pyPLUTO Documentation

Release 1.0

Bhargav Vaidya , Denis Stepanovs

May 04, 2012

CONTENTS

CHAPTER

ONE

INFORMATION

This is the documentation for pyPLUTO code. This is created using a cool software named Sphinx.

Author Bhargav Vaidya (b.vaidya at leeds.ac.uk) and Denis Stepanovs (stepanovs at mpia.de)

Release 1.0

Date May 04, 2012

1.1 Installation

After ensuring that all of the above pre-requisites are working and installed the user can download pyPLUTO-1.zip from here.

The code can be installed using a standard procedure. This can be done by following steps:

1. **Global Install**

The Python version from the EPD version by default creates a PYTHONPATH. If no option is chosen for preferred path then in that case the code will be installed in that default path. This might require the user to have access to the root password:

- Unzip the source code : `unzip pyPLUTO-1.0.zip`
- Enter into the directory : `cd pyPLUTO-1.0`
- Install the code in the default path : `python setup.py install`

2. **Local Install**

The best practice is to create your own PYTHONPATH and do a local install in the following way:

- Create a directory where to store this module : `mkdir MyPython_Modules`
- Unzip the source code : `unzip pyPLUTO-1.0.zip`
- Enter into the directory : `cd pyPLUTO-1.0`
- Install the code in the directory created : `python setup.py install --prefix=<path to MyPython_Modules>`
- Then append the following in your .bashrc : `export PYTHONPATH = $HOME/MyPython_Modules/lib/python<ver>/site-packages`

where <ver> is the python version which the user have used to install the package.

After the successful installation, the user can start using GUI application by appending the <path to GUI_pyPLUTO.py> into their PATH.

1.2 Loading the Data

class pyPLUTO.**pload**(*ns*, *w_dir=None*)

This Class has all the routines loading the data from the binary files output from PLUTO Simulations. Assign an object when the data is loaded for some *ns*.

Usage:

```
import pyPLUTO as pp

wdir = '/path/to/the data files/'

D = pp.pload(1,w_dir=wdir)
```

Now D is the pyPLUTO.pload object having all the relevant information of the corresponding data file - data.0001.dbl.

It has the following attributes –

data()

This method loads the data from the file name "data.ns.dbl" or "varname.ns.dbl".

geometry()

This method has the geometry information of the problem considered.

get_varinfo()

This method reads the dbl.out and stores the information in a dictionary.

Keyword Arguments:

fltype – returns the filetype of storing data (single_file or multiple_file)

nvar – Number of variables

allvars – A list of variables names. Each of the variable name will be the attributes to the pyPLUTO.pload object.

grid()

This method returns the necessary grid information in form a dictionary.

Keyword Arguments:

n1 – number of grid cells in x1 direction

n2 – number of grid cells in x2 direction

n3 – number of grid cells in x3 direction

x1 – Array x1

x2 - Array x2

x3 - Array x3

dx1 - Array dx1

dx2 - Array dx2

dx3 - Array dx3

time_info()

This method returns a dictionary that has the time information for the step ns.

Keyword Arguments:

time – Gets the simulation time at step ns.

dt – Get the time step dt for step ns.

Nstep – Get the value of nstep for the step ns.

1.3 Viewing the Data

class pyPLUTO.**Image**

This Class has all the routines for the imaging the data and plotting various contours and fieldlines on these images. CALLED AFTER pyPLUTO.pload object is defined

field_interp(*var1*, *var2*, *x*, *y*, *dx*, *dy*, *xp*, *yp*)

This method interpolates value of vector fields (var1 and var2) on field points (xp and yp). The field points are obtained from the method field_line.

***Arguments*:** var1 – 2D Vector field in X direction

var2 – 2D Vector field in Y direction

x – 1D X array

y – 1D Y array

dx – 1D grid spacing array in X direction

dy – 1D grid spacing array in Y direction

xp – field point in X direction

yp – field point in Y direction

field_line(*var1*, *var2*, *x*, *y*, *dx*, *dy*, *x0*, *y0*)

This method is used to obtain field lines (same as fieldline.pro in PLUTO IDL tools).

