Center of Excellence for HPC
Astrophysical Applications

A Practical Introduction to gPLUTO

A. Mignone[1], M. Rossazza[1], S. Truzzi[1], V. Berta[1],
A. Suriano[1], M. Bugli[1,2], G. Mattia[3]

[1]Physics Department, University of Torino
[2]IAP-CNRS (Paris)
[3]MPIA (Heidelberg)

Prerequisites

- In order to run the code and follow the steps of this tutorial, you will need a Linux-based pc with the following requisites:
  - *gPLUTO:* [→ http://plutocode.ph.unito.it/].
  - *C++ compiler* (g++ or NVIDIA compiler nvc++)
  - The *make utility*
  - The *Python interpreter*
  - *Gnuplot* (a free visualization program)

- Optionally:
  - *VisIt*[3] [an Open source interactive visualization, → https://wci.llnl.gov/simulation/computer-codes/visit/downloads]
  - *PyPLUTO* [available in Tools/PyPLUTO folder —> https://www.theoj.org/joss-papers/joss.08448]

Downloading and Unpacking *gPLUTO*

- You may download ***gPLUTO*** using our GiTLab Repository

```
> git clone https://gitlab.com/PLUTO-code/gPLUTO
```

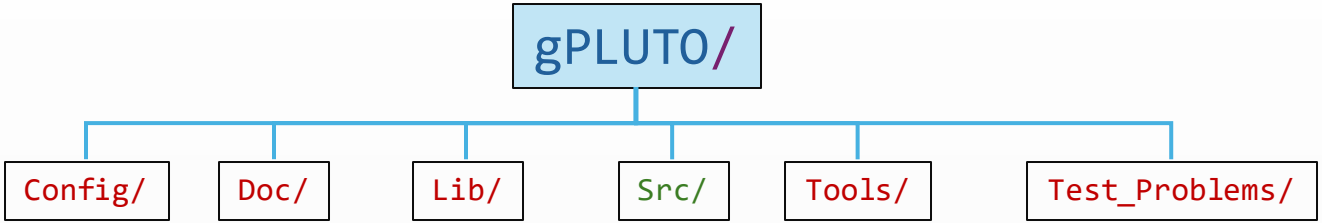
- This will create, at the level where you unpacked, the main directory `gPLUTO/`. Change directory and lists its content:

```
> cd gPLUTO
> ls -l
```

- It should look like

```
drwxr-xr-x  10 mignone  staff    320 Dec  4 00:12 Config
drwxr-xr-x   8 mignone  staff    256 Dec  4 00:13 Doc
drwxr-xr-x  85 mignone  staff   2720 Dec  4 10:51 Src
drwxr-xr-x   6 mignone  staff    192 Dec  4 11:27 Test_Problems
drwxr-xr-x   7 mignone  staff    224 Dec  4 00:12 Tools
-rwxr-xr-x@  1 mignone  staff   4056 May 13  2018 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 `*.cpp` and `*.hpp` source files with the exception of the `init.cpp` 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.

SESSION #1:
THE SHOCK-TUBE PROBLEM

Preparing to Run gPLUTO

- gPLUTO should be compiled and executed in a separate working directory which may be anywhere on your local hard drive. To this end, set the environment variable PLUTO_DIR correctly:

```
> export PLUTO_DIR=/<fullpath>/gPLUTO  # set it also in your .bashrc or similar
```

- Change directory to Test_Problems/HD/Shock_Tubes:

```
> cd $PLUTO_DIR/Test_Problems/HD/Shock_Tubes
```

- In order to configure gPLUTO, 4 basic steps needs to be followed:

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

Through the Python script

Manually

The Python Menu (Step #1 & #2)

- Copy, e.g., the first configuration and then run the the *python* script:

```
> cp definitions_01.hpp definitions.hpp
> cp pluto_01.ini pluto.ini
> python3 $PLUTO_DIR/setup.py
```

- The script will now enter into the main menu:

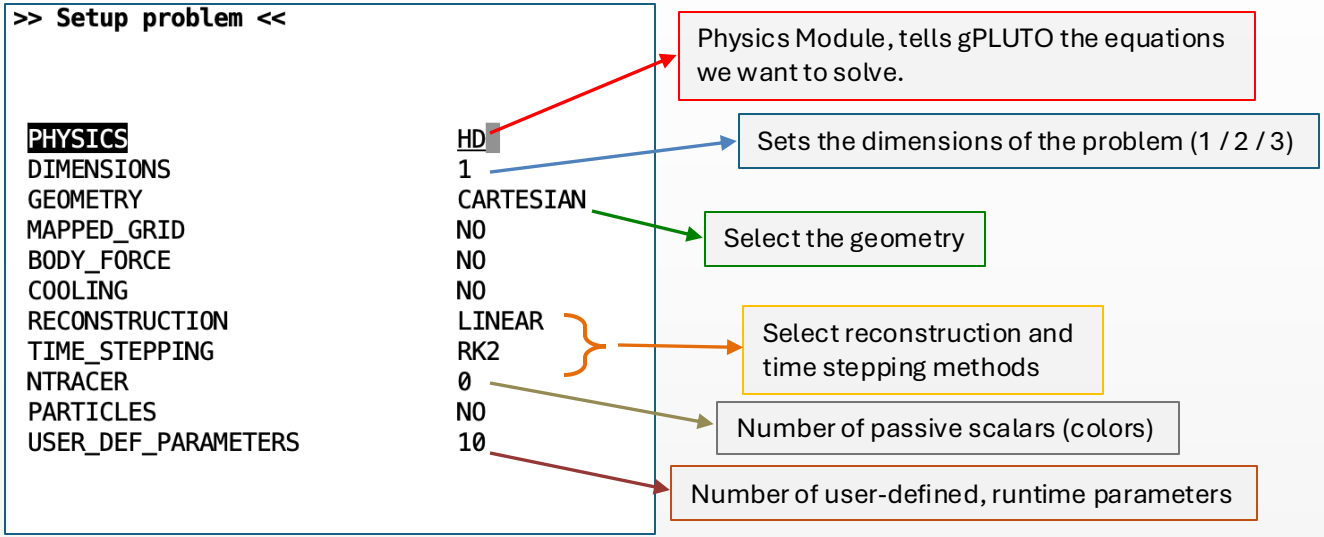
```
>> Python setup (March 2025) <<

Working dir:      /Users/mignone/gPLUTO-dev/Test_Problems/HD/Shock_Tubes
PLUTO dir:        /Users/mignone/gPLUTO-dev/
Enabled Modules: [None]


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

- Press enter under "*Setup problem*"

```
>> Setup problem <<


 PHYSICS                HD
 DIMENSIONS             1
 GEOMETRY               CARTESIAN
 MAPPED_GRID            NO
 BODY_FORCE             NO
 COOLING                NO
 RECONSTRUCTION         LINEAR
 TIME_STEPPING          RK2
 NTRACER                0
 PARTICLES              NO
 USER_DEF_PARAMETERS    10
```

The “Setup problem” menu

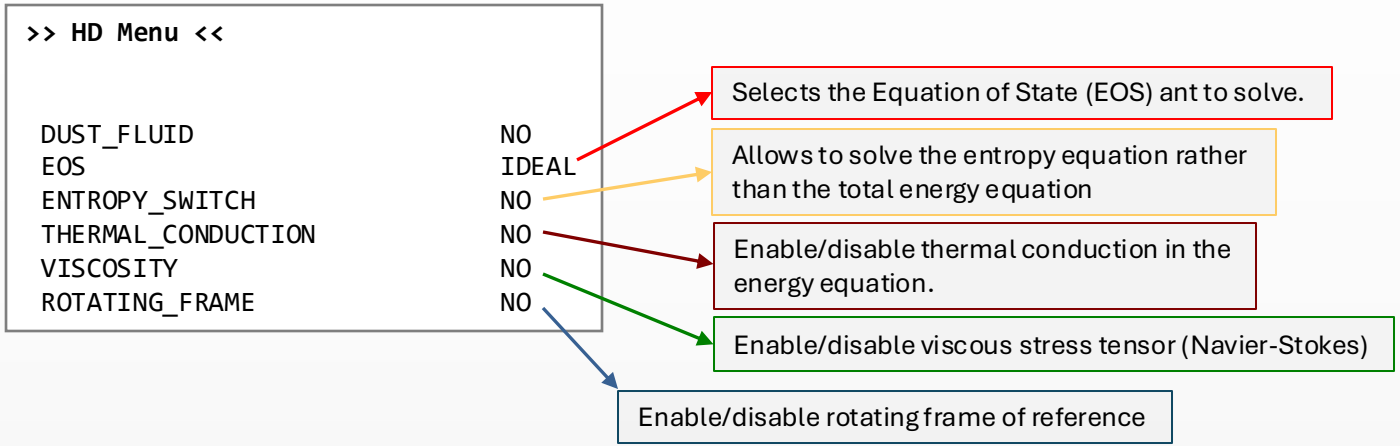
- In a self-explanatory way, the setup menu allows the user to configure the basic features for your problem:



- 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.

The “User-Defined” Parameters Sub-menu

- In the next screen, you will see the runtime parameters read by pluto at runtime:

```
>> User-defined Parameters <<


0               RHO_LEFT
1               VX1_LEFT
2               VX2_LEFT
3               VX3_LEFT
4               PRS_LEFT
5               RHO_RIGHT
6               VX1_RIGHT
7               VX2_RIGHT
8               VX3_RIGHT
9               PRS_RIGHT
```

Used to define the fluid state to the left of the interface.

Used to define the fluid state to the left of the interface.

- Press enter to move to the next screen.

Makefile Menu

- If you do not have a makefile, or once you're back to the main menu, you can select a different `makefile` (→ "*Change makefile*")

```
>> Change makefile <<


linux.g++.defs
linux.mpicxx.acc.defs
linux.mpicxx.defs
linux.nvc++.acc.defs
linux.nvc++.acc.nvtx.defs
linux.nvc++.debug.defs
linux.nvc++.defs
linux.nvc++.nvtx.defs
```

- Configuration files are taken from the `Config/` directory. If you have no idea, simply go with `linux.g++.defs` (for a serial run) or `linux.mpicxx.defs` (for a parallel run).

The “User-Defined” Sub-menu

- After you exit the python menu, you can still edit `definitions.hpp` manually to add some (more advanced) control option. These are described at the end of the userguide.
- For instance:

Enable writing of “`pluto.gp`”, a Gnuplot script file containing grid and variable index information useful for plotting purposes.

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


#define  GNUPLOT_HEADER          YES
#define  LIMITER                 VANLEER_LIM
```

Sets the piecewise linear limiter

- Several other control switches are available (see the `userguide.pdf`)

Specifying Initial Conditions (step #3):

- Initial condition are coded inside `init.cpp` file using the `Init()` function;

- This file is always part of your local working directory.

