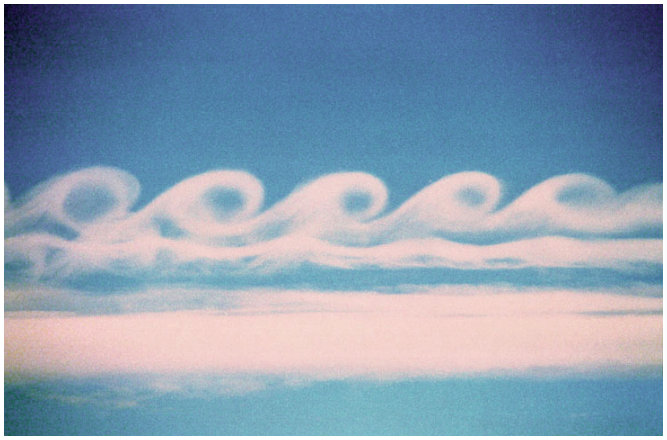
- Config/: contains machine architecture dependent files, such as information about C compiler, flags, library paths and so on. Useful for creating the *makefile*;
- Doc/: documentation directory;
- Lib/: repository for additional libraries;
- Src/: main repository for all **.c* source files with the exception of the *init.c* file, which is left to the user;
- Tools/: Collection of useful tools, such as Python scripts, IDL visualization routines, pyPLUTO, etc...;
- Test_Problems/: a directory containing several test-problems used for code verification.

EXAMPLE #1:
2D THE KELVIN-HELMHOLTZ INSTABILITY

1 - Kelvin-Helmholtz Instability

- The Kelvin–Helmholtz Instability (KHI) develops at the interface between two fluids in relative motion;
- Important in atmospheric flows, interaction between solar wind and magnetosphere (space weather), supersonic jet propagation (astrophysic etc…





Equations

- The instability may be analyzed using the compressible Euler equations,

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) & = & 0 \\ \rho \left(\dfrac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) + \nabla p & = & 0 \\ \dfrac{\partial p}{\partial t} + \boldsymbol{u} \cdot \nabla p + \gamma p \nabla \cdot \boldsymbol{u} & = & 0 \end{cases}$$

- In the vortex-sheet approximation, the velocity at equilibrium has a jump in the transverse direction

$$\mathbf{v} = \begin{cases} +M/2 \hat{\mathbf{e}}_x & \text{for} \quad y > 0 \\ -M/2 \hat{\mathbf{e}}_x & \text{for} \quad y < 0 \end{cases}$$

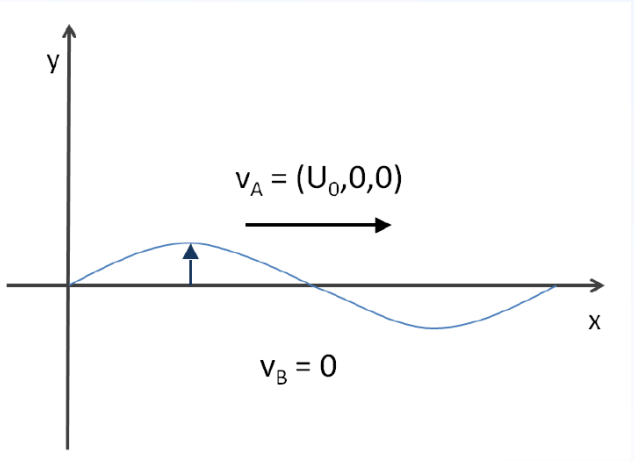
- Equilibrium density and pressure are constant,

Equations

- A linearization of the equations can be carried out for small perturbations,

$$q(\mathbf{x},t) = q_0 + q'(\mathbf{x},t)$$

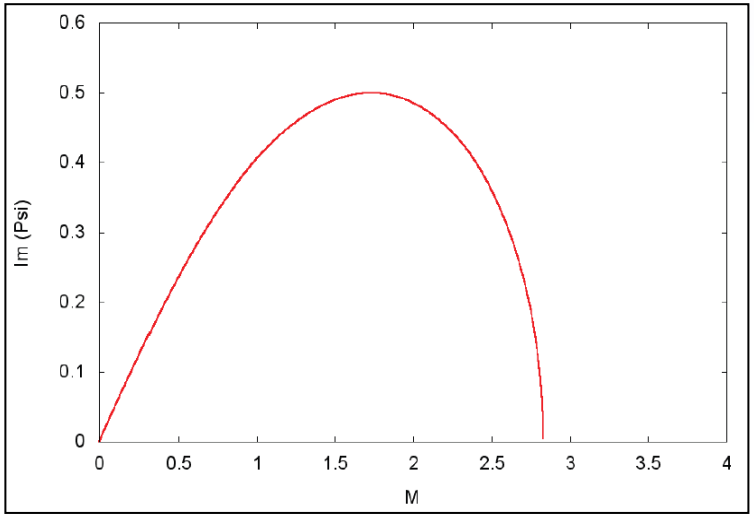
- where q' = f(y)$e^{i(kx - \omega t)}$ is a complex quantity.



- The linearization process leads to the following dispersion relation

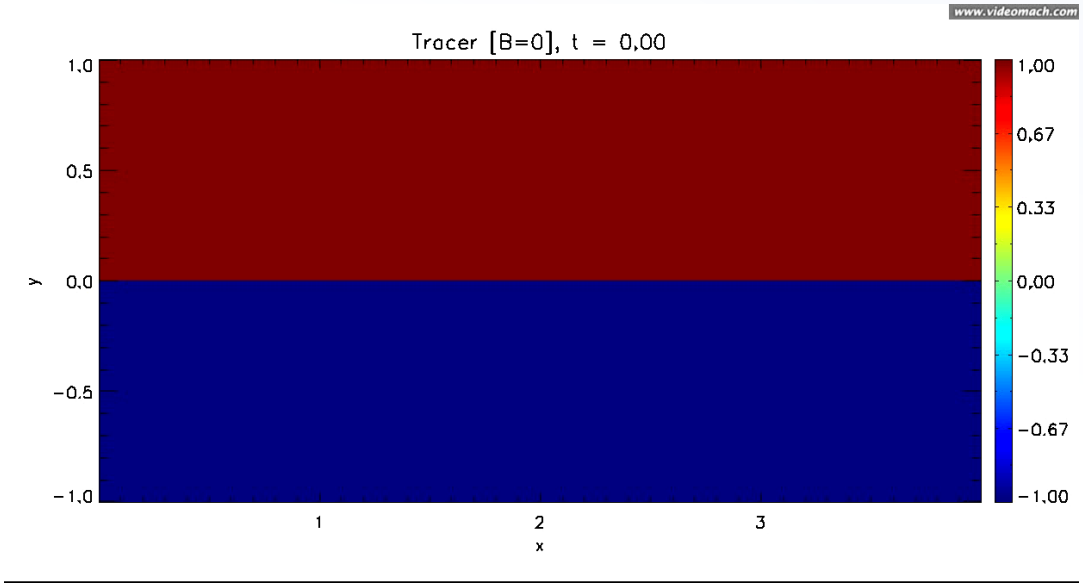
$$\frac{\omega}{kc_s} = \frac{M}{2} \pm i\left[\sqrt{M^2+1} - \left(\frac{M^2}{4}+1\right)\right]^{1/2}$$

- A complex value of ω(k) indicates an instability.



KH: Nonlinear evolution

- The growth of instability leads to a deformation of the interface creating a typical roll-up structure with vortex formation:



- Kinetic (ordered) energy is dissipated into disordered energy and the growth of perturbation at the interface leads to the generation of interacting vortices.

- Vortex merging leads to larger velocity shear and mixing of fluids from the two different regions.

Preparing to Run PLUTO

- PLUTO should be compiled and executed in a separate working directory which may be anywhere on your local hard drive
- To this end, we first need to set the environment variable PLUTO_DIR to point to this directory. In a bash shell,

```
> export PLUTO_DIR=$HOME/tmp/PLUTO
```

- Change directory to Test_Problems/MHD/Kelvin_Helmholtz;

- In order to configure PLUTO, *four* basic steps are needed:

1. Creating the problem header file (*definitions.h*);
2. Choosing the *makefile*;
3. Tuning the runtime initialization file *pluto.ini*;
4. Coding initial & boundary conditions (*init.c*);

Through the Python script

Manually

The Python Menu (Step #1 & #2)

- Run the *python* script:

```
> python $PLUTO_DIR/setup.py
```

- The script will now enter into the main menu:

