PLUTO code Essentials

Getting Started

Bhargav Vaidya

Indian Institute of Technology Indore

July 30, 2020

Outline

Basic Requisites

Code Compilation

- Serial version - C compiler e.g. gcc
- Parallel Version - MPICH library v2.0+ e.g., mpicc, mpirun, mpiexec etc.
- Python v2.7+, curses library (*optional*)
- (*only for AMR*) C++ compilers, Chombo Library & HDF5

Data Analysis and Visualization

- Python v2.7+ or v3.5+ **OR** Gnuplot **OR** IDL
- Recommended for 3D visualization and volume rendering – LLNL VisiT **OR** Kitware Paraview

Downloading from the Web-page

Code Webpage : `http://plutocode.ph.unito.it`

Unpacking and Installing the code

- Untar the .TAR.GZ file $\rightarrow$ tar -xvzf pluto-xx.tar.gz *where xx is the PLUTO version* $\rightarrow$ will create a folder named PLUTO.
  Latest version is 4.3 (June 2018)
- Define a PLUTO_DIR in your shell $\rightarrow$
  bashrc: export PLUTO_DIR =< Path to the PLUTO directory >
  tcsh: setenv PLUTO_DIR < Path to the PLUTO directory >

Comprehensive Documentation

The unique selling point of the code is the exhaustive documentation.

- The pdf version can be found in $PLUTO_DIR/Doc/userguide.pdf
- Additional there is a Doxygen documentation for all the test problems and source codes.

PLUTO v. 4.3 (June 2018)

User's Guide

(http://plutocode.ph.unito.it)

Developer: A. Mignone[1,2] [mignone@ph.unito.it]

Contributors: C. Zanni[2] [zanni@oato.inaf.it]
B. Vaidya[3] [bvaidya@iiti.ac.in]
G. Muscianisi[4]
P. Tzeferacos[5]
O. Tesileanu[6]
T. Matsakos

1 Dipartimento di Fisica, Turin University, Via P. Giuria 1 - 10125 Torino (TO), Italy
2 INAF Osservatorio Astronomico di Torino, Via Osservatorio, 20 10025 Pino Torinese (TO), Italy
3 Indian Institute of Technology Indore, Indore, Khandwa Road, Simrol (Indore)
4 Consorzio Interuniversitario CINECA, via Magnanelli, 6/3, 40033 Casalecchio di Reno (Bologna), Italy
5 FLASH Center, University of Chicago, USA
6 Department of Physics, University of Bucharest, Str. Atomistilor nr. 405, RO-077125 Magurele, Ilfov, Romania

Problem Description

- What is the underlying physics ?
  $\rightarrow$ With or without magnetic fields ? Is the flow relativistic ?
- In what geometry do you wish to solve the equations ?
  $\rightarrow$ What are the dimensions? What are the grid extends for each of these dimensions?
- Does the problem require to add source terms?
  $\rightarrow$ What is the functional form of source term ? Which conservation equations are affected?
- What physical conditions would be used to prescribe boundary conditions?
  $\rightarrow$ Does the solution requires userdef boundary conditions? How to minimize the effects of boundary where not required?
- What is the time-scale upto which the simulation should run?

An Example!

Interaction of solar wind with Earth's Magneto-sphere.

- What is the underlying physics ? Non-relativistic with magnetic fields
- In what geometry do you wish to solve the equations ? 3D Cartesian $\rightarrow (x, y, z) = (20R_{\mathrm{E}}, 20R_{\mathrm{E}}, 100R_{\mathrm{E}})$, Earth is centered at (0,0,0)
- Does the problem require to add source terms? Yes $\rightarrow$ Gravity to support Earth's magneto-sphere.
- What physical conditions would be used to prescribe boundary conditions? Injection of solar wind on left z axis and free flow in all other possible directions.
- What is the time-scale upto which the simulation should run? Till steady state is achieved.

Setting up using Python

In order to input the problem definitions to the code a python interface is created.