```
/* ********************************************************************* */
void Init (double *v, double x1, double x2, double x3, RunConfig &run_config)
/*
 *
 ********************************************************************* */
{
  double *inputParam = run_config.inputParam;

  run_config.gamma = 1.4;

  if (x1 < 0.5){
    v[RHO] = inputParam[RHO_LEFT];
    v[VX1] = inputParam[VX1_LEFT];
    v[VX2] = inputParam[VX2_LEFT];
    v[VX3] = inputParam[VX3_LEFT];
    #if EOS == IDEAL
    v[PRS] = inputParam[PRS_LEFT];
    #endif
  }else{
    v[RHO] = inputParam[RHO_RIGHT];
    v[VX1] = inputParam[VX1_RIGHT];
    v[VX2] = inputParam[VX2_RIGHT];
    v[VX3] = inputParam[VX3_RIGHT];
    #if EOS == IDEAL
    v[PRS] = inputParam[PRS_RIGHT];
    #endif
  }
}
```

Runtime Parameters (step #4): `pluto.ini`

- At runtime, *gPLUTO* 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 should be edited manually.

```
[Grid]

X1-grid    0.0    400    1.0
X2-grid    0.0      1    1.0
X3-grid    0.0      1    1.0

[Time]

CFL              0.8
CFL_max_var      1.1
tstop            0.2
first_dt         1.e-4

[Solver]

Solver         roe

[Boundary]

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

[Static Grid Output]

uservar    0
dbl        90.5  -1   single_file
flt        0.02  -1   single_file
vtk        -1.0  -1   single_file
tab        0.02  -1
ppm        -1.0  -1
png        -1.0  -1
log               1
analysis   -1.0  -1

[Parameters]

RHO_LEFT                    1.0
VX1_LEFT                    0.0
VX2_LEFT                    0.0
VX3_LEFT                    0.0
PRS_LEFT                    1.0
RHO_RIGHT                   0.125
VX1_RIGHT                   0.0
VX2_RIGHT                   0.0
VX3_RIGHT                   0.0
PRS_RIGHT                   0.1
```

Compiling and Running

- *gPLUTO* can now be compiled by typing `make` at the command prompt:

```
> make -j     # use make -j for parallel compilation (faster)
```

- Run the code by typing: `> ./pluto`

```
> ./pluto > out.log # redirect std output to "out.log"
```

- Several command line options are possible. Some examples are:

Command Line Options	Action
`-xres n1`	Set the grid resolution in the x1 direction to n1 zones, overriding 'pluto.ini'. Grid resolution in the other direction modified as well to preserve cell aspect ratio.
`-maxsteps n`	Stop computations after 'n' steps.
`-no-write`	Suppress output to disk.

Output log

- Upon running:

```
...
> Memory allocation
> Assigning initial conditions (Startup) ...
> Writing file #0 (dbl) to disk...
> Writing file #0 (flt) to disk...
> Writing file #0 (tab) to disk...

> Starting computation
> nstep = 0; t = 0.000000e+00; dt = 1.000000e-04;    0.0 %
  Max(MACH) =  0.140992
  Max(vc)   =  1.000000e+00   1.303189e-01   0.000000e+00   0.000000e+00  1.000000e+00
  Min(vc)   =  1.250000e-01   0.000000e+00   0.000000e+00   0.000000e+00  1.000000e-01
> nstep = 1; t = 1.000000e-04; dt = 1.100000e-04;    0.0 %
  Max(MACH) =  0.262296
  Max(vc)   =  1.000000e+00   2.480148e-01   0.000000e+00   0.000000e+00  1.000000e+00
  Min(vc)   =  1.250000e-01   0.000000e+00   0.000000e+00   0.000000e+00  1.000000e-01

...

> nstep = 233; t = 1.996363e-01; dt = 3.636627e-04;   99.8 %
  Max(MACH) =  1.100734
  Max(vc)   =  1.000000e+00   9.280666e-01   0.000000e+00   0.000000e+00  1.000000e+00
  Min(vc)   =  1.250000e-01   0.000000e+00   0.000000e+00   0.000000e+00  1.000000e-01
> Writing file #1 (dbl) to disk...
> Writing file #10 (flt) to disk...
> Writing file #10 (tab) to disk...

> Total allocated memory    0.10 Mb
> Elapsed time (sec):         0.031060
> Done.
```

Current time

Current time step

Maximum Mach number at each step

Everything's fine !

Data Visualization

- Different visualization packages may be used to display *gPLUTO* data-files. Some popular packages are reported below:

	Gnuplot	**IDL**	**PyPLUTO**	**VisIt**	**Paraview**
Free/Open Source	Yes	No	Yes	Yes	Yes
File support	ASCII (`.tab`) Binary	All	All	VTK, HDF5	VTK, HDF5
Data Analysis	No	Yes	Yes	No	No

- If you're not familiar with any of these packages, you can use gnuplot since it's practically available by default on any Linux platform.
- VisIt is also recommended for visualization of `.vtk` files.

Visualization with Gnuplot

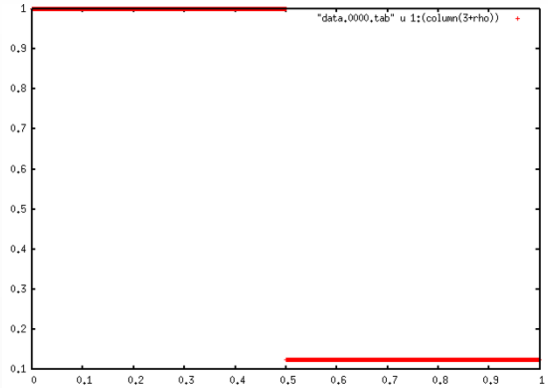
- ASCII datafile (`.tab`) can be easily visualized with GNUPLOT, using the plot command, e.g.

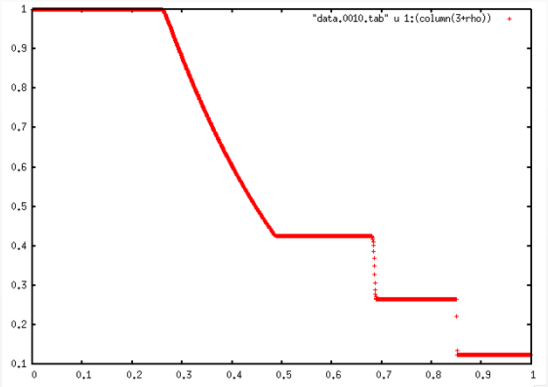
```
gnuplot> plot “data.0000.tab” using 1:3
```

- This will produce a 1D plot (x,y) of the initial condition (‘`0000`’).
- The keyword ‘`using`’ specify that the x-coordinate is taken from the 1st column and the y-coordinate from the 3rd column (= density);
- To plot the solution at the 10th output, type

```
gnuplot> plot “data.0010.tab” using 1:3
```

- Other variables may be plotted as well (4,5,6 = velocity, 7 = gas pressure);





Visualization with Gnuplot

- Variable names (together with grid information) are written by *gPLUTO* to a special file (`pluto.gp`) at runtime;
- You can load them as follows:

```
gnuplot> dtype='tab'                                  # select the data type
gnuplot> load "pluto.gp"                              # load some info from the simulation
gnuplot> plot "data.0005.tab" u 1:(column(vx1))       # plot velocity
```

- For the shock tube, variable are named as: `rho` (density), `vx1` (x-velocity) and `prs` (gas pressure).

- A simple animation script can also be used to produce an animation[1],

```
gnuplot> load "shock_tubes.gp"
```

[1]Set the GNUPLOT_LIB environment variable in your shell using
`export GNUPLOT_LIB=$PLUTO_DIR/Tools/Gnuplot`

Visualization with PyPLUTO

To install PyPLUTO, Python ≥3.10 is required.

- The best practice is to create a separate Python environment:

```
> python(3) -m venv workshop
> source workshop/bin/activate
```

- or a conda environment

```
> conda create -n workshop
> conda activate workshop
```

- To install PyPLUTO:

```
> cd $PLUTO_DIR/Tools/PyPLUTO
> pip install ./
```

- This should install all the necessary packages (numpy, matplotlib, scipy, pandas, etc...).

Visualization with PyPLUTO

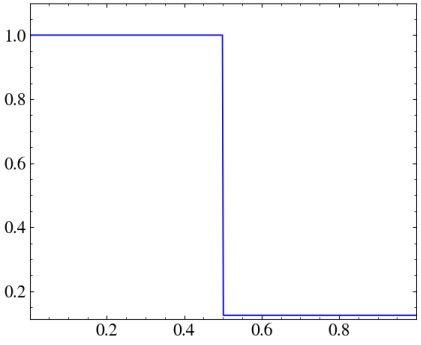
- All output files can be loaded with PyPLUTO:

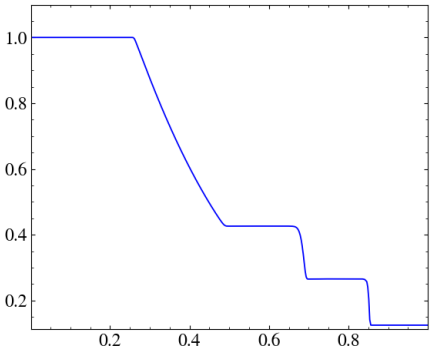
```
> import pyPLUTO as pp
> D = pp.Load(0)
> I = pp.Image().plot(D.x1,D.rho)
> pp.show(block = False)
```

- This will produce a 1D plot (x,y) of the initial condition (‘`0000`’).
- The variables are loaded as attributes of the Load class (including userdef variables).
- To plot the solution at the last output, type

```
> D = pp.Load()
```

- Other variables may be plotted as well (their name is the same of the one in files `format.out`);

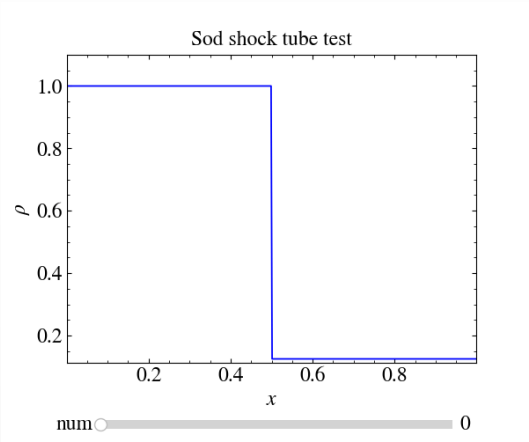




Visualization with PyPLUTO

- You can plot multiple variables or customize the axes as follows:

```
> import pyPLUTO as pp
> D = pp.Load()
> I = pp.Image()
> I.plot(D.x1, D.rho, label = ‘rho’, legpos = 0)
> I.plot(D.x1, D.prs, xtitle = ‘x’)
> pp,show()
```



- For a quick documentation some useful ways are:

```
> help(pp.Image().plot)
> print(pp.Image())
```

- A simple animation script can also be used to produce an animation,

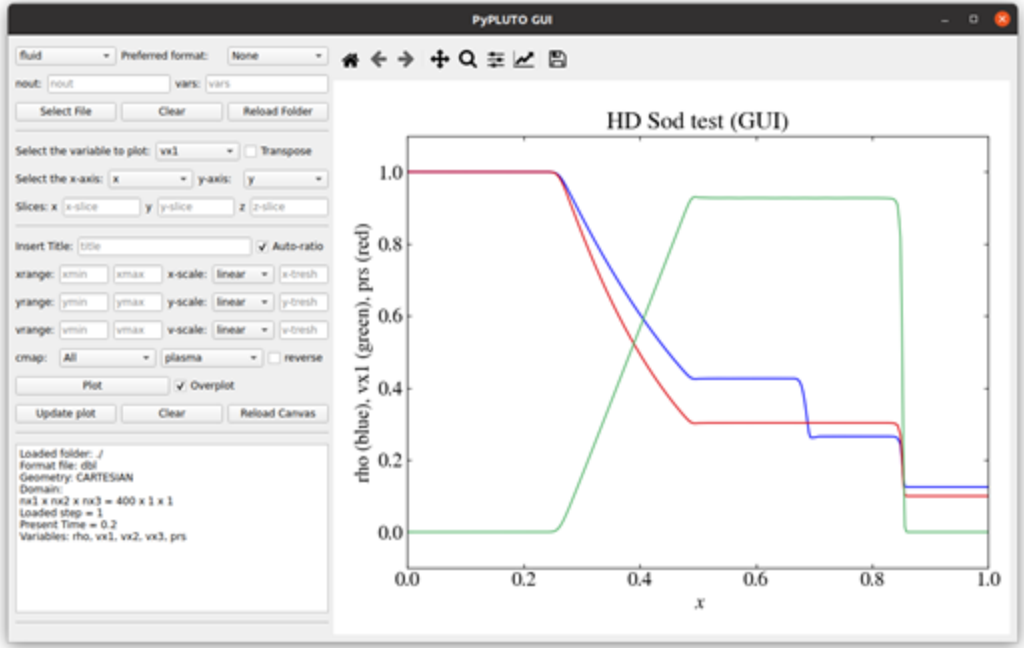
```
python ST_animation.py
```

Visualization with PyPLUTO

- A PyPLUTO subpackage consists in a GUI, which can be opened by typing:

```
> pypluto-gui
```

on the terminal.



Testing Various Configurations

- Implement the condition for an isolated contact wave. Compare the Roe (or HLLC) Riemann solver with HLL.

- Change the resolution;

SESSION #2:
2D MHD BLAST WAVE PROBLEM

Setting up the Problem

- Change directory to `Test_Problems/MHD/Blast_Simple`, copy the 1st configuration file and run the Python script:

```
> cp definitions_01.hpp definitions.hpp
> cp pluto_01.ini pluto.ini
> python3 $PLUTO_DIR/setup.py
```

- The main setup menu should look like

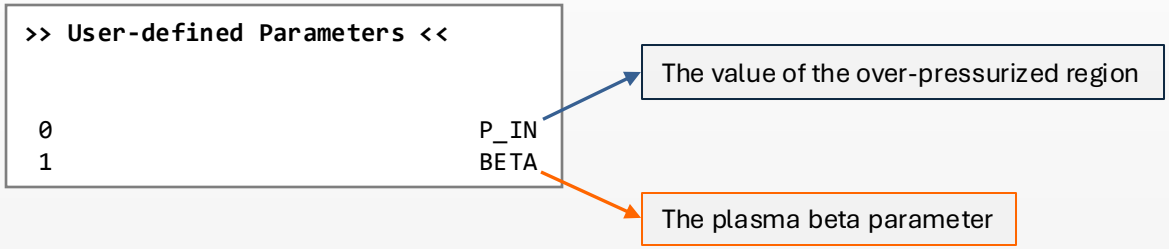
```
>> Setup problem <<

 PHYSICS                 MHD
 DIMENSIONS              2
 GEOMETRY                CARTESIAN
 BODY_FORCE              NO
 COOLING                 NO
 RECONSTRUCTION          LINEAR
 TIME_STEPPING           RK2
 NTRACER                 0
 PARTICLES               NO
 USER_DEF_PARAMETERS     2
```

We are now using the *MHD module*, in *2 dimensions* and, *2 user-defined parameters*.

Setting the Parameters

- We can use the constrained transport method to control the divergence of B and define the name of 2 parameters that will be used later in our problem definition.

- Just scroll down with your arrow keys and check that the your userdef parameter names are already set to

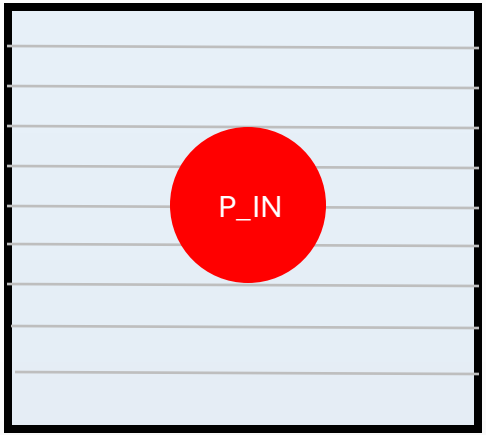


Setting Initial & Boundary Conditions

- We consider a 2D Cartesian box, filled with uniform density fluid and horizontal magnetic field. An over-pressurized region is set inside a circle of radius r_0:

- For simplicity we choose a zero-gradient boundary conditions in all directions.

- The over-pressurized region drives a blast wave delimited by an outer fast forward shock propagating (nearly) radially while magnetic field lines pile up behind the shock thus building a region of higher magnetic pressure.



Specifying Initial Conditions

- Initial conditions are coded inside `init.cpp` file using the `Init()` function;
- This file is always in your local working directory;
- Parameters are included using the array `run_config->inputParam[<name>];`
- The value of these parameters is read at runtime from `pluto.ini`.

```
void Init (double *v, double x1, double x2, double x3, RunConfig &run_config)
{
  double *inputParam = run_config.inputParam;
  double r, theta, phi, B0;

  r = DIM_EXPAND(x1*x1, + x2*x2, + x3*x3);
  r = sqrt(r);

  v[RHO] = 1.0;
  v[VX1] = 0.0;
  v[VX2] = 0.0;
  v[VX3] = 0.0;
  v[PRS] = 1.0/run_config.gamma;
  B0     = sqrt(2.0*v[PRS]/inputParam[BETA]);

  if (r <= 0.1) v[PRS] = inputParam[P_IN]; /* Change pressure internally */

  v[BX1] = B0;
  v[BX2] = 0.0;
  v[BX3] = 0.0;

  v[AX1] = 0.0;
  v[AX2] =  v[BX3]*x1;
  v[AX3] = -v[BX2]*x1 + v[BX1]*x2;
}
```

Runtime Parameters: `pluto.ini`

- At runtime, `gPLUTO` reads the `pluto.ini` file which controls several options use by the code at runtime, such as grid generation, CFL number, boundary conditions, output type and so forth.

- Make sure you're writing using your preferred data format.

- Problem-parameters are set at the end.

```
[Grid]

X1-grid    -0.5    192    0.5
X2-grid    -0.5    192    0.5
X3-grid    -0.5      1    0.5

[Time]

CFL              0.4
CFL_max_var      1.1
tstop            5.0e-2
first_dt         1.e-6

[Solver]

Solver         hll

[Boundary]

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

[Static Grid Output]

uservar    0
dbl       -2.5e-3  -1   single_file
flt        5.0e-3  -1   single_file
vtk       -2.5e-3  -1   single_file
dbl.h5    -1.0     -1
tab        5.0e-3  -1
ppm       -1.0     -1
png       -1.0     -1
log        10
analysis  -1.0     -1

[Chombo HDF5 output]

Checkpoint_interval  -1.0  0
Plot_interval         1.0  0

[Parameters]

P_IN                      1.e2
BETA                      0.1
```

Compiling and Running

- Compile by typing "`make`" at the terminal

- Run by typing

```
> ./pluto   # Serial run using (using e.g., linux.g++.defs)
```

or

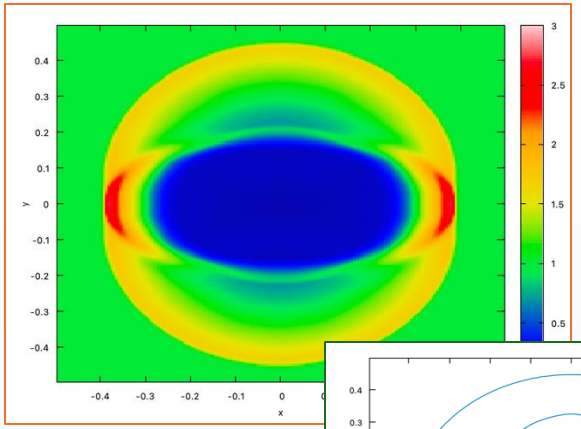
```
> mpirun -np <n> ./pluto   # Parallel run (linux.mpicxx.defs must have been selected)
```

- Depending on your processor speed, the code may take few minutes to run.

Visualization with Gnuplot

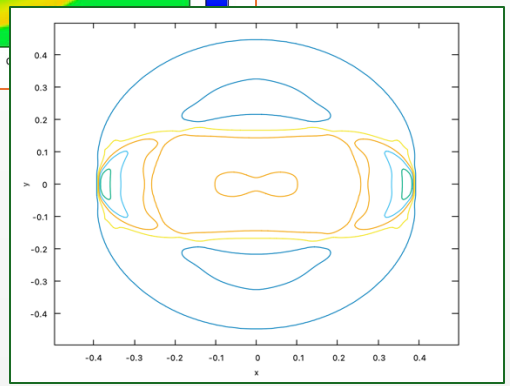
- 2D visualization can also be done with gnuplot using the `splot` command.
- To produce a coloured map:

```
gnuplot> load “pluto.gp”
gnuplot> set pm3d map
gnuplot> splot "data.0010.tab" u 1:2:(column(rho))
```



- To draw contour levels,