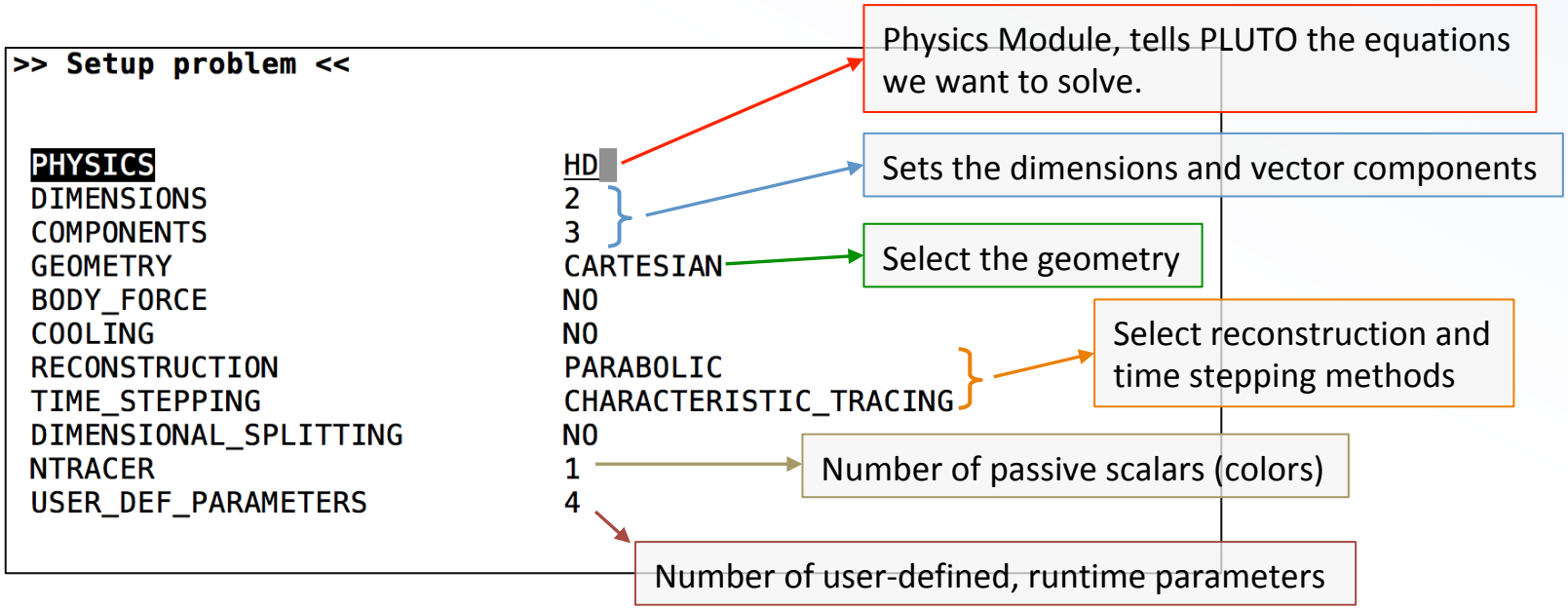
```
>> Python setup (Nov 2017) <<

Working dir: /Users/mignone/Didattica/Fluidi_e_Plasmi/Codes/PLUTO/KH
PLUTO dir  : /Users/mignone/PLUTO


 Setup problem
 Change makefile
 Auto-update
 Save Setup
 Quit
```

- Press enter under "*Setup problem*"

```
>> Setup problem <<


 PHYSICS                    HD
 DIMENSIONS                 2
 COMPONENTS                 3
 GEOMETRY                   CARTESIAN
 BODY_FORCE                 NO
 COOLING                    NO
 RECONSTRUCTION             PARABOLIC
 TIME_STEPPING              CHARACTERISTIC_TRACING
 DIMENSIONAL_SPLITTING      NO
 NTRACER                    1
 USER_DEF_PARAMETERS        4
```

The “Setup problem” menu

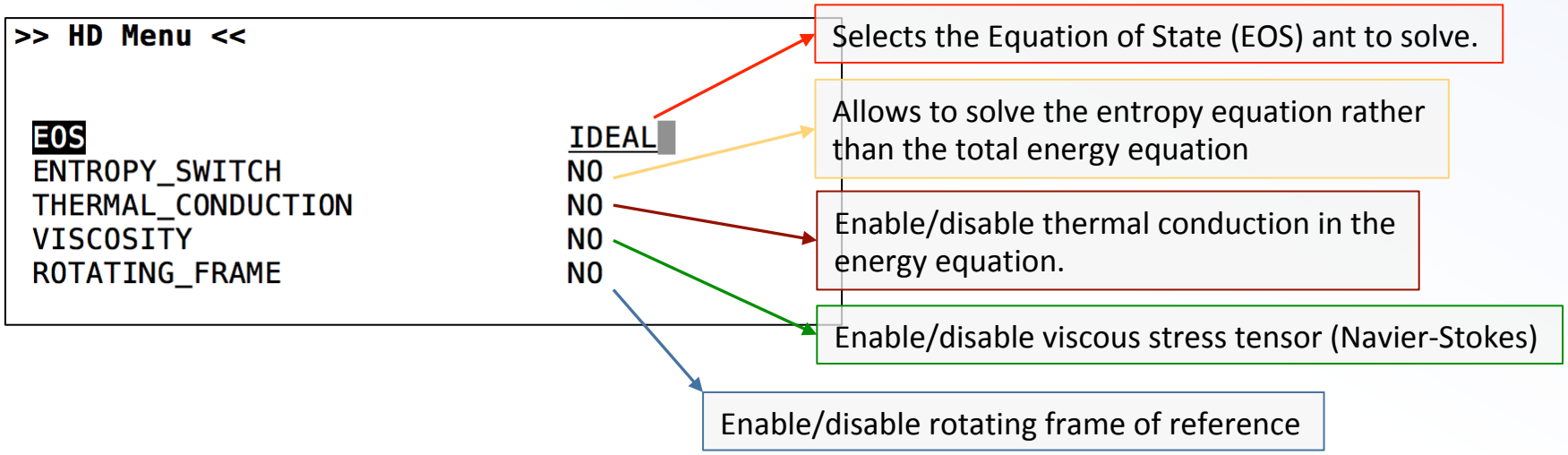
- In a self-explanatory way, the setup menu allows the user to configure the basic features for your problem:



- By pressing enter, you’ll be directed to a second menu →

The “Physics” Sub-menu

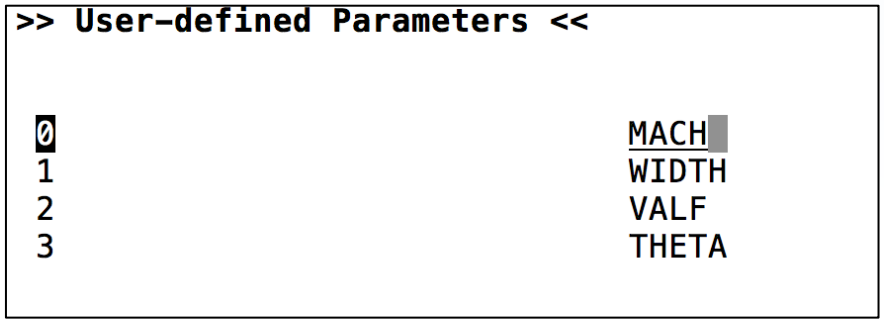
- In the following menu, only directive relative to the HD module will be shown



- For the problem at hand, we neglect dissipative effects.

User-Defined Parameters

- The third menu can be used to setup parameter names:



Makefile Menu

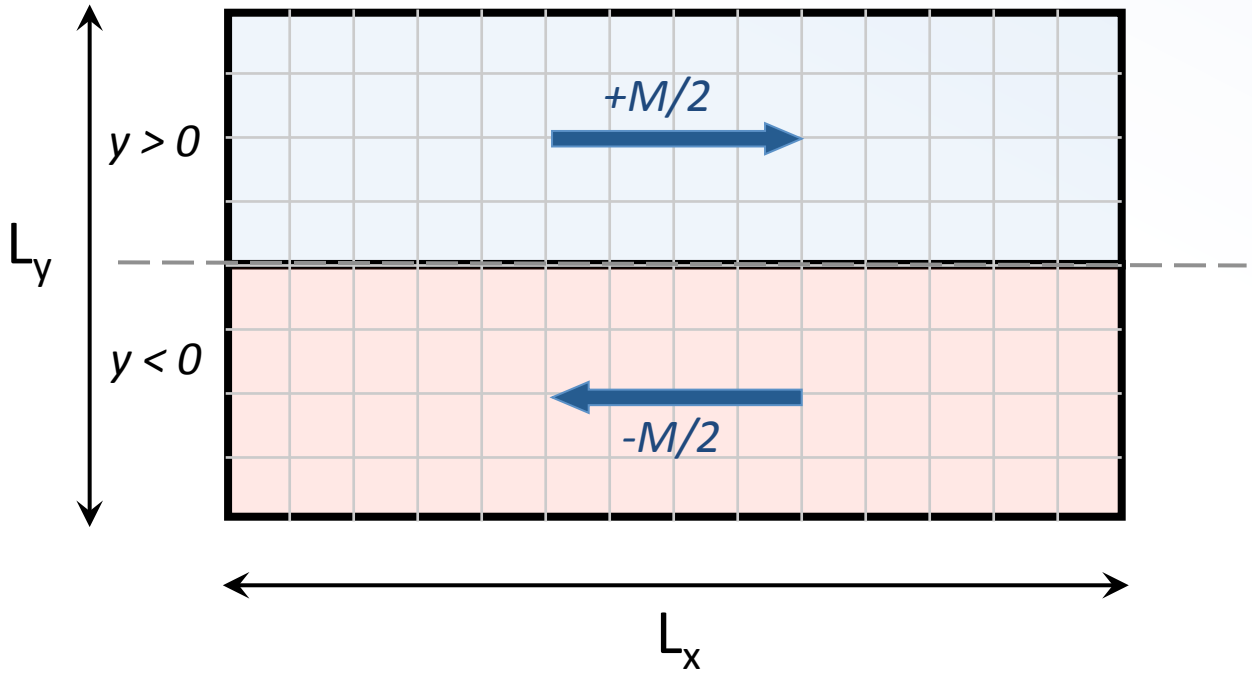
- Once we're back to the main menu, we can select a different makefile (→ "*Change makefile*")

```
>> Change makefile <<


 Aurora.gcc.defs
 Aurora.mpicc.defs
 CYGWIN_NT-6.1.x86_64.gcc.HDF5.defs
 Darwin.gcc.defs
 Darwin.mpicc.defs
 FERMI.mpixlc.defs
 Linux.gcc.defs
 Linux.mpicc.defs
 MARCONI.mpiicc.defs
 Template.defs
 debug.defs
 phw184mc.gcc.defs
 phw184mc.mpicc.defs
 preprocessed.defs
 profile.defs
```

- Configuration files are taken from the Config/ directory.
- If you have no idea, simply go with *Linux.gcc.defs* (for a serial run) or *Linux.mpicc.defs* (for a parallel run).