***Arguments*:** var1 – 2D Vector field in X direction

var2 – 2D Vector field in Y direction

x – 1D X array

y – 1D Y array

dx – 1D grid spacing array in X direction

dy – 1D grid spacing array in Y direction

x0 – foot point of the field line in X direction

y0 – foot point of the field line in Y direction

getSphData(*Data*, *w_dir=None*, ***kwargs*)

This method transforms the vector and scalar fields from Spherical co-ordinates to Cylindrical.

Arguments:

Data – puPLUTO.pload object

w_dir – /path/to/the/working/directory/

Keywords:

rphi – [Default] is set to False implies that the r-theta plane is transformed. If set True then the r-phi plane is transformed.

x2cut – Applicable for 3D data and it determines the co-ordinate of the x2 plane while r-phi is set to True.

x3cut – Applicable for 3D data and it determines the co-ordinate of the x3 plane while r-phi is set to False.

myfieldlines (*Data*, *x0arr*, *y0arr*, *stream=False*, ***kwargs*)

This method overplots the magnetic field lines at the footpoints given by (x0arr[i],y0arr[i]).

Arguments:

Data – pyPLUTO.pload object

x0arr – array of x co-ordinates of the footpoints

y0arr – array of y co-ordinates of the footpoints

stream – keyword for two different ways of calculating the field lines.

True – plots contours of rAphi (needs to store vector potential)

False – plots the fieldlines obtained from the field_line routine. (Default option)

Keywords:

colors – A list of matplotlib colors to represent the lines. The length of this list should be same as that of x0arr.

lw – Integer value that determines the linewidth of each line.

ls – Determines the linestyle of each line.

pldisplay (*var*, ***kwargs*)

This method allows the user to display a 2D data using the matplotlib's pcolormesh.

Arguments:

var – 2D array that needs to be displayed.

Keywords:

x1 – The 'x' array

x2 – The 'y' array

vmin – The minimum value of the 2D array (Default : min(var))

vmax – The maximum value of the 2D array (Default : max(var))

title – Sets the title of the image.

label1 – Sets the X Label (Default: 'XLabel')

label2 – Sets the Y Label (Default: 'YLabel')

cbar – Its a tuple to set the colorbar on or off. cbar = (True,'vertical') – Displays a vertical colorbar

cbar = (True,'horizontal') – Displays a horizontal colorbar

cbar = (False,'') – Displays no colorbar.

Usage:

```
import pyPLUTO as pp
wdir = '/path/to/the data files/'
D = pp.pload(1,w_dir=wdir)
I = pp.Image()
f1 = figure()
```

```
ax1 = f1.add_subplot(111)

I.pldisplay(D.v2,x1=D.x1,x2=D.x2,cbar=(True,'vertical'),title='Velocity',label1=
```

pltSphData(*Data*, *w_dir=None*, ***kwargs*)

This method plots the transformed data obtained from getSphData using the matplotlib's imshow

Arguments:

Data – pyPLUTO.pload object

w_dir – /path/to/the/working/directory/

Keywords:

plvar – A string which represents the plot variable.

1.4 Additional tools

class pyPLUTO.**Tools**

This Class has all the functions doing basic mathematical operations to the vector or scalar fields. It is called after pyPLUTO.pload object is defined.

Div(*u1*, *u2*, *x1*, *x2*, *dx1*, *dx2*, *geometry=None*)

This method calculates the divergence of the 2D vector fields u1 and u2. It requires the vectors x1 and x2 with their respective grid spacings dx1 and dx2.

The keyword *geometry* is by default set to 'cartesian'. It can be set to eitherone of the following : *cartesian*, *cylindrical*, *spherical* or *polar*. To calculate the divergence of the vector fields, respective geometric corrections are taken into account based on the value of this keyword.