*p*ython $PLUTO_DIR/setup.py <options >

```
>> Python setup (May 2018) <<

Working dir: /Users/Bhargav/PLUTO_Dev/PLUTO-4.3
PLUTO dir  : /Users/Bhargav/PLUTO_Dev/pluto


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

Setting up using Python

Options	Remarks/Modules
	Default option
--with-fd	Using the Finite Difference Scheme (Only non-relativistic physics)
--with-sb	Shearing Box
--with-fargo	FARGO module for Acceretion Disk
--with-chombo	The chombo module for AMR runs
--with-particles	Invoking the Particle module

Input Files : **definitions.h**

Example : Magnetized Non-relativistic Blast wave in 2D

- Common definition block
- Physics dependent block
- User-defined (labeled) parameters
- User-defined constants [more for expert users]

```
#define  PHYSICS                        MHD
#define  DIMENSIONS                     2
#define  COMPONENTS                     2
#define  GEOMETRY                       CARTESIAN
#define  BODY_FORCE                     NO
#define  FORCED_TURB                    NO
#define  COOLING                        NO
#define  RECONSTRUCTION                 LINEAR
#define  TIME_STEPPING                  RK2
#define  DIMENSIONAL_SPLITTING          NO
#define  NTRACER                        0
#define  USER_DEF_PARAMETERS            7

/* -- physics dependent declarations -- */

#define  EOS                            IDEAL
#define  ENTROPY_SWITCH                 NO
#define  DIVB_CONTROL                   CONSTRAINED_TRANSPORT
#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  P_IN                           0
#define  P_OUT                          1
#define  BMAG                           2
#define  THETA                          3
#define  PHI                            4
#define  RADIUS                         5
#define  GAMMA                          6

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

#define  CHAR_LIMITING                  YES
#define  LIMITER                        VANLEER_LIM
#define  CT_EMF_AVERAGE                 ARITHMETIC
#define  CT_EN_CORRECTION               YES
#define  ASSIGN_VECTOR_POTENTIAL        YES

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

Input Files : **pluto.ini** - Part I

- Grid block
- Chombo block
- Time Block
- Solver Block
- Boundary Block

```
[Grid]

X1-grid    1    -0.5    200    u    0.5
X2-grid    1    -0.5    200    u    0.5
X3-grid    1    -0.5    1    u    0.5

[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              0.4
CFL_max_var    1.1
tstop            0.01
first_dt         1.e-6

[Solver]

Solver         roe

[Boundary]

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

Input Files : **pluto.ini** - Part II

- Static Grid Output Block
- Chombo HDF5 output Block
- Parameters Block

```
[Static Grid Output]

uservar    0
dbl       1.e3  -1   single_file
flt       -1.0  -1   single_file
vtk       -1.0  -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]

P_IN                        1.e3
P_OUT                       0.1
BMAG                        28.2094791773878
THETA                       90.0
PHI                         90.0
RADIUS                      0.1
GAMMA                       1.4
```

Input Files : **init.c**

Init block : Inputs –

- **v[NVAR]** $\rightarrow$ an array of primitive variables
- x1, x2, x3 $\rightarrow$ Point co-ordinate for the chosen geometry.
- Used to set the initial conditions in the domain point by point.

```c
/* ********************************************************************* */
void Init (double *us, double x1, double x2, double x3)
/*
 *
 ********************************************************************* */
{
  double r, theta, phi, B0;
  g_gamma = g_inputParam[GAMMA];
  r = D_EXPAND(x1*x1, + x2*x2, + x3*x3);
  r = sqrt(r);

  us[RHO] = 1.0;
  us[VX1] = 0.0;
  us[VX2] = 0.0;
  us[VX3] = 0.0;
  us[PRS] = g_inputParam[P_OUT];
  if (r <= g_inputParam[RADIUS]) us[PRS] = g_inputParam[P_IN];

  theta = g_inputParam[THETA]*CONST_PI/180.0;
  phi   =   g_inputParam[PHI]*CONST_PI/180.0;
  B0    = g_inputParam[BMAG];

  us[BX1] = B0*sin(theta)*cos(phi);
  us[BX2] = B0*sin(theta)*sin(phi);
  us[BX3] = B0*cos(theta);


  #if GEOMETRY == CARTESIAN
   us[AX1] = 0.0;
   us[AX2] =  us[BX3]*x1;
   us[AX3] = -us[BX2]*x1 + us[BX1]*x2;
  #elif GEOMETRY == CYLINDRICAL
   us[AX1] = us[AX2] = 0.0;
   us[AX3] = 0.5*us[BX2]*x1;
  #endif

  #if BACKGROUND_FIELD == YES
   us[BX1] = us[BX2] = us[BX3] =
   us[AX1] = us[AX2] = us[AX3] = 0.0;
  #endif
}
```

Makefile & Compilation

A **makefile** is created based on the architecture/compiler of your choice. Some standard combinations are available in the option of *Change Makefile* option of the setup.