Setting Initial & Boundary Conditions (Step #3 & #4)

- We start considering a 2D Cartesian box, filled with two uniform fluids ($\rho = 1$, $p = 1/\Gamma$) in pressure equilibrium and in relative motion:



- For simplicity we choose a periodic boundary in the x-direction and zero-gradient boundary conditions in the y-direction.

Specifying Initial Conditions:

- Initial condition must be coded inside *init.c* file using the `Init()` function;
- This file is always in your local working directory.

```
/* ********************************************************************* */
void Init (double *v, double x, double y, double z)
/*
 *
 *********************************************************************** */
{
  static int first_call = 1;
  double rnd, eps;
  double M     = g_inputParam[MACH];
  double w     = g_inputParam[WIDTH];
  double vA    = g_inputParam[VALF];
  double theta = g_inputParam[THETA]/180.0*CONST_PI;

  if (first_call == 1){          /* Seed random number sequence */
    srand(time(NULL) + prank);
    first_call = 0;
  }
  rnd = (double)(rand())/((double)RAND_MAX + 1.0); /* Generate random number */
  eps = 0.01*M;                                    /* Perturbation amplitude */

  v[RHO] = 1.0;                                    /* Set constant density  */
  v[VX1] = 0.5*M*tanh(y/w);                        /* Main flow             */
  v[VX2] = eps*2.0*(rnd-0.5)*exp(-y*y*10.0); /* Vertical perturbation */
  v[VX3] = 0.0;

  v[PRS] = 1.0/g_gamma;                  /* Set constant pressure (cs = 1) */
  v[TRC]  = (y < 0.0 ? 1.0:-1.0);        /* Passive tracer (color)         */

  #if PHYSICS == MHD
  v[BX1] = vA*cos(theta);     /* Magnetic field lies in the x-z plane.        */
  v[BX2] = 0.0;               /* Theta is the angle between B and x-direction */
  v[BX3] = vA*sin(theta);

  v[AX1] = 0.0;
  v[AX2] = 0.0;
  v[AX3] = y*v[BX1];
  #endif
}
```

Runtime Parameters: *pluto.ini*

- At runtime, PLUTO reads the *pluto.ini* file which is used to control several options use by the code at runtime, such as grid generation, CFL number, boundary conditions, output type and so forth.

- The file contains several blocks, of the form

```
[Block]

label       ...   fields   ...
label       ...   fields   ...
label       ...   fields   ...
```

- This file can be edited manually.

```
[Grid]

X1-grid  1   0.0  160  u  1.0
X2-grid  1  -1.0  320  u  1.0
X3-grid  1   0.0    1  u  1.0

[Time]

CFL              0.8
CFL_max_var      1.1
tstop            10.0
first_dt         1.e-4

[Solver]

Solver         roe

[Boundary]

X1-beg        periodic
X1-end        periodic
X2-beg        outflow
X2-end        outflow
X3-beg        outflow
X3-end        outflow

[Static Grid Output]

uservar    0
dbl        10.0  -1   single_file
flt       -1.0   -1   single_file
vtk        0.1   -1   single_file
tab       -0.1   -1
ppm       -1.0   -1
png       -1.0   -1
log        10
analysis  -1.0   -1

[Parameters]

MACH                    5.0
WIDTH                   0.00001
VALF                    0.1
THETA                   90.0
```

Compiling and Running

- PLUTO can now be compiled by typing “make” at the command prompt.

- You can now run the code by typing:

  - Local machine, serial run: *./pluto*
  - Local machine, parallel run: *mpirun −np <n> ./pluto*
  - Cluster (Marconi): use the *qsub* command for job submission

Output log

```
   ...

> Memory allocation
> Assigning initial conditions (Startup) ...
> Normalization Units:

   [Density]:      1.673e-24 (gr/cm^3), 1.000e+00 (1/cm^3)
   [Pressure]:     1.673e-14 (dyne/cm^2)
   [Velocity]:     1.000e+05 (cm/s)
   [Length]:       1.496e+13 (cm)
   [Temperature]:  1.203e+02 X (p/rho*mu) (K)
   [Time]:         1.496e+08 (sec), 4.744e+00 (yrs)

> Number of processors: 4
> Proc size:            80 X 160
> Parallel Directions:  X1/X2
> Writing file #0 (dbl) to disk... [ 0.00 sec]
> Writing file #0 (vtk) to disk... [ 0.00 sec]
> Starting computation...

step:0; t = 0.0000e+00; dt = 1.0000e-04; 0 %; [0.000000, 0]
step:10; t = 1.5937e-03; dt = 2.5937e-04; 0 %; [0.500702, 0]
step:20; t = 5.7275e-03; dt = 6.7275e-04; 0 %; [0.503232, 0]
step:30; t = 1.6449e-02; dt = 1.7449e-03; 0 %; [0.504323, 0]
step:40; t = 4.2983e-02; dt = 3.3267e-03; 0 %; [0.503382, 0]

      ...

step:3540; t = 9.9427e+00; dt = 2.7772e-03; 99 %; [1.167689, 0]
step:3550; t = 9.9705e+00; dt = 2.7828e-03; 99 %; [1.171169, 0]
step:3560; t = 9.9983e+00; dt = 1.6718e-03; 99 %; [1.176916, 0]
> Writing file #1 (dbl) to disk... [ 0.00 sec]
> Writing file #100 (vtk) to disk... [ 0.00 sec]

> Total allocated memory    7.10 Mb (proc #0)
> Elapsed time              0d:0h:0m:55s
> Average time/step         1.54e-02  (sec)
> Local time                Tue Nov 14 09:43:32 2017
> Done
```

Physical units

Max Mach number at each step

Current time

Current time step

Everything's fine !

Visualization with IDL

- Set the IDL path to include PLUTO IDL script directory:

```
> export IDL_PATH='<IDL_DEFAULT>:'$HOME/tmp/PLUTO/Tools/IDL
```

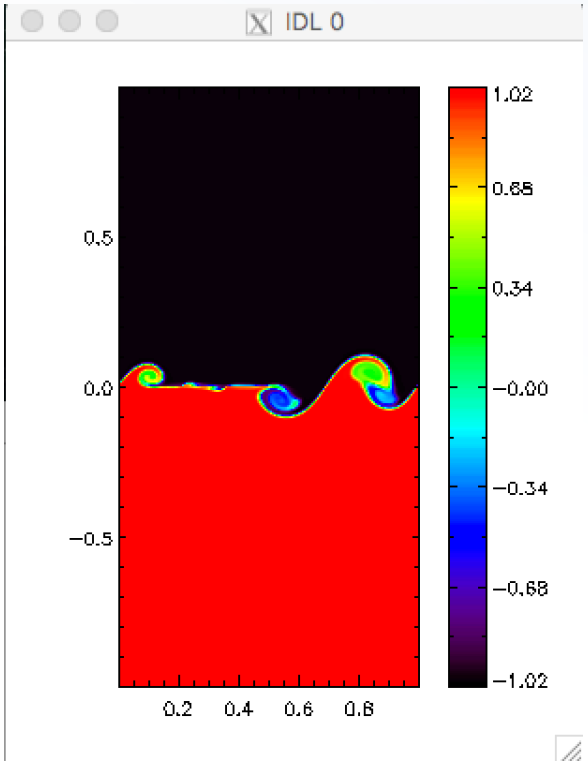
- Launch IDL, l

```
IDL> pload,/vtk,25
```

Load output data #25 in vtk format

```
IDL> display,x1=x1,x2=x2,/vbar,tr1
```

Display, the tracer field adding Coordinates and vertical colorbar.