```
gnuplot> load “pluto.gp”
gnuplot> set contour base
gnuplot> set view map
gnuplot> unset surface            # Disable surface plot
gnuplot> set style data lines     # Use lines instead of
points
gnuplot> splot "data.0010.tab" u 1:2:(column(prs))
```

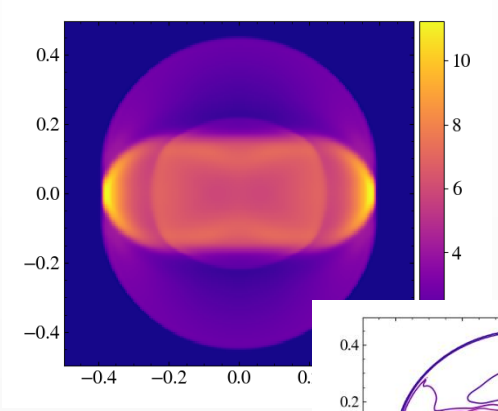


- The script `mhd_blast.gp` can be used to produce an animation.

Visualization with PyPLUTO

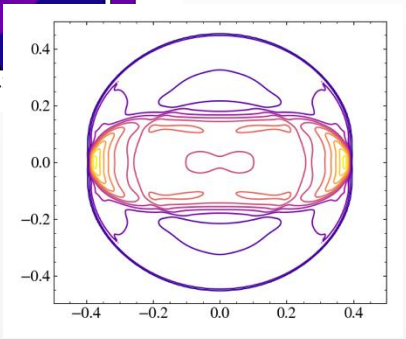
- 2D visualization can also be done with PyPLUTO using the `display` command.
- To produce a coloured map:

```
> import pyPLUTO as pp
> D = pp.Load()
> pp.Image().display(D.prs, x1 = D.x1, x2 = D.x2, cpos = 'right')
> pp.show()
```



- To draw contour levels,

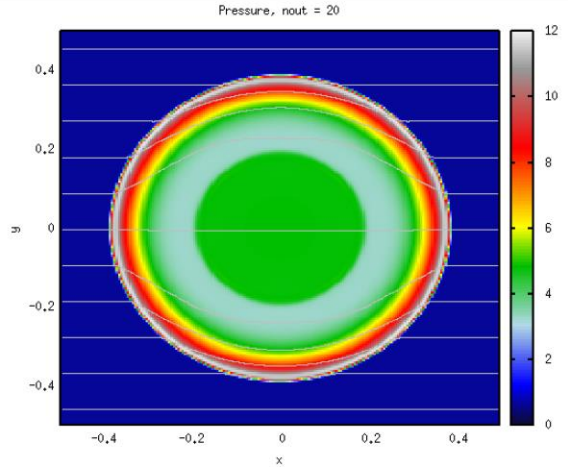
```
> import pyPLUTO as pp
> D = pp.Load()
> pp.Image().contour(D.prs, x1 = D.x1, x2 = D.x2, cmap =
'plasma', levels = 10)
> pp.show()
```

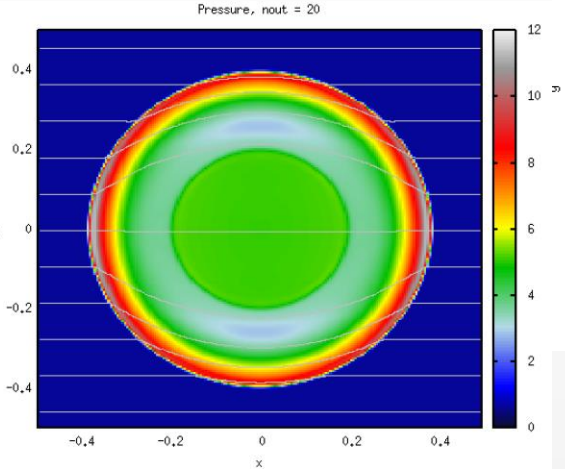


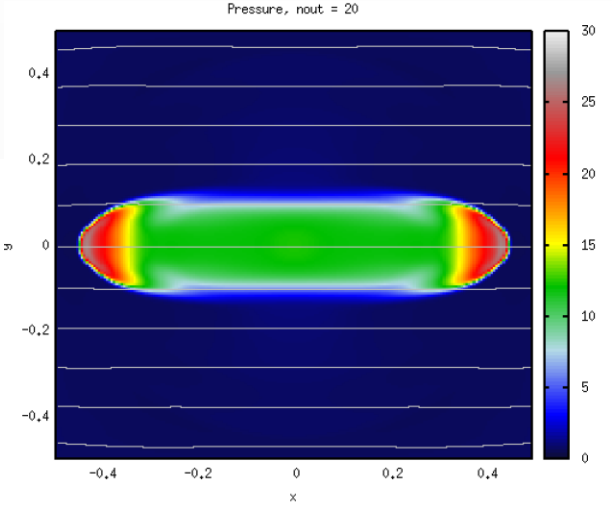
- The script `BW_video.py` can be used to produce an animation.

Parameter Study

- Try different computations by decreasing the plasma β parameter (100, 1, 0.01). What happens ? Explain.





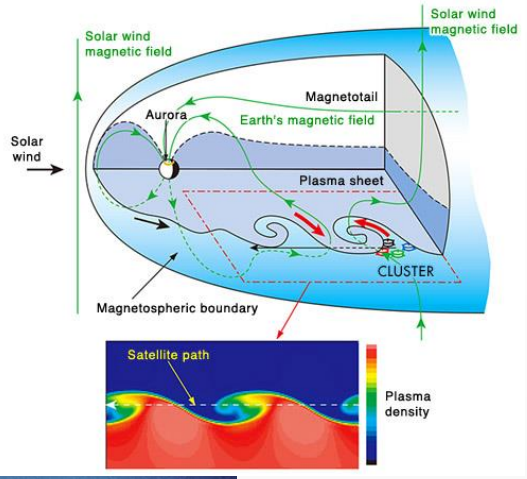


SESSION #3:
THE KELVIN-HELMHOLTZ INSTABILITY (KHI)

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 (astrophysics), etc…



Equations & Initial Configuration

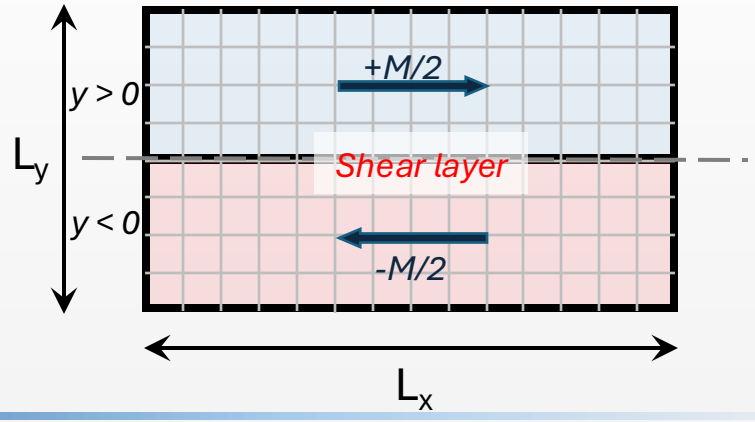
- The instability may be analyzed using the compressible Euler equations

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

- We consider an initial uniform background with constant density and pressure ($\rho=1$, $c_s=1$) and velocity shear given by

$$\mathbf{u} = \frac{M}{2} \tanh\left(\frac{y}{a}\right) \hat{\mathbf{e}}_x$$

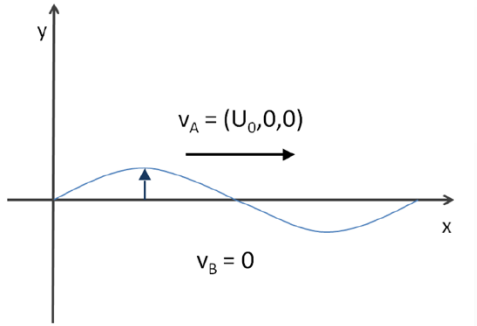
- We apply periodicity in the x-direction and reflecting b.c. in the y-direction.



Linear Theory: the vortex-sheet case

- The *vortex-sheet* case is obtained when a → 0:

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



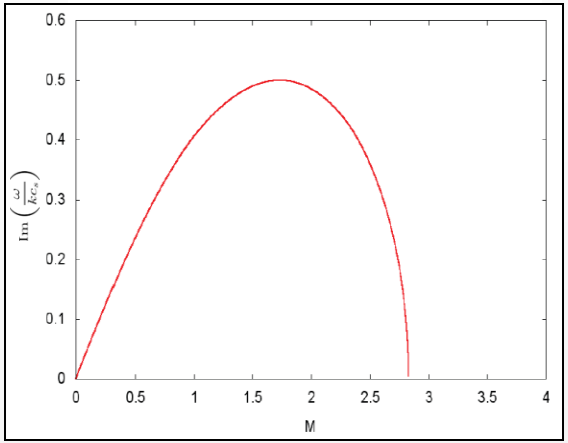
- Linearization of the equations can be carried out for small perturbations,

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

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

- The linearization process leads, for a=0 to the following analytical dispersion relation

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



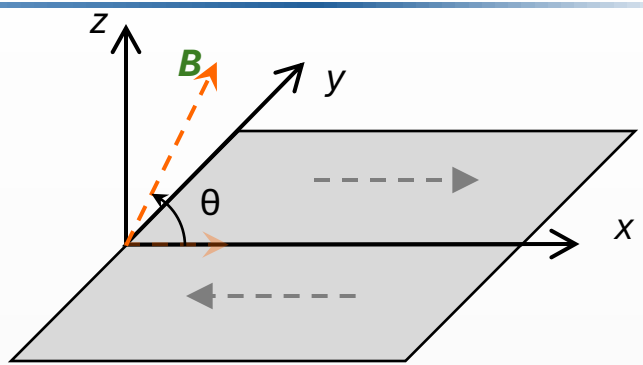
- A complex value of ω(k) indicates an instability.

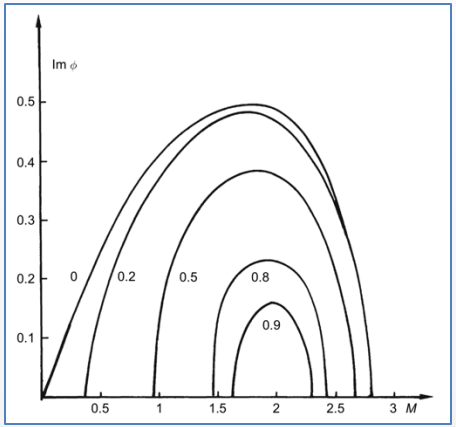
Linear Theory: vortex-sheet in MHD

- When a magnetic field is included, the linearization leads to a 10th degree polynomial;

- The effect of B on the development of the KHI strictly depends on its geometry:

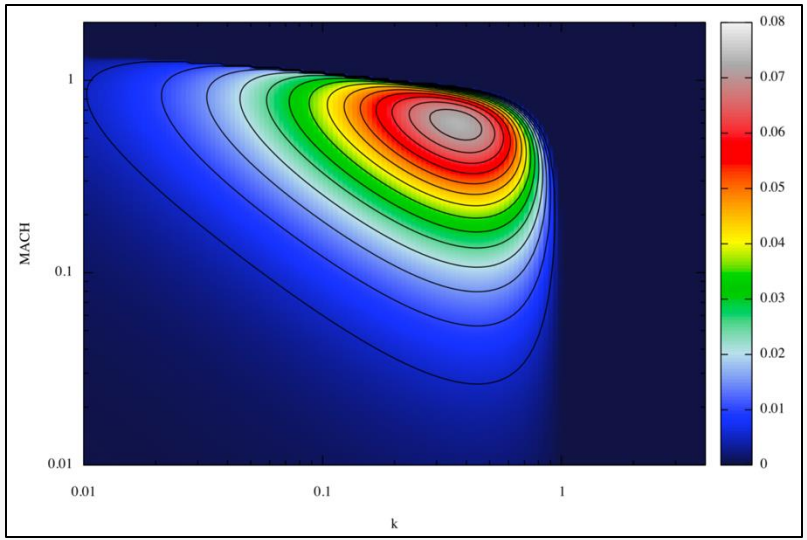
  - θ = π/2: when **v** and **B** are perpendicular the features of the instability are basically unmodified if we redefine the Mach number as $M = v/\mathrm{sqrt}(v_A^2 + c_s^2)$;
  - θ = 0: when **v** and **B** are parallel, the magnetic field has a stabilizing effect: for $M < 2v_A/c_s$ the flow is stable, while the supersonic cut off decreases from $M = \mathrm{sqrt}(8) \rightarrow 2$ and for $v_A/c_s > 1$ the two limits coincide: highly magnetized flows are always stable against the KHI.





Linear Theory: the finite velocity shear

- When the shear width a≠0, the eigenvalue problem has to be solved numerically.
- For $u_x(y) = M/2\ \tanh(y/a)$ the region of instability [defined as $-\mathrm{Im}(\omega a/c_s)$] in the (k, M) plane is found to be

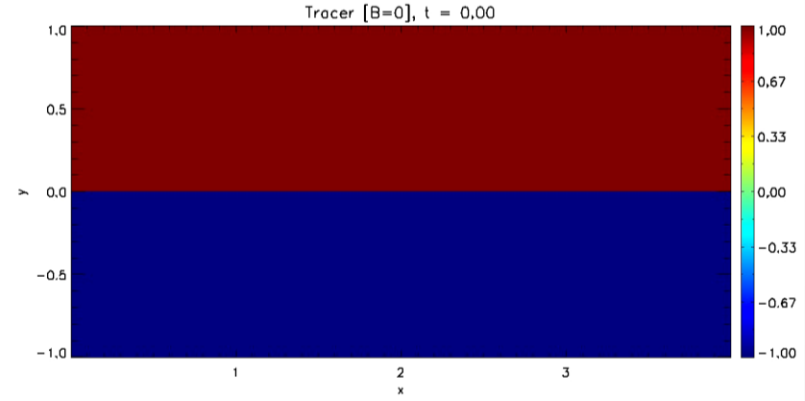


- A cut-off is found around ka ≈ 1.
- Flow with M > 1 tend to become stable

- For M = 1 (B=0) the peak growth rate is $\mathrm{Im}(\omega) \approx 0.071\ c_s/a$, $ka \approx 0.396$.

KHI: 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.

Setting up the Problem

- Change directory to `Test_Problems/MHD/Kelvin_Helmholtz` and run the Python script

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

- The main setup menu should look like

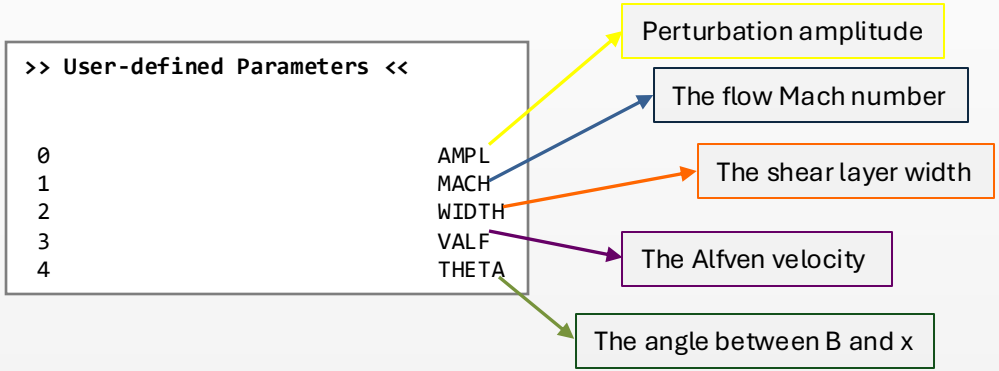
```
>> Setup problem <<