Grad(*phi*, *x1*, *x2*, *dx1*, *dx2*, *polar=False*)

This method calculates the gradient of the 2D scalar phi. It requires the vectors x1 and x2 with their respective grid spacings dx1 and dx2.

The keyword *polar* is by default set to False, when set True respective geometric corrections are taken into account for calculating the gradient.

RTh2Cyl(*R*, *Th*, *X1*, *X2*)

Transforms vector (X1,X2) given in spherical coordinates to cylindrical. X1 and X2 could correspond to Br and Bth, R and Th - matrices with sph. coordinates The result is (Y1,Y2) which correspond to vector in cylindrical coords (Br,Bz)

congrid(*a*, *newdims*, *method='linear'*, *centre=False*, *minusone=False*)

Arbitrary resampling of source array to new dimension sizes. Currently only supports maintaining the same number of dimensions. To use 1-D arrays, first promote them to shape (x,1).

Uses the same parameters and creates the same co-ordinate lookup points as IDL"s congrid routine, which apparently originally came from a VAX/VMS routine of the same name.

method:

neighbour - closest value from original data

nearest and linear - uses n x 1-D interpolations using scipy.interpolate.interp1d

(see Numerical Recipes for validity of use of n 1-D interpolations)

spline - uses ndimage.map_coordinates

centre:

True - interpolation points are at the centres of the bins

False - points are at the front edge of the bin

minusone:

For example- inarray.shape = (i,j) & new dimensions = (x,y)

False - inarray is resampled by factors of (i/x) * (j/y)

True - inarray is resampled by(i-1)/(x-1) * (j-1)/(y-1)

This prevents extrapolation one element beyond bounds of input array.

deriv(*Y*, *X=None*)
Calculates the derivative of Y with respect to X.

Keywords: Y : 1-D array to be differentiated. X : 1-D array with len(X) = len(Y).

If X is not specified then by default X is chosen to be an equally spaced array having same number of elements as Y.

getUniformGrid(*r*, *th*, *rho*, *Nr*, *Nth*)
Transforms data with non-uniform axes (stretched) into uniform. r, th - grids, rho(r,th) - data, Nr and Nth - sizes of new (uniform) grid

myInterpol(*RR*, *N*)
Interpolates vector RR to N-grids. Returns RRi-interpolated vector and NNi - grid of

sph2cyl(*D*, *Dx*, *rphi=None*, *theta0=None*)
Transforms spherical data into cilindrical using interpolation. D - structure got from 'pload' method. Dx - data itself (D.rho for example). Transforms poloidal (R-Theta) data by default. Use rphi=True to get (R-Phi) transformation for fixed theta0

pyPLUTO.**get_nstepstr**(*ns*)
Convert the float input *ns* into a string that would match the data file name.