Visualization with Gnuplot

- Set the path

```
> export GNUPLOT_LIB=$PLUTO_DIR/Tools/Gnuplot
```

- After launching gnuplot,

```
gnuplot> dsize=4
```

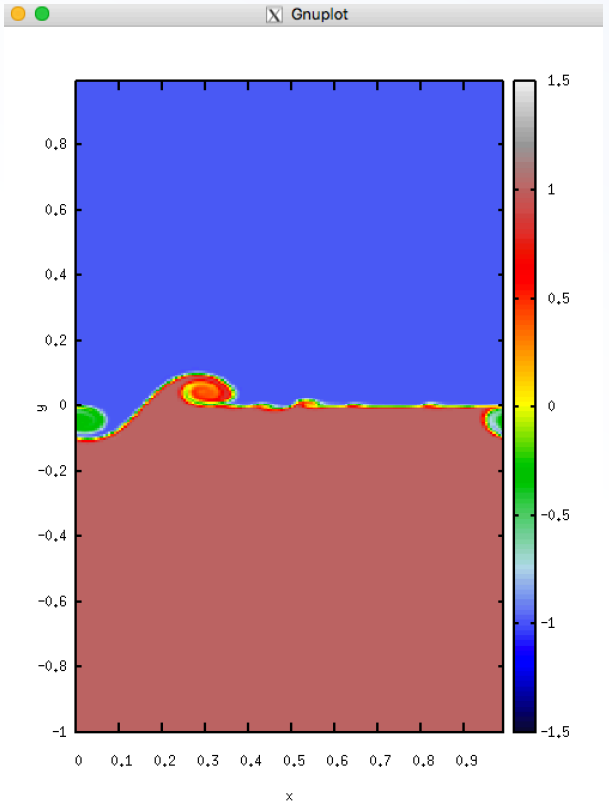
Tells gnuplot we're reading single precision (4 bytes) datafile

```
gnuplot> load "pluto.gp"
```

Read Gnuplot scripts

```
gnuplot> nvar=5; splot "data.0025.flt" @BINARR
```

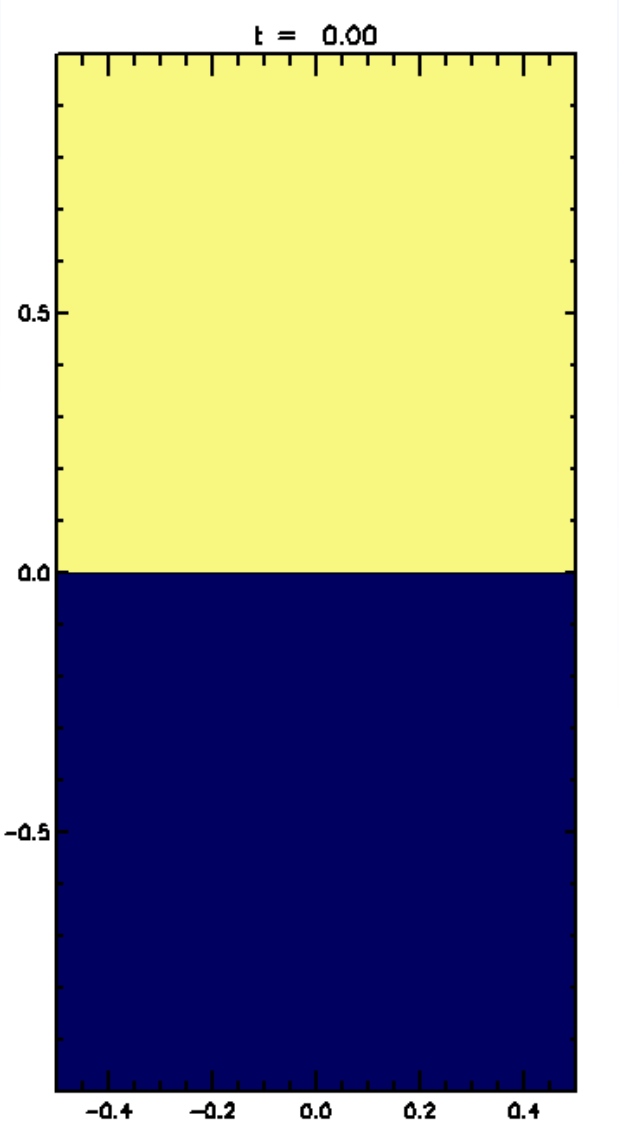
Display variable #5 (tracer) on screen



EXAMPLE #2: RAYLEIGH-TAYLOR INSTABILITY

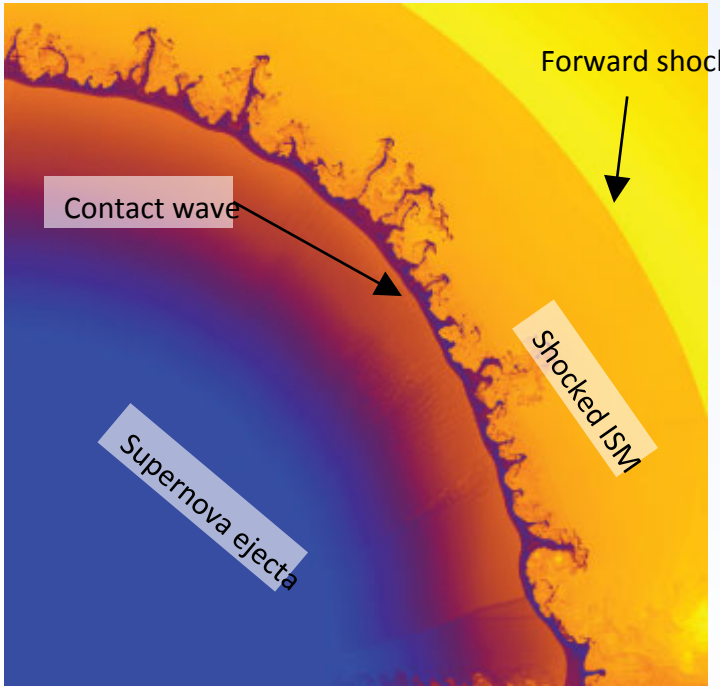
The Rayleigh-Taylor Instability

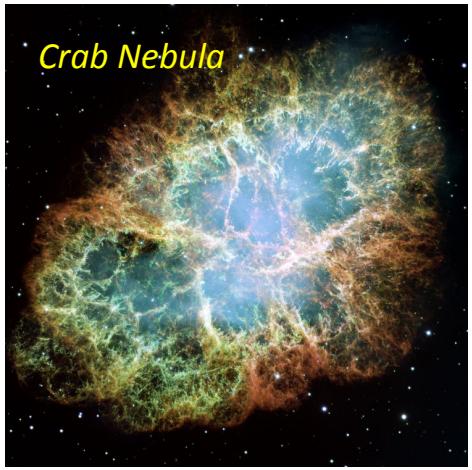
- The Rayleigh-Taylor (RT) instability occurs at the interface between two fluids of different densities, when the light one supports the heavier ones against graivity.
- The amplitude growths and the further upward motion of the lighter fluid assumes the form of rising bubble while the sinking fluid become finger-shaped.
- As the instability proceeds, fingers evolve into mushroom-like vortex motion accompainied by secondary shear flow instabilities.



Astrophysical Application: SN Remnants

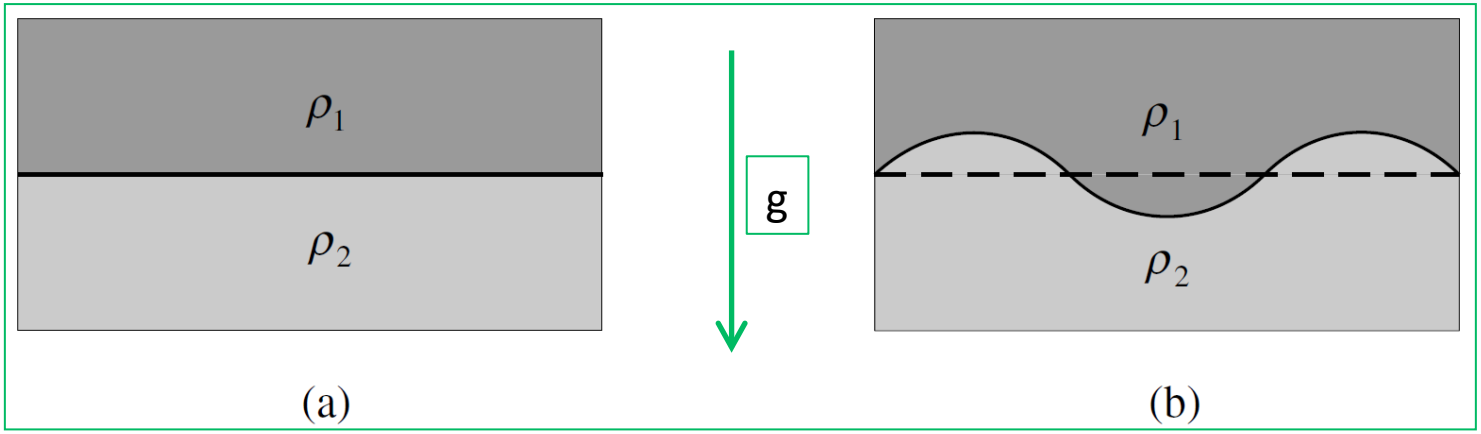
- In a supernova (SN) explosion a large amount of energy is released resulting in a formation of a large scale SN remnant (SNR)
- The dense shell of ejected material *decelerates* in a rarefied interstellar medium (ISM) and is unstable to RT-type instabilities.
- Responsible for finger-like structures of material protruding from the contact discontinuity between the two media.
- These instabilities modify the morphology of the SNR causing a departure of the ejecta from spherical symmetry.