 PHYSICS                 MHD
 DIMENSIONS              2
 GEOMETRY                CARTESIAN
 BODY_FORCE              NO
 COOLING                 NO
 RECONSTRUCTION          LINEAR
 TIME_STEPPING           RK2
 NTRACER                 1
 USER_DEF_PARAMETERS     5
```

- We are now using the *MHD module*, in *2 dimensions* with *5 user-defined parameters*.

Setting the Parameters

- We can use the constrained transport method to control the divergence of B, and define the name of 4 parameters that will be used later in our problem definition.

- Just scroll down with your arrow keys and check that the your userdef parameter names are already set to



Specifying Initial Conditions:

- Initial condition must be coded inside `init.cpp` file using the `Init()` function or the `InitDomain()` function.

- We'll use the latter for this case, to show how to assign initial conditions by directly performing the loop over the computational domain.

```
/* ************************************************************************
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, nv;
  int ibeg = grid->lbeg[IDIR], iend = grid->lend[IDIR];
  int jbeg = grid->lbeg[JDIR], jend = grid->lend[JDIR];
  int kbeg = grid->lbeg[KDIR], kend = grid->lend[KDIR];

  RunConfig &run_config = *(d->run_config);

  double *inputParam = run_config.inputParam;
  double B0, rnd, scrh;
  double x,y,z;
  double *xC   = grid->xC[IDIR], *yC = grid->xC[JDIR], *zC = grid->xC[K
  double *xL   = grid->xL[IDIR], *yL = grid->xL[JDIR], *zL = grid->xL[K
  double Ly    = grid->xend_glob[JDIR] - grid->xbeg_glob[JDIR];
  double M     = inputParam[MACH];
  double eps   = inputParam[AMPL]*M;  /* Perturbation amplitude */
  double a     = inputParam[WIDTH];
  double vA    = inputParam[VALF];
  double theta = inputParam[THETA]/180.0*CONST_PI;
  double v[256], u[256];

  RBox   box, box_tot;
```

Runtime Parameters: `pluto.ini`

- At runtime, gPLUTO 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.

- This file can be edited manually.

- We set `a = 0.02, M = 1, θ = 0` and consider the two (or more) cases:

  1. HD case (`VALF = 0`)
  2. MHD case (`VALF = 0.3`)

```
[Grid]

X1-grid    0.0   128    1.0
X2-grid   -1.0   256    1.0
X3-grid    0.0     1    1.0

[Time]

CFL              0.4
CFL_max_var      1.1
tstop            20.0
first_dt         1.e-4

[Solver]

Solver          roe

[Boundary]

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

[Static Grid Output]

uservar    0
dbl        1.0  -1    single_file
flt        0.2  -1    single_file
vtk       -0.1  -1    single_file
tab       -0.1  -1
ppm       -1.0  -1
png       -1.0  -1
log        10
analysis   0.1  -1

[Parameters]

AMPL                        1.e-6
MACH                        1.0
WIDTH                       0.02
VALF                        0.0
THETA                       0.0
```

Runtime Analysis: The Analysis() function

- gPLUTO can perform on the fly data reduction through the function Analysis() in your init.cpp file.
- This function is automatically called by the code to produce a multi-column ASCII data file of the form:

```
# [0]           [1]            [2]            [3]
# t             max-min        max(By^2)      pm
0.000000e+00  9.975486e-07  0.000000e+00  0.000000e+00
9.928043e-02  2.572954e-07  0.000000e+00  0.000000e+00
1.992804e-01  1.941973e-07  0.000000e+00  0.000000e+00
2.992804e-01  1.848784e-07  0.000000e+00  0.000000e+00
3.992804e-01  1.703704e-07  0.000000e+00  0.000000e+00
4.992804e-01  1.779959e-07  0.000000e+00  0.000000e+00
5.992804e-01  2.118356e-07  0.000000e+00  0.000000e+00
6.992804e-01  2.698428e-07  0.000000e+00  0.000000e+00
7.992804e-01  3.626834e-07  0.000000e+00  0.000000e+00
8.992804e-01  4.847207e-07  0.000000e+00  0.000000e+00
...
```

- The second column can be used to measure the growth rate.

Compiling and Running

- Compile by typing `make` at the terminal
- Run by typing either

```
> ./pluto   # Serial run (Linux.gcc.defs must have been selected)
```

or

```
> mpirun -np <n> ./pluto # Parallel run
```

- Depending on your processor speed, the code may take few minutes to run.

Visualization with IDL

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

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

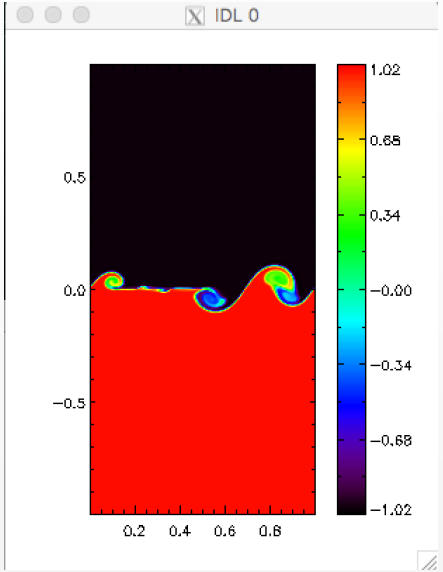
- Launch IDL,

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

Load output data #25 in vtk format

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

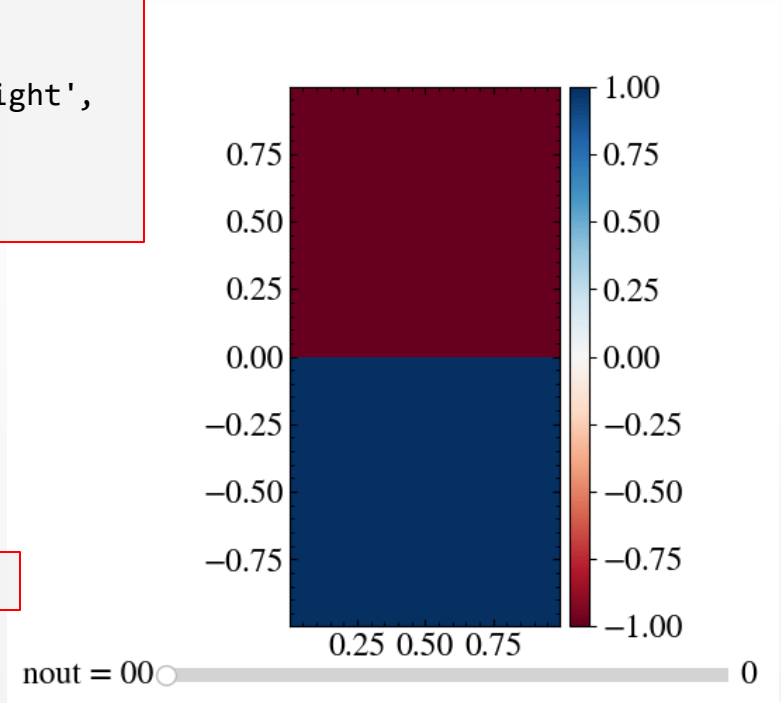
Display, the tracer field adding Coordinates and vertical colorbar.



Visualization with PyPLUTO

- PyPLUTO allows for simple interactive plots with the ‘interactive’ method:

```
> D = pp.Load('all', vars = 'tr1')
> I = pp.Image()
> I.interactive(D.tr1, x1 = D.x1, x2 = D.x2, cpos = 'right',
> … aspect = 'equal', cmap = ‘RdBu’)
> …
> I.animate(interval = 100)
```



- A gif can be created by providing a file name:

```
> I.animate(‘KHI.gif’, interval = 100)
```

- To load `.dat` files (e.g., the khi.dat):

```
> res = pp.Load(None).read_file('khi.dat', skip = 1)
```

- where res is a Dictionary.

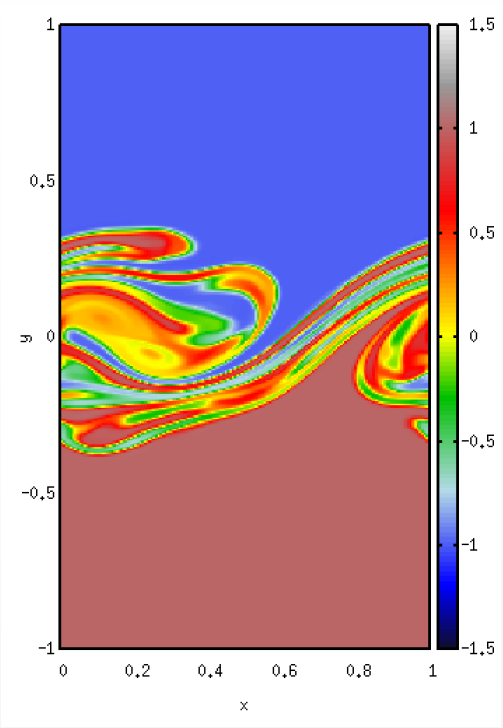
Analysis of the KHI

- The script “`khi.gp`” can be used to produce an animation with Gnuplot.

- Study the evolution of the instability without magnetic field (vA = 0).
  - Can you identify the linear phase and the transition to the subsequent nonlinear evolution ?
  - Can you measure the growth rate of the instability ?

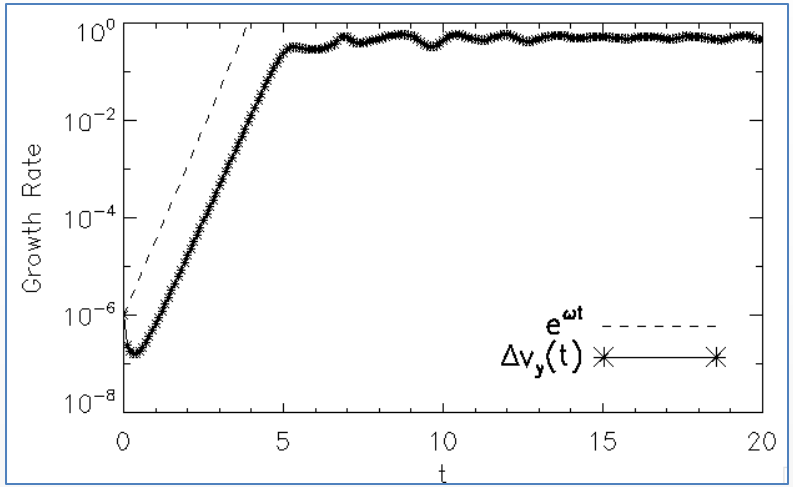
- Increase the magnetic field strength (0 < vA < 1) and confirm that the instability becomes suppressed as vA → cs in the parallel case (θ = 0).

- Change the magnetic field geometry by considering the perpendicular case (θ = π/2). What do you see ?



Comparing Growth Rates

- Plot the analytical growth rate ($\propto e^{\omega t}$) vs. the numerical one (2nd column of "`khi.dat`").
- Here $\mathrm{Im}(\omega) \approx 0.071\ c_s/a$ is the analytical growth rate.



- Try to lower the resolution or switch to a more diffusive Riemann solver. What do you observe ?

SESSION #4: MHD JET PROPAGATION

Initial Configuration

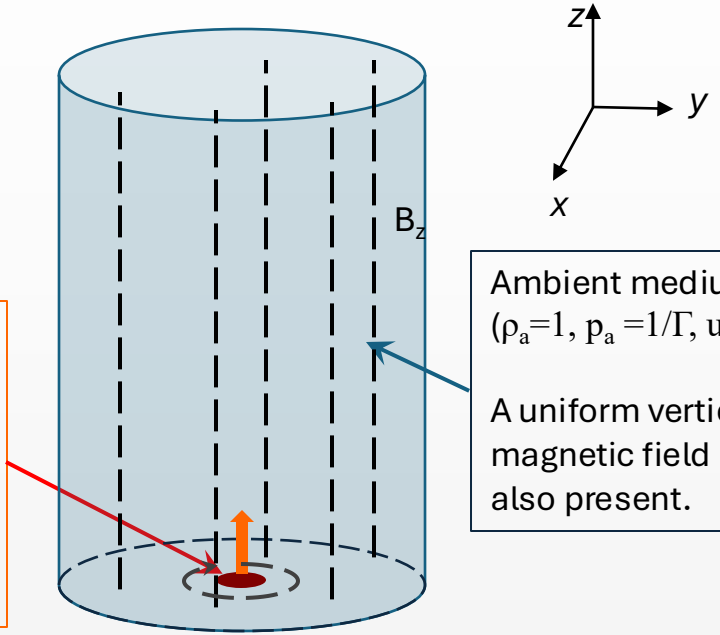
- Jet propagation is modeled through the injection of supersonic matter at a small circular nozzle in cylindrical coordinates (x1 = R, x2 = φ, x3 = z).

$$\begin{cases} \sigma_z = \dfrac{B_z^2}{2p_a} \\ \sigma_\phi = \dfrac{< B_\phi^2 >}{2p_a} \end{cases} \Longrightarrow \begin{cases} B_z^2 = 2\sigma_z p_a \\ B_m^2 = \dfrac{2\sigma_\phi}{a^2(1/2 - 2\log a)} p_a \end{cases}$$

The jet enters at the lower-z (=x3) boundary from a circular nozzle of radius R_j=1. Jet in pressure equilibrium (p_j=p_a), with density contrast:

$\eta=\rho_j/\rho_a$, velocity $u_j >> c_s$.

It carries both a poloidal (B_z) and toroidal (B_ϕ) magnetic field



Ambient medium (ρ_a=1, p_a =1/Γ, u_a=0).

A uniform vertical magnetic field B_z is also present.

Initial Configuration

- Change directory to `Test_Problems/MHD/Jet` and run the Python script

- This problem runs either in 2D (`definitions_01.hpp` and `pluto_01.ini`) or 3D Cartesian coordinates (`definitions_10.hpp` and `pluto_10.ini`).

- Let's begin with a 2D run:

```
> cp definitions_01.hpp definitions.hpp
> cp pluto_01.ini  pluto.ini
> python $PLUTO_DIR/setup.py
```

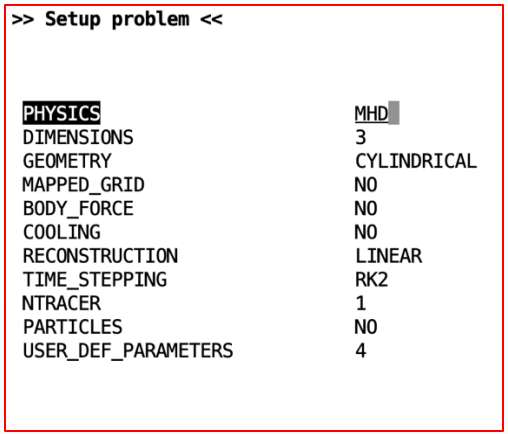
Setting up the Problem

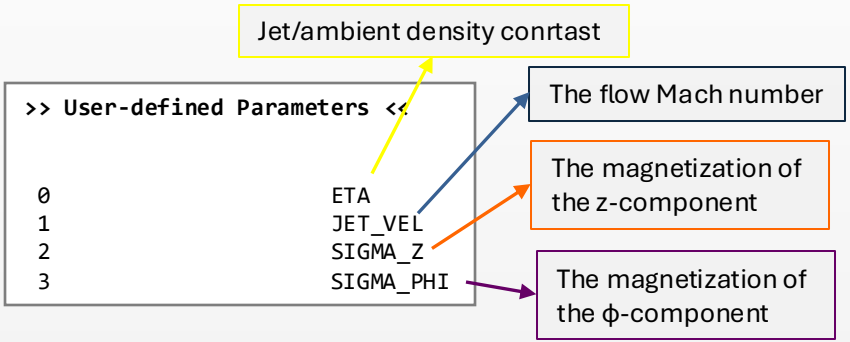
- The main setup menu should look like

- We are now using the *MHD module*, *3 dimensions** R, φ, z with *4 user-defined parameters*.

- We can use the constrained transport method to control the divergence of B and define the name of 4 parameters that will be used later in our problem definition.

- Just scroll down with your arrow keys and check that your user-def parameter names are already set to





* The φ coordinate will be set cyclic later.

Specifying Initial Conditions:

- The `Init()` function is used to prescribe the ambient values only:

```
/* ********************************************************************* */
/*!
 *  Define ambient values and units.
 ********************************************************************* */
void Init (double *v, double x1, double x2, double x3, RunConfig &run_config)
{
  double R = Radius(x1,x2,x3);