```
>> Change makefile <<


 Darwin.gcc.defs
 Darwin.mpicc.defs
 Linux.gcc.defs
 Linux.mpicc.defs
 MARCONI.mpiicc.defs
 Template.defs
 debug.defs
 profile.defs
```

Finally, compile the code using the - **make** command in the terminal to get the executable PLUTO!

Running the Code

Check with ldd if all libraries are linked. Serial and Parallel run commands.

- If compiled with **gcc** the command to run is (Serial mode) : **./pluto**
- If compiled with Parallel compilers liked **mpicc**, then the command to run is : **mpiexec -n 4 ./pluto**

At the end of the run, the code writes the data in prescribed format along with **.out** and **.log** files.
The **grid.out** contains information about the grid to be read for visualization.
The **.out** files corresponding to each data-set has information on variables stored at different time.

Output Files : **log files**

- The log files keep track of the progress of the simulations
- For parallel job, each processor writes it own log file.
- Frequency as to when the log output should written is governed by "log" input in pluto.ini

```
> 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)
  [Mag Field]:    4.585e-07 (Gauss)

> Number of processors: 1
> Proc size:              200 X 200
> Writing file #0 (dbl) to disk...
> Starting computation...

step:0; t = 0.0000e+00; dt = 1.0000e-06; 0.0 %
        [Mach = 0.131337]
step:1; t = 1.0000e-06; dt = 1.1000e-06; 0.0 %
        [Mach = 0.268750]
step:2; t = 2.1000e-06; dt = 1.2100e-06; 0.0 %
        [Mach = 0.400550]
step:3; t = 3.3100e-06; dt = 1.3310e-06; 0.0 %
        [Mach = 0.517421]
step:4; t = 4.6410e-06; dt = 1.4641e-06; 0.0 %
        [Mach = 0.614759]
step:5; t = 6.1051e-06; dt = 1.6105e-06; 0.1 %
        [Mach = 0.692004]
step:320; t = 9.7649e-03; dt = 3.2638e-05; 97.6 %
          [Mach = 11.409206]
step:321; t = 9.7975e-03; dt = 3.2629e-05; 98.0 %
          [Mach = 11.374097]
step:322; t = 9.8302e-03; dt = 3.2619e-05; 98.3 %
          [Mach = 11.333627]
step:323; t = 9.8628e-03; dt = 3.2608e-05; 98.6 %
          [Mach = 11.283199]
step:324; t = 9.8954e-03; dt = 3.2599e-05; 99.0 %
          [Mach = 11.251811]
step:325; t = 9.9280e-03; dt = 3.2591e-05; 99.3 %
          [Mach = 11.241501]
step:326; t = 9.9606e-03; dt = 3.2583e-05; 99.6 %
          [Mach = 11.228911]
step:327; t = 9.9932e-03; dt = 6.8371e-06; 99.9 %
          [Mach = 11.211180]
> Writing file #1 (dbl) to disk...