RTI: Analysis

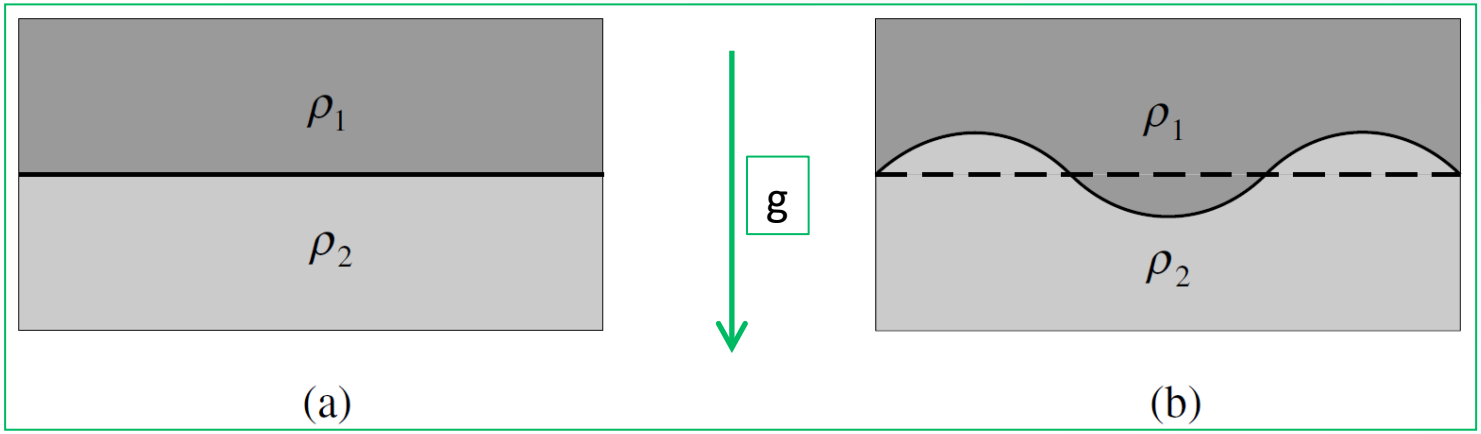
- Consider the case where two fluids with densities ρ_1 and ρ_2 are on top of each other:



- This is an equilibrium situation if the stratification is supported by a pressure gradient: grad(p) = ρg.

- Is this a stable equilibrium ?

Rayleigh-Taylor Instability (RTI)

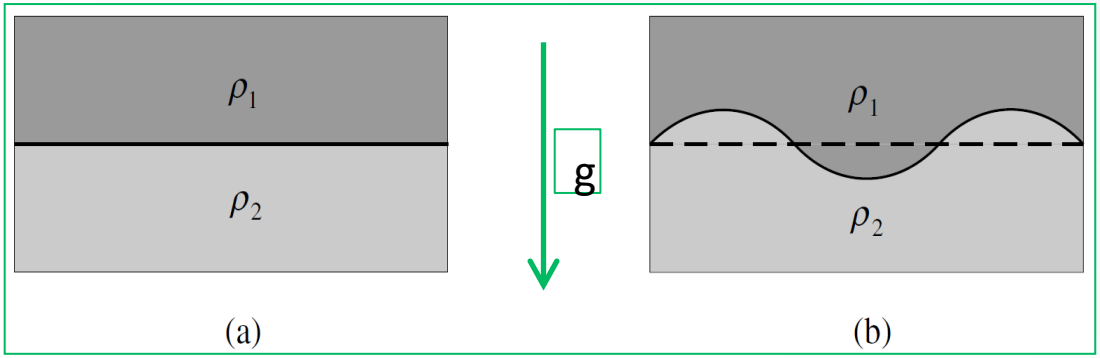


- Let's now perturb the equilibrium by rippling the boundary layer.

- The fluid element of density ρ_1 has moved downwards with consequent loss of gravitational potential energy while the opposite is true of the fluid element of density ρ_2.

- It is intuitively obvious that the only stable equilibrium is to have the denser fluid supporting the less dense. The instability that arises when $\rho_2 < \rho_1$ is called the Rayleigh–Taylor instability.

RTI: Linear Analysis



- From normal mode analysis it is found that, in absence of magnetic field,

$$\omega^2 = -\frac{kg(\rho_1 - \rho_2)}{\rho_1 + \rho_2}$$

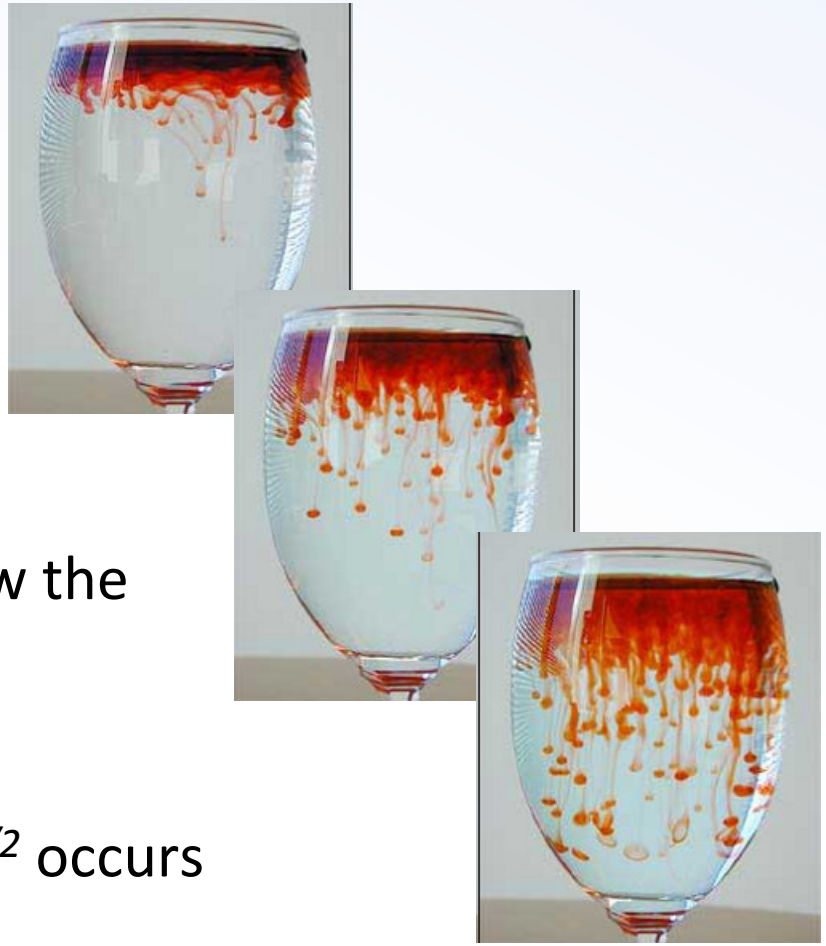
- If $\rho_1 < \rho_2$ the equilibrium is *stable* → surface gravity waves

- If $\rho_1 > \rho_2$ the equilibrium is *unstable* → Rayleigh-Taylor instability

RTI: Linear Evolution

- When $\rho_1 > \rho_2$ the growth rate is purely imaginary and this corresponds to an instability:

$$\omega = \pm i\sqrt{kg\frac{\rho_1 - \rho_2}{\rho_1 + \rho_2}}$$

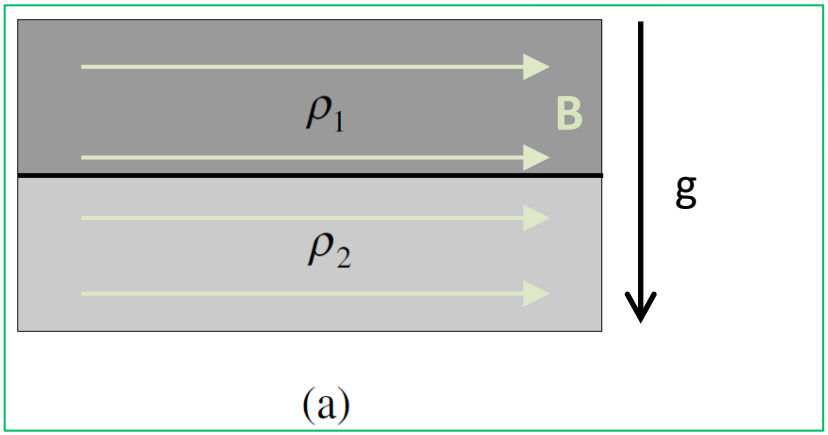


- The heavier fluid will try to sink below the lighter fluid.

- Clearly, the fastest growth rate $(kg)^{1/2}$ occurs when $\rho_1 >> \rho_2$.

RTI: Effects of Magnetic Fields

- In a plasma, density and pressure are tied, magnetic field can change pressure but not density.



- The presence of a uniform magnetic field has the effects of reducing the growth rate of modes parallel to it, although the interface still remains Rayleigh–Taylor unstable in the perpendicular direction.