  // Set c.g.s units & floor values

  run_config.unit_density  = CONST_amu*1.e3;
  run_config.unit_length   = 2.5e15;
  run_config.unit_velocity = 1.e5;
  #if (GEOMETRY == CARTESIAN) && (DIMENSIONS == 3)
  run_config.small_prs     = 1.e-3;
  #endif

  // Assign initial conditions

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

  // Ambient magnetic field

  #if PHYSICS == MHD
  v[BX1] = 0.0;
  v[BX2] = 0.0;
  v[BX3] = sqrt(2.0*run_config.inputParam[SIGMA_Z]*pa);

  #if (GEOMETRY == CARTESIAN) && (DIMENSIONS == 3)
  v[AX1] = 0.0;
  v[AX2] = v[BX3]*x2;
  v[AX3] = 0.0;
  #elif GEOMETRY == CYLINDRICAL
  v[AX1] = 0.0;
  v[AX2] = 0.5*R*v[BX3];
  v[AX3] = 0.0;
  #endif

  #endif
  v[TRC] = 0.0;
}
```

- Supersonic injection occurs as a boundary condition using the `UserDefBoundary()` function:

- The function `GetJetValues()` sets the actual jet parameters

```
void UserDefBoundary (const Data *d, RBox *box, int side, Grid *grid)
{
  int  i, j, k, nv;
  int  ibeg = grid->lbeg[IDIR], iend = grid->lend[IDIR];
  int  jbeg = grid->lbeg[JDIR], jend = grid->lend[JDIR];
  int  kbeg = grid->lbeg[KDIR], kend = grid->lend[KDIR];

  double *x1 = grid->xC[IDIR], *x1s = grid->xL[IDIR];
  double *x2 = grid->xC[JDIR], *x2s = grid->xL[JDIR];
  double *x3 = grid->xC[KDIR], *x3s = grid->xL[KDIR];
  double time = d->Dts->time;

  RunConfig &run_config = *(d->run_config);

  if (side == X3_BEG){

    // 1. Assign cell-centered values

    #pragma acc parallel loop collapse(3) present(box, d)
    BOX_LOOP(box,i,j,k){
      double vjet[16], vout[16];
      double R = Radius(x1[i], x2[j], x3[k]);

      // 1a. Store jet values

      GetJetValues (x1[i],x2[j],x3[k], vjet, run_config);
      #if (GEOMETRY == CARTESIAN) && (DIMENSIONS == 3)
      JetPerturbation (x1[i], x2[j], x3[k], vjet, run_config);
      #endif

      // 1b. Copy and reflect ambient medium values

      #pragma acc loop seq
      NVAR_LOOP(nv) vout[nv] = d->Vc(i,j,2*kbeg-k-1,nv);
      vout[VX3] *= -1.0;
      #if PHYSICS == MHD
      vout[BX1] *= -1.0;
      vout[BX2] *= -1.0;
      #endif
```

Runtime Parameters: `pluto.ini`

- The `pluto.ini` is read, as usual, at runtime to set grid dimensions, boundary conditions, output type and so forth.

- This file can be edited manually:

  - Change the final time to `tstop = 1.8`
  - Consider three cases with $u_j/c_{sa} = 20$, $\sigma_z = 0$:

    Case #1: Ballistic jet: $\eta = 10$, $\sigma_\varphi = 0.01$;
    Case #2: Under-dense weakly magnetized, $\eta = 0.05$, $\sigma_\varphi = 0.01$;
    Case #3: Under-dense, strongly magnetized $\eta = 0.05$, $\sigma_\varphi = 10$.

- Note that an axisymmetric boundary is prescribed at `R = 0` and a user-defined boundary holds at `z = 0`.

```
[Grid]

X1-grid    0.0   160   10.0
X2-grid    0.0     1    1.0
X3-grid    0.0   480   30.0

[Time]

CFL              0.4
CFL_max_var      1.1
tstop            1.8
first_dt         3.e-7

[Solver]

Solver         hllc

[Boundary]

X1-beg       axisymmetric
X1-end       outflow
X2-beg       outflow
X2-end       outflow
X3-beg       userdef
X3-end       outflow
```

Compiling and Running

- Compile by typing `make` at the terminal
- Run by typing either

```
> ./pluto   # Serial run (Linux.gcc.defs must have been selected)
```

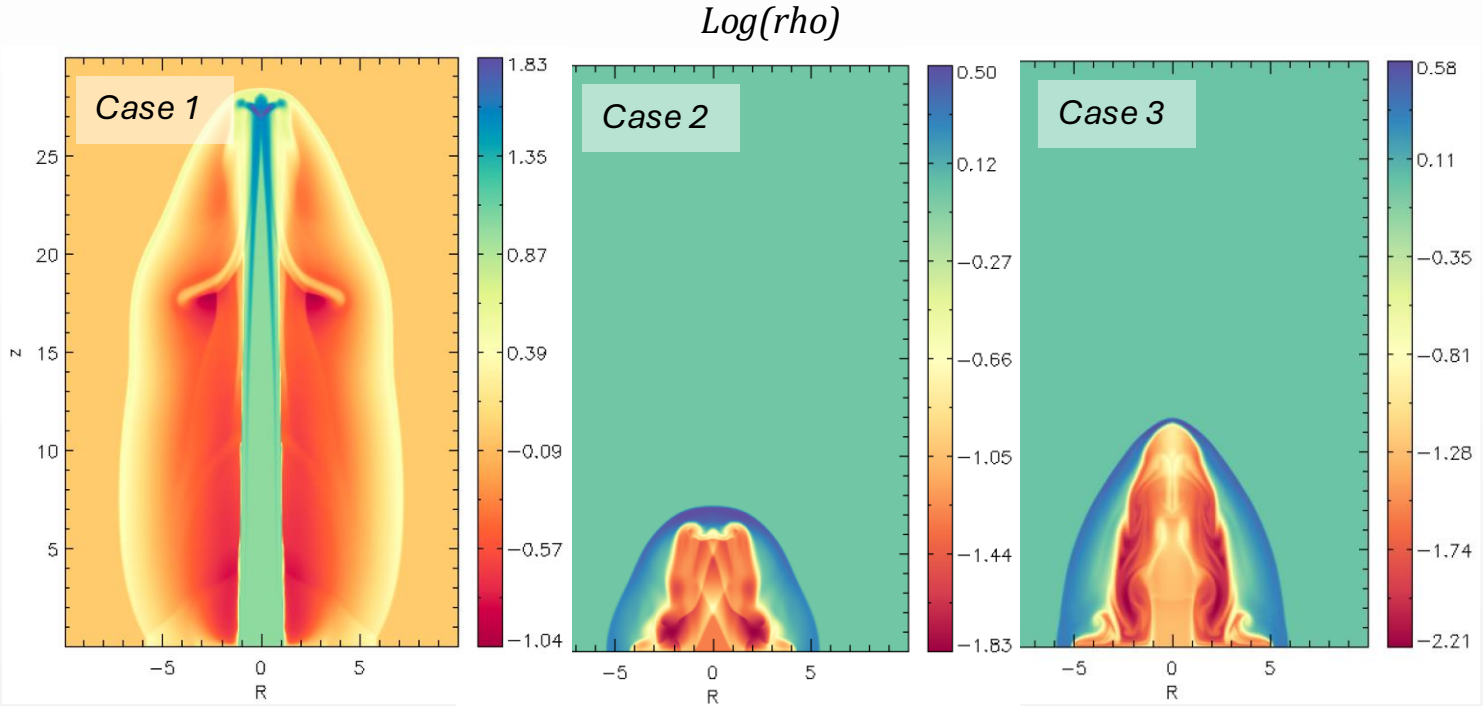
or

```
> mpirun -np <n> ./pluto # Parallel run
```

- Depending on your processor speed, the code may take few minutes to run.

Comparing Jet Morphologies

- The density contrast η and the magnetization σ_ϕ field influence the propagation and morphological properties of the jet:



3D Jet: Initial Configuration

- You may now try a 3D run, using one GPU or multiple cores.

```
> cp definitions_10.hpp definitions.hpp
> cp pluto_10.ini  pluto.ini
> python $PLUTO_DIR/setup.py
```

- Proceed as before by choosing a suitable `makefile`, e.g. `linux.mpicxx.defs` (for standard MPI run) or `linux.nvc++.acc.defs` (using GPU acceleration).

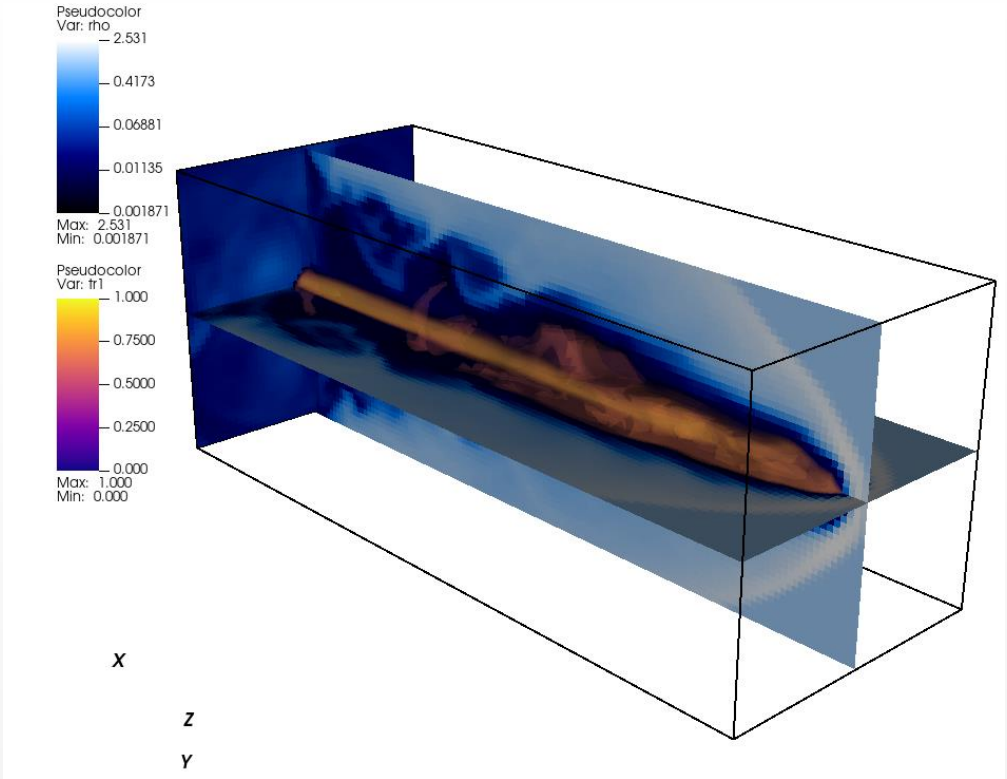
- The parameters and domain size are now slightly different to favour the onset of Current-driven instabilities (the |m|=1 *kink* mode).

- Also, we use a less dissipative method now (RK3 + PPM).

Visualization

- You may use Visit or Paraview to 3D volumetric rendering / slicing.

- The tracer variable (`tr1`) can be used to distinguish between jet and ambient material.



Appendix

Installing HPC-SDK

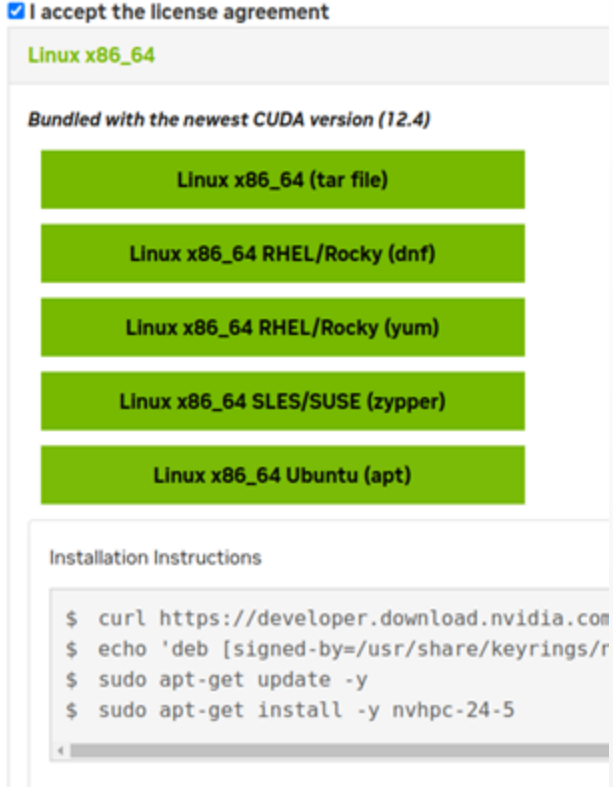
Go to:

https://developer.nvidia.com/nvidia-hpc-sdk-245-downloads.

Accept license agreement and for Linux x86_64 you can select apt instructions.

After the installation these lines should be added to your ~/.bashrc (following https://docs.nvidia.com/hpc-sdk/archive/24.5/hpc-sdk-install-guide/index.html):