> Total allocated memory   12.87 Mb
> Elapsed time              0d:0h:0m:18s
> Average time/step         5.49e-02  (sec)
> Local time                Fri Jun 14 04:02:32 2019
> Done
```

Suffix Properties

Command : ./pluto <*suffix options*>

Options	Function
-restart n	Restarts from data.nnnn.dbl file
-maxsteps n	Runs the code for n steps.
-no-write	Does not write any data files
-xres Nx	Overwrites the resolution set in pluto.ini with Nx along X and scales accordingly in other direction

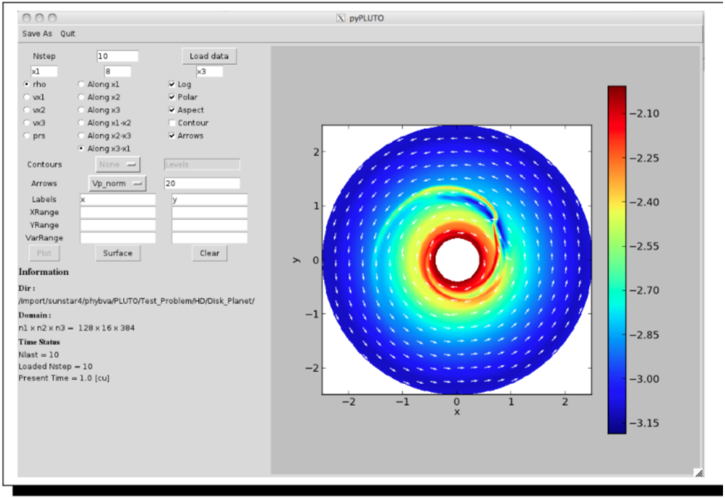
Data formats

The code outputs in various data formats either in the *single file* format or *multiple file* format. The different usually used formats are -

- **.dbl** - Native binary in double format. Useful for restarting the code.
- **.flt** - Native binary float format
- **.vtk** - Visualization Tool kit format. (VisIt visualization)
- **.hdf5** - Obtained for AMR run (VisIt visualization)
- **.tab**, **.ppm** - Not very relevant for general runs.

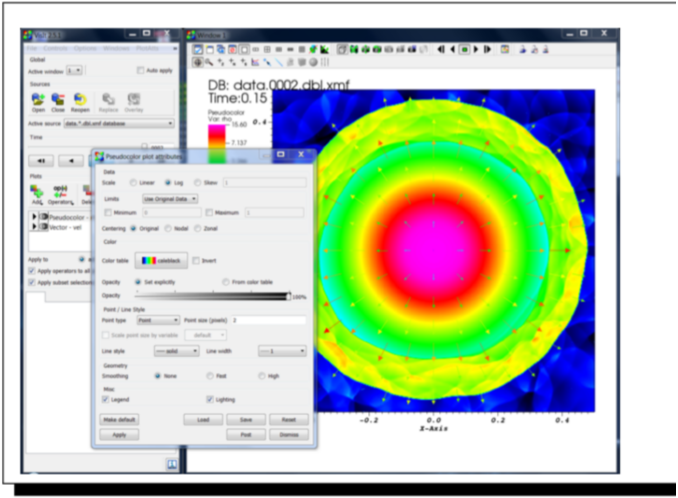
Visualization using Python

Valid for all of the above mentioned data formats – Does not support 3D visualization.



Visualization using Visit

Valid for the **.vtk** and **.hdf5** data file formats – Very useful for 3D visualization.



Various PHYSICS module

- Hydrodynamics (HD)
- Magneto-Hydrodynamics (MHD)
- (Special) Relativistic HD
- (Special) Relativistic MHD
- Particles – a) Lagrangian, b) MHD-PIC, c) Dust.

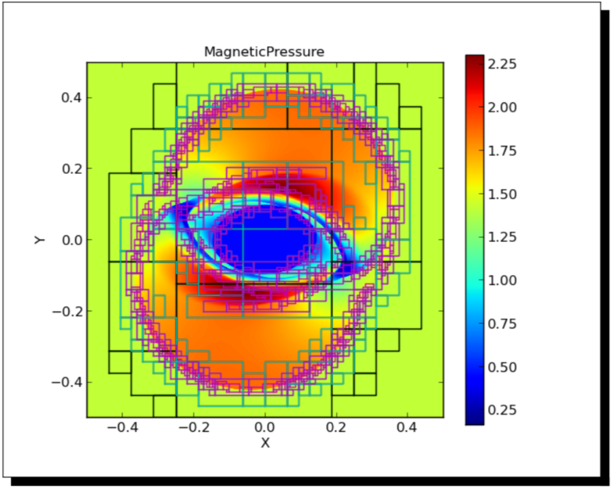
The $\nabla \cdot \vec{B} = 0$ constraint is governed by i) Powell's Eight wave method, ii) Divergence Cleaning approach and iii) Constraint Transport.

Source Terms, Non-Ideal Physics

- Body Force : Gravity in both Vector and Potential format
- Optically Thin Radiation Cooling.
- Forced Turbulence with appropriate stirring
- Ambipolar Diffusion
- Hall Effect
- Magnetic Resistivity
- Thermal Conduction
- Viscosity
- Option for working the Rotating Frame.
- Options for various EoS.

Adaptive Mesh Refinement

PLUTO code has fully developed AMR capability supporting all geometry and dimensions using the CHOMBO library.



Hands-on Session with PLUTO

I will discuss the following –

- *HD Sod Shock tube problem*
- *MHD Blast Wave problem*

You will have to run the following

- Rayleigh-Taylor Instability
- Kelvin-Helmholtz Instability
- Study of Shock-cloud collision.