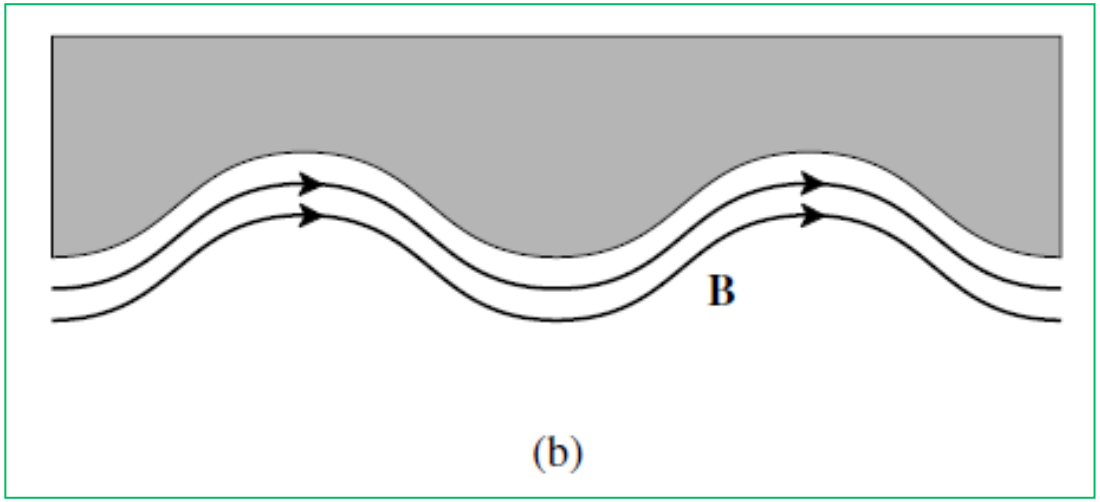
Magnetized RTI: Linear Analysis

- The growth rate now becomes

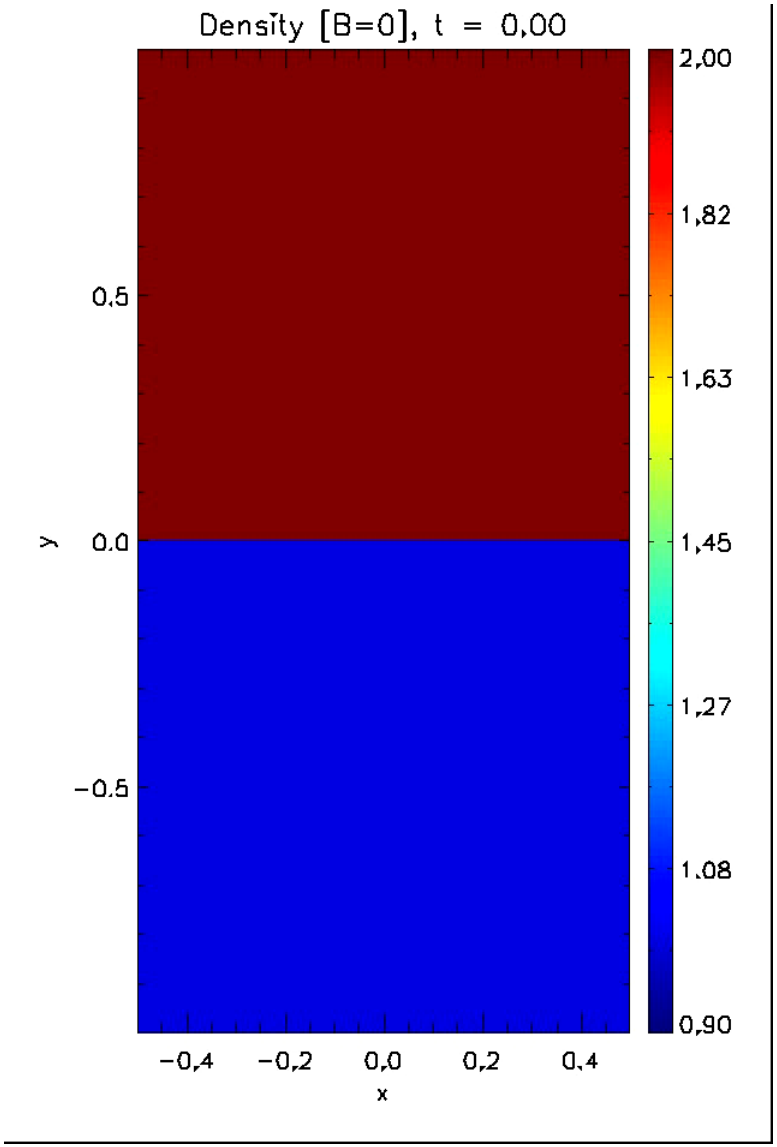
$$\omega^2 = -|\boldsymbol{k}|\, g\frac{\rho_1 - \rho_2}{\rho_1 + \rho_2} + 2\frac{(\boldsymbol{k}\cdot\boldsymbol{B}_0)^2}{4\pi(\rho_1 + \rho_2)}$$

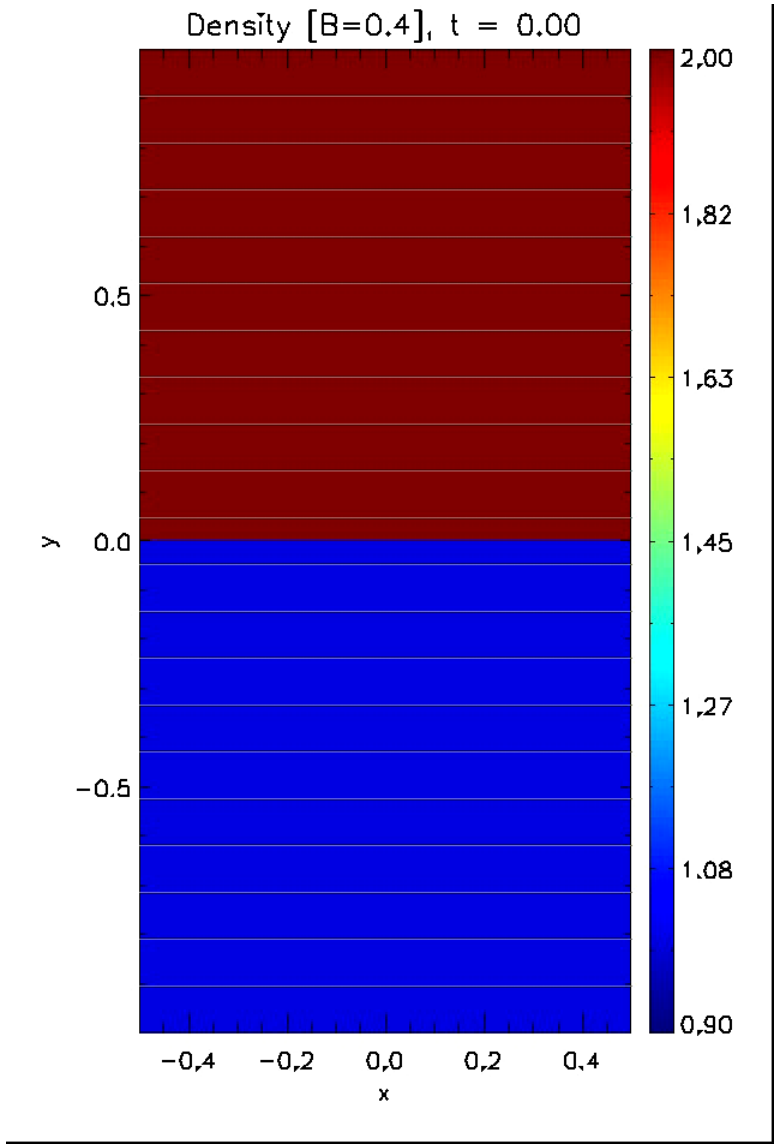
- Shorter wave modes such that $k \geq \dfrac{\mu_0 g(\rho_1 - \rho_2)}{2B_0^2\cos^2\theta}$ are stabilized.

- Bending of field lines resist to the growth of perturbation:



Nonlinear Evolution: HD vs MHD





Initial condition (Test_Problems/MHD/Rayleigh_Taylor/)

```c
/* ********************************************************************** */
void Init (double *v, double x1, double x2, double x3)
/*
 *
 **********************************************************************  */
{
  double x=x1, y=x2, z=x3;
  double rnd, Bc, g;

  g    = g_inputParam[GRAV];

  if (y < 0.0) v[RHO] = 1.0;
  else         v[RHO] = g_inputParam[ETA];

  v[PRS] = 1.0/g_gamma + v[RHO]*g*y;  /* Hydrostatic balance */
  v[VX1] = v[VX2] = v[VX3] = 0.0;

/* ---------------------------------
    Add perturbation
   --------------------------------- */

  #if USE_RANDOM_PERTURBATION == YES
   rnd    = RandomNumber(-1,1);
   v[VX2] = 1.e-2*rnd*exp(-y*y*200.0);
  #else
   #if DIMENSIONS == 2
    v[VX2] = -1.e-2*(1.0 + cos(2.0*CONST_PI*x))*exp(-y*y*200.0);
   #elif DIMENSIONS == 3
    rnd = sqrt(x*x + z*z);
    v[VX2] = -1.e-2*exp(-rnd*rnd/(0.2*0.2))/cosh(y*y/(0.1*0.1));
   #endif
  #endif

/* ---------------------------------
    Set magnetic field in units of Bc
   --------------------------------- */

  #if PHYSICS == MHD
   Bc = sqrt((g_inputParam[ETA] - 1.0)*fabs(g)); /* Critical field */
   v[BX1] = g_inputParam[CHI]*Bc/sqrt(4.0*CONST_PI);
   v[BX2] = 0.0;
   v[BX3] = 0.0;
   v[AX1] = v[AX2] = 0.0;
   v[AX3] = y*v[BX1];
  #endif

}
```