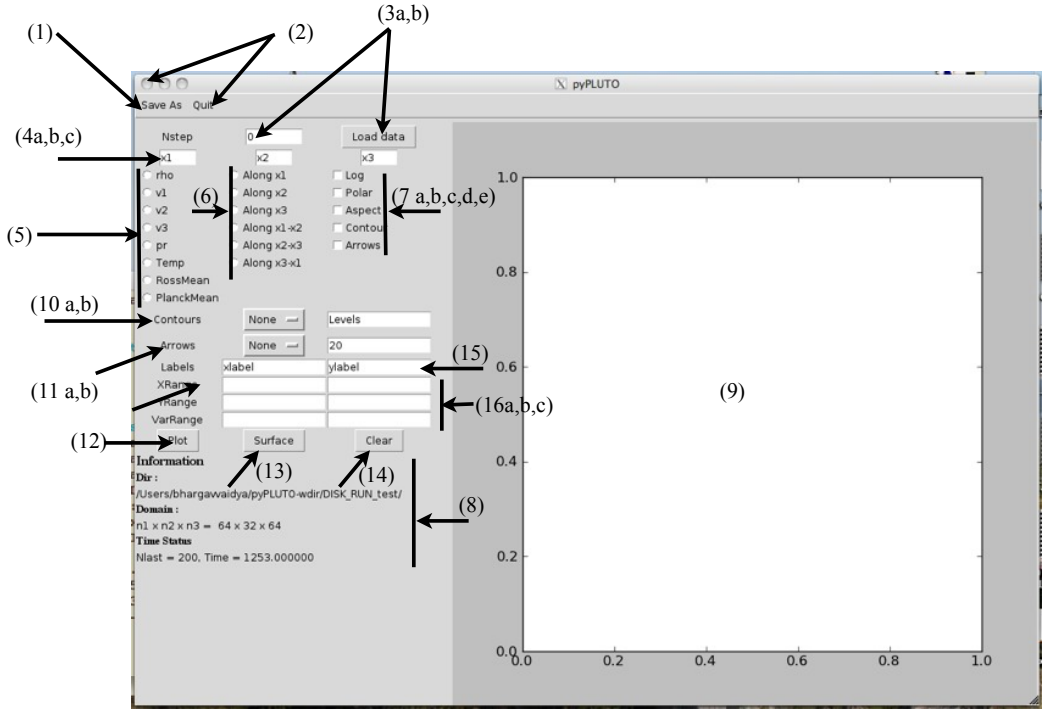
pyPLUTO.**nlast_info**(*w_dir=None*)
Prints the information of the last step of the simulation as obtained from dbl.out

CHAPTER

TWO

GRAPHIC USER INTERFACE

The Graphic User Interface for the pyPLUTO code.



The Graphic User Interface of the pyPLUTO code for a typical three dimensional Hydrodynamical example. The various functionality are marked with numbers as shown in the figure. In-detailed description of these functionalities are described below.

The various functionalities in the Graphic User Interface (GUI) of the pyPLUTO code are explained in details below:

1. *Save As* : The drop down menu allows the user to save the figure displayed in various formats viz. ‘eps’, ‘pdf’, ‘jpg’, ‘png’.

2. *Quit* : The user can close the GUI using this button.

3. *Loading the Data* : This allows the user to load data.

   a. *Nstep* : The number of the data file should be inputted in the blank panel provided. By default initial data file i.e. Nstep = 0 is loaded. The input number should be a valid positive number and the corresponding data file should exists in the working directory.

   b. *Load Data* : The user can click on this button to load the appropriate data file whose number is inputted in the Nstep panel. Each time the user modifies the value in the Nstep field, this button needs to be clicked to load the corresponding file.

4. *Axis cuts* : The user can choose the appropriate values in these panels to plot/image a particular cut/slice. No default value is set so error will occur if invalid entry is inputted.

   a. *x1* : The field to input the x1 cut value. This field should not be blank if the user either chooses *Along x2* or *Along x3* or *Along x2-x3* in (6). This field is disabled if the problem considered is 1-D.

   b. *x2* : The field to input the x2 cut value. This field should not be blank if the user either chooses *Along x1* or *Along x3* or *Along x3-x1* in (6). This field is disabled if the problem considered is 1-D.

   c. *x3* : The field to input the x3 cut value. This field should not be blank if the user either chooses *Along x1* or *Along x2* or *Along x1-x2* in (6). This field is disabled if the problem considered is either 1-D or 2-D.

5. *Choose Variables* : The user can choose the variable to plot from the list. This list also includes any additional variable stored using the *userdef_output.c*. So basically all the variables listed in the *dbl.out* are enlisted in this column. The code will have a disabled *Plot* and *Surface*, until a variable is chosen in this column.

6. *Choose Slices* : The code provides the user to either plot a 1D *Plot* of any of the chosen variable along any axis. Alternatively, the code also allows the user to make a 2D *Surface* image along any combination of two axes. This column is redundant in case the numerical problem under consideration is a 1D problem. With the choice of the slice, the user should also have appropriate axis cuts specified as mentioned above else the code will result into an error.

7. *Additional Processing* : The GUI interface allows the user to carry some additional processing on the variable chosen. They are the following -

   a. *Log* : The user