```
NVARCH=`uname -s`_`uname -m`; export NVARCH
NVCOMPILERS=/opt/nvidia/hpc_sdk; export NVCOMPILERS
MANPATH=$MANPATH:$NVCOMPILERS/$NVARCH/24.5/compilers/man; export MANPATH
PATH=$NVCOMPILERS/$NVARCH/24.5/compilers/bin:$PATH; export PATH
export PATH=$NVCOMPILERS/$NVARCH/24.5/comm_libs/mpi/bin:$PATH
export MANPATH=$MANPATH:$NVCOMPILERS/$NVARCH/24.5/comm_libs/mpi/man
```



Installing HPC-SDK

To check you have everything to run gPLUTO type on the terminal `which nvc`. It should print something like:

```
nvc 24.5-0 64-bit target on x86-64 Linux -tp haswell
NVIDIA Compilers and Tools
Copyright (c) 2024, NVIDIA CORPORATION & AFFILIATES.  All rights reserved.
```

Typing `nvidia-smi` should print:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.183.01             Driver Version: 535.183.01   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce GTX 1650 Ti     Off | 00000000:01:00.0 Off |                  N/A |
| N/A   47C    P0              5W /  50W |      6MiB /  4096MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

If you have similar results the installation was successful!

Installing HPC-SDK

If you see an error with `nvidia-smi` or `nvaccelinfo` you are probably missing the CUDA toolkit and/or the NVIDIA driver.

Go to https://developer.nvidia.com/cuda-toolkit and click "Download now".

Here you find both **CUDA Toolkit Installer** and **Driver Installer.**

Karolina system

PUT INTO OPERATION

summer 2021

THEORETICAL PEAK PERFORMANCE

15,690 Tflop/s

OPERATING SYSTEM

Rocky Linux 8.9

COMPUTE NODES

831

CPU

720x 2x AMD 7H12, 64 cores, 2,6 GHz, 92,160 cores in total
72x 2x AMD 7763, 64 cores, 2,45 GHz, 9,216 cores in total
32x Intel Xeon-SC 8628, 24 cores, 2,9 GHz, 768 cores in total
36x 2x AMD 7H12, 64 cores, 2,6 GHz, 4,608 cores in total
2x 2x AMD 7452, 32 cores, 2,35 GHz, 128 cores in total

ACCELERATORS

576x NVIDIA A100

DOCUMENTATION

https://docs.it4i.cz/en/docs/introduction

NODE HOURS AVAILABLE

Karolina CPU: 1000
Karolina GPU: 500

Barbora system

PUT INTO OPERATION

autumn 2019

THEORETICAL PEAK PERFORMANCE

849 TFlop/s

OPERATING SYSTEM

CentOS 64bit 7.x

COMPUTE NODES

201

CPU

2 x Intel Cascade Lake, 18 cores, 2.6 GHz

ACCELERATORS

32x NVIDIA Tesla V100

DOCUMENTATION

https://docs.it4i.cz/en/docs/introduction

NODE HOURS AVAILABLE

Barbora CPU: 200
Barbora GPU: 200

Karolina command lines

login with:

```
ssh -i /path/to/.ssh/key atr-25-5-XXX@login1.karolina.it4i.cz
```

then `touch ~/.Xauthority` and exit

login again with -X:

```
ssh -i -X /path/to/.ssh/key atr-25-5-XXX@login1.karolina.it4i.cz
```

Download gPLUTO:

```
git clone https://gitlab.com/PLUTO-code/gPLUTO.git
```

Set virtual Environment to install PyPLUTO and load modules

```
python3 -m venv workshop
source workshop/bin/activate
export PLUTO_DIR=/home/atr-25-5-XXX/gPLUTO
ml Python/3.12.3-GCCcore-14.2.0
ml NVHPC/25.3-CUDA-12.8.0
export ver=/apps/all/NVHPC/25.3-CUDA-12.8.0/Linux_x86_64/25.3
export LIBRARY_PATH="$LIBRARY_PATH:$ver/comm_libs/nccl/lib"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$ver/comm_libs/nccl/lib"
export CPATH="$CPATH:$ver/comm_libs/nccl/include"
cd gPLUTO/Tools/PyPLUTO/
pip install ./
```

Barbora command lines

login with:

```
ssh -i /path/to/.ssh/key atr-25-5-XXX@login1.barbora.it4i.cz
```

then `touch ~/.Xauthority` and exit

login again with -X:

```
ssh -i -X /path/to/.ssh/key atr-25-5-XXX@login1.barbora.it4i.cz
```

Download gPLUTO:

```
git clone https://gitlab.com/PLUTO-code/gPLUTO.git
```

Set virtual Environment to install PyPLUTO and load modules

```
python3 -m venv workshop
source workshop/bin/activate
export PLUTO_DIR=/home/atr-25-5-XXX/gPLUTO
ml NVHPC/24.3-CUDA-12.3.0
ml Python/3.12.3-GCCcore-13.3.0
export ver=/apps/all/NVHPC/24.3-CUDA-12.3.0/Linux_x86_64/24.3
export LIBRARY_PATH="$LIBRARY_PATH:$ver/comm_libs/12.3/nccl/lib"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$ver/comm_libs/12.3/nccl/lib"
export CPATH="$CPATH:$ver/comm_libs/nccl/12.3/include"
cd gPLUTO/Tools/PyPLUTO/
pip install ./
```

Karolina *sbatch script*

```
#!/usr/bin/bash
#SBATCH --job-name pluto
#SBATCH --account ATR-25-5
#SBATCH --partition qcpu_exp # qgpu_exp
#SBATCH --ntasks=1 # 1 task
#SBATCH --cpus-per-task=1 # 1 CPU core for that task
##SBATCH --gpus=1 # put only for gpu partition
#SBATCH --time 00:01:00

export PLUTO_DIR=/home/atr-25-5-XXX/gPLUTO-dev
ml NVHPC/25.3-CUDA-12.8.0

./pluto > pluto.log
```

Barbora *sbatch script*

```
#!/usr/bin/bash
#SBATCH --job-name pluto
#SBATCH --account ATR-25-5
#SBATCH --partition qcpu_exp # qgpu_exp
#SBATCH --ntasks=1 # 1 task
#SBATCH --cpus-per-task=1 # 1 CPU core for that task
##SBATCH --gpus=1 # put only for gpu partition
#SBATCH --time 00:01:00

export PLUTO_DIR=/home/atr-25-5-XXX/gPLUTO-dev
ml NVHPC/24.3-CUDA-12.3.0

./pluto > pluto.log
```

Four Interactive nodes with GPU

Thanks to M. Aldinucci and S. Rabellino (Computer Science Dept.)

Each node equipped with:

1-GPU:	**V100-SXM2-32GB**
4 cores CPU:	**Intel Xeon Processor (Skylake, IBRS)**
RAM:	**32GB**

How to connect:

`ssh -X plutosymp<a>0<n>@pluto-symposium-node<N>.hpc4ai.unito.it`

<a> = nodes letter (a , b , c , d)

<n> = user number (1 to 4)

<N> = node number (1 to 4)

pass: webex chat

After ssh:
You can follow the instructions and can use gnuplot directly
*From $HOME you have ../shared folder where you find bashrc with path for nvc++

Run on HPC Jureca (interactive)

1 Download gPLUTO

```
git clone https://gitlab.com/PLUTO-code/gPLUTO.git
```

2 Request interactive node

```
salloc --account=punch_astro --partition=dc-gpu --time 01:00:00 --gres=gpu:4 --nodes 1
```

3 enter in the interactive node

sgoto job_number node_number (node number is relative 0 is the first node you request)

4 load module you need

```
module purge
module load NVHPC/24.3-CUDA-12
module load OpenMPI
```

5 export PLUTO_DIR

```
export PLUTO_DIR=/<fullpath>/gPLUTO  # set it also in your .bashrc or similar
```

6 cd to problem folder and cp the configuration files

```
cd $PLUTO_DIR/Test_Problems/Simple_Tests/Jet
```

7 problem setup

```
python3 $PLUTO_DIR/setup.py
```

8 compile

```
make -j
```

9 run / run in parallel

```
./pluto
```

```
srun -w $HOSTNAME --ntasks 4 --cpus-per-task 32 --gres=gpu:4 ./pluto
```

THE END