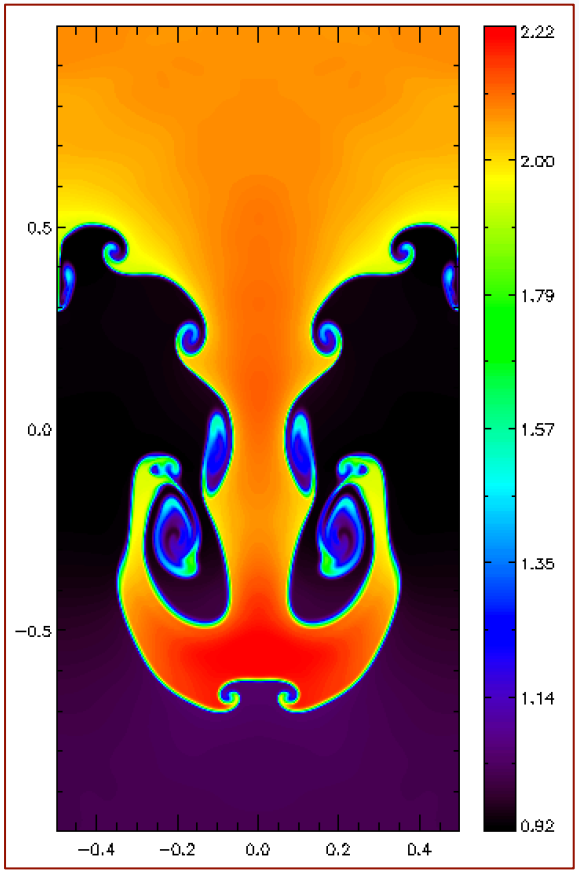
$$\rho(y) = \begin{cases} 1 & \text{for} \quad y \le 0 \\ \eta & \text{for} \quad y > 0 \end{cases}, \qquad p = p_0 + \rho y g$$

$$\boldsymbol{B} = \chi B_c \hat{\mathbf{i}}, \qquad B_c \equiv \sqrt{(\rho_{\rm hi} - \rho_{\rm lo}) L |g|} \quad \rightarrow \quad \sqrt{(\eta - 1.0)|g|}$$

Runinng the Test Problem…

- Run the python script, select your makefile
- Change the resolution from pluto.ini
- Compile
- Run
- Use IDL, gnuplot (or other) to visualize data.

ρ, t = 15



EXAMPLE #3:
JET PROPAGATION

Accretion Physics

- *Accretion disk*

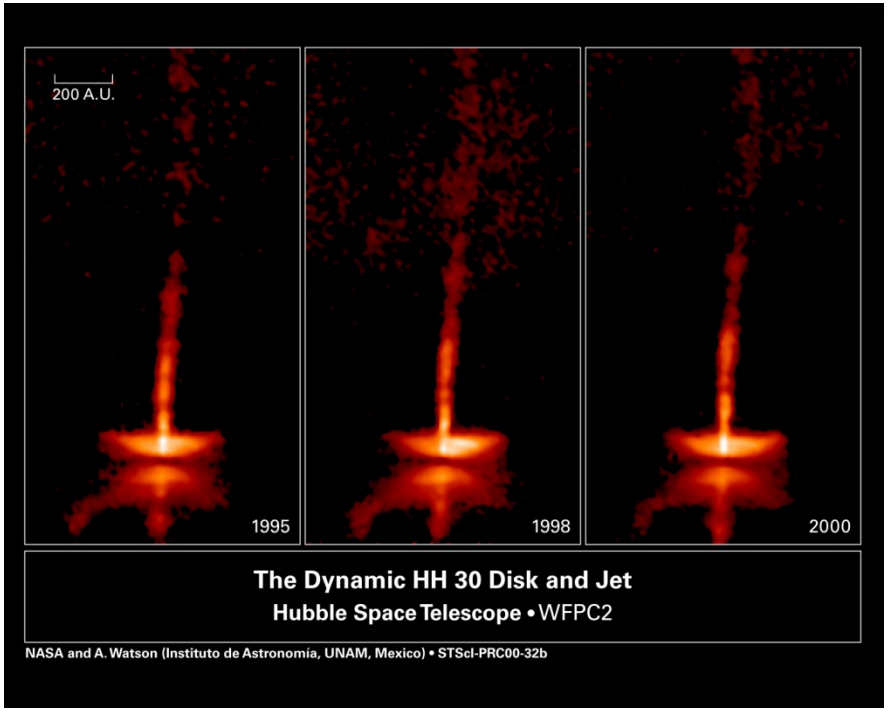
  structure of gas in orbital motion around a central object (typically newborn star, neutron star, supermassive black hole);

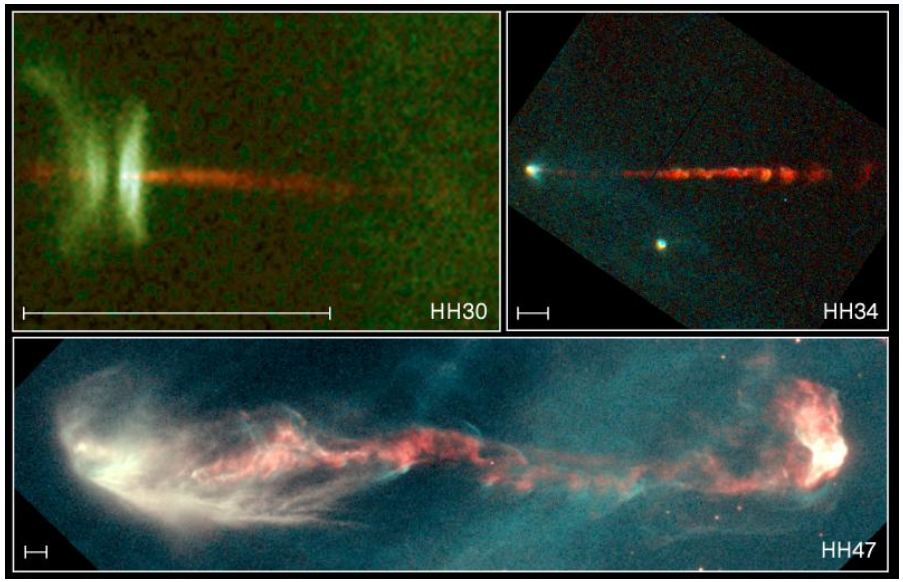
- *Jet*

  collimated outflow emerging from the innermost regions of an accretion disk.

Accretion/Ejection systems

Star Formation Regions / Young Stellar Objects (YSO)





vel ~100–500 km/s; n~10^3–10^4 cm^{-3}; Size~10^3–10^5 AU; T~10^3–10^4 K; Age: 10^4–10^5 yrs

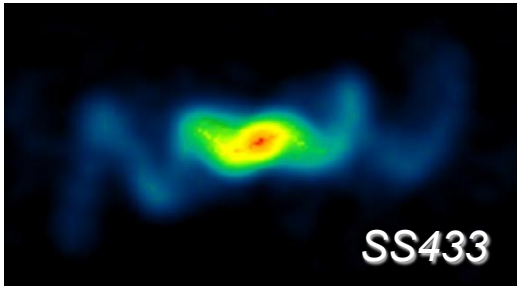
Thermal emission processes

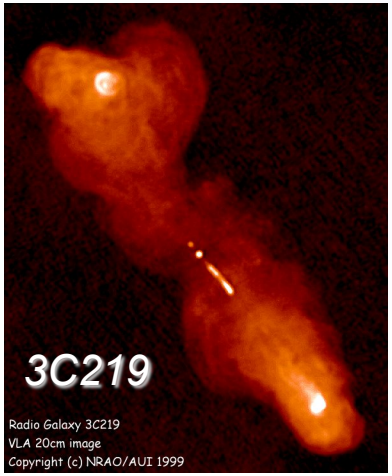
Accretion/Ejection systems

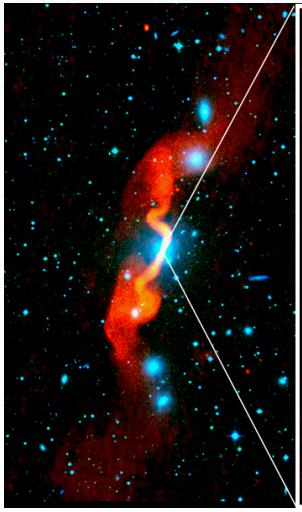
Compact Objects (BH, NS)

X-Ray Binaries

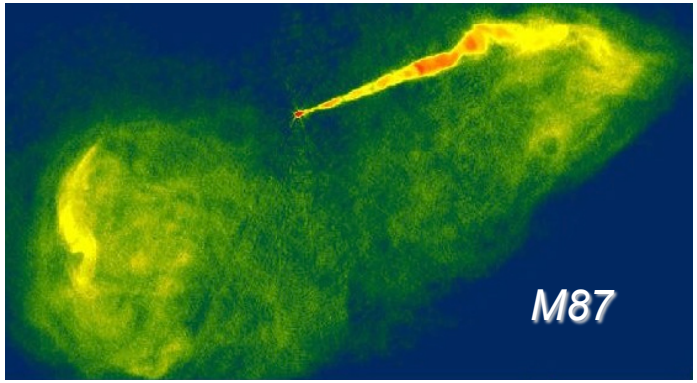
Micro Quasars







Active Galactic Nuclei (AGN)

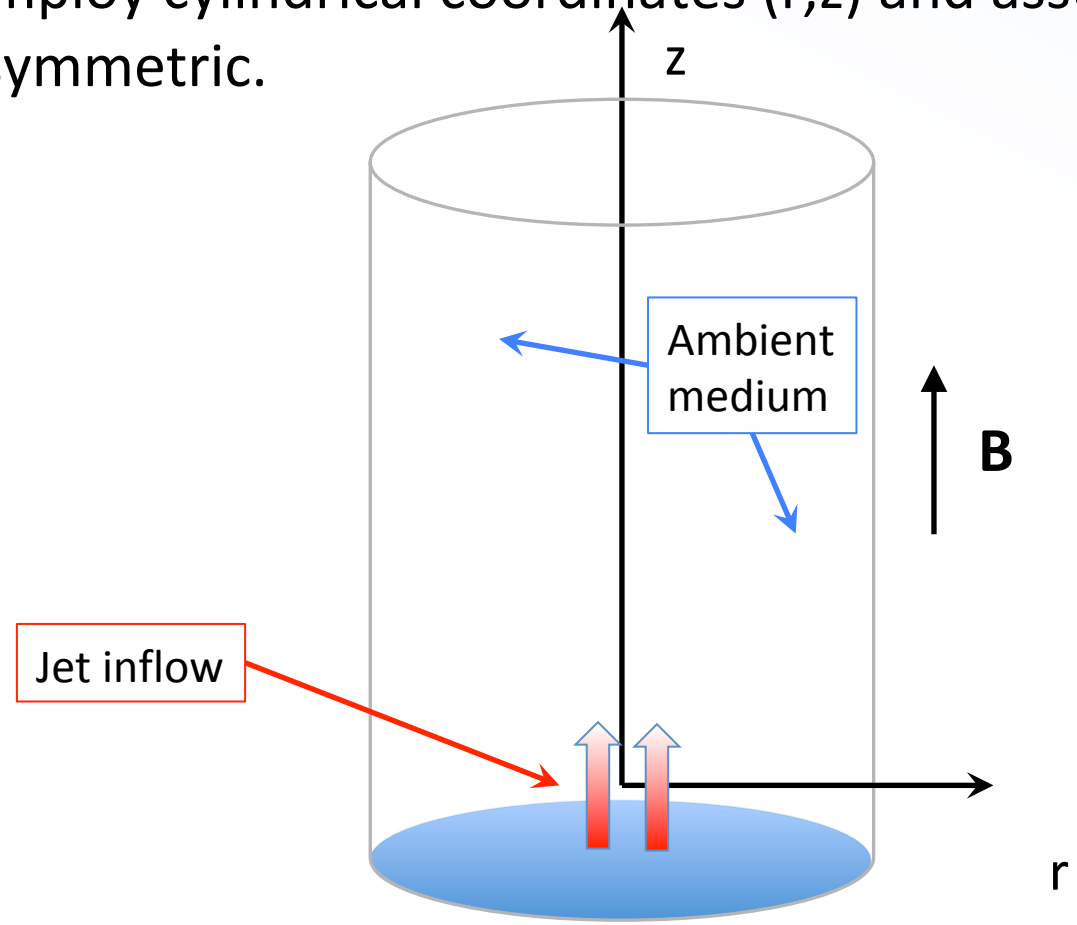


Vel~10^{-2}-1 c; n~10^{-5}-10^{-3} cm $^{-3}$; Size ~ few kpc–Mpc; Age~10^{7}-10^{8} yrs;

Non-thermal emission processes

Setting up the Problem: Initial Conditions

- Test problem can be found in Test_Problems/MHD/Jet_Simplified/
- We employ cylindrical coordinates (r,z) and assume the problem is axisymmetric.



Setting up the Problem: Initial Conditions

- The Init() function is now used to setup the ambient conditions only.

```
/* *********************************************************
void Init (double *v, double x1, double x2, double x3)
/*
 * Define ambient values and units.
 *********************************************************
{
  int    nv;
  double nH = 200.0;
  static double veq[NVAR];

  v[RHO] = 1.0;
  v[VX1] = v[VX2] = v[VX3] = 0.0;
  v[PRS] = pa = 0.6;

#if PHYSICS == MHD
  EXPAND(v[BX1] = 0.0;                                    ,
         v[BX2] = sqrt(2.0*g_inputParam[SIGMA_Z]*pa); ,
         v[BX3] = 0.0;)

  v[AX1] = v[AX2] = 0.0;
  v[AX3] = 0.5*x1*v[BX2];
#endif

  v[TRC] = 0.0;

  g_smallPressure  = 1.e-3;
}
```

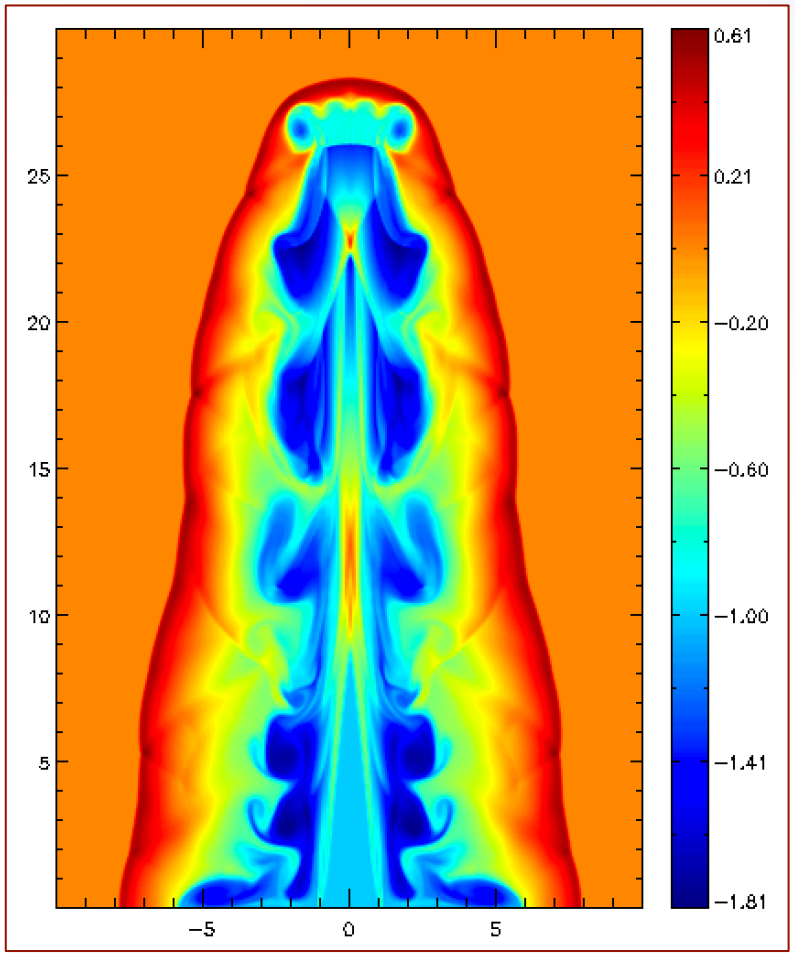
Setting up the Problem: Boundary Conditions

- Boundary conditions at the lower z-boundary (X2_BEG) are used to inject the supersonic beam;
- A supersonic beam is injected in a small nozzle at the lower (z=0) boundary using a user-defined boundary condition.

```
/* ********************************************************************* */
void UserDefBoundary (const Data *d, RBox *box, int side, Grid *grid)
/*!
 *
 *********************************************************************** */
{
  int    i, j, k, nv;
  double *x1 = grid->xgc[IDIR];
  double *x2 = grid->xgc[JDIR];
  double *x3 = grid->xgc[KDIR];
  double r, vjet[256], vout[NVAR], q;
  double t, omega; /* pulsed jet frequency */

  if (side == X2_BEG){  /* -- X2_BEG boundary -- */
    if (box->vpos == CENTER){    /* -- cell-centered boundary conditions -- */
      BOX_LOOP(box,k,j,i){
        GetJetValues (x1[i], vjet);

        /* -- copy and reflect ambient medium values -- */

        VAR_LOOP(nv) vout[nv] = d->Vc[nv][k][2*JBEG - j - 1][i];
        vout[VX2] *= -1.0;
        #if PHYSICS == MHD
        EXPAND(vout[BX1] *= -1.0; ,
                                ; ,
               vout[BX3] *= -1.0;)
        #endif
        VAR_LOOP(nv){
          d->Vc[nv][k][j][i] = vout[nv]-(vout[nv]-vjet[nv])*Profile(x1[i], nv);
        }
      }

    }else if (box->vpos == X1FACE){ /* -- staggered fields -- */
      #ifdef STAGGERED_MHD
      x1 = grid->xr[IDIR];
      BOX_LOOP(box,k,j,i){
        vout[BX1] = -d->Vs[BX1s][k][2*JBEG - j - 1][i];
        d->Vs[BX1s][k][j][i] =    vout[BX1]
                                - (vout[BX1] - vjet[BX1])*Profile(fabs(x1[i]), BX1);
      }
      #endif
    }else if (box->vpos == X3FACE){

    }
  }
}
```

Running the Test Problem…

- Have a look at pluto.ini;
- Compile
- Run
- Visualize data.



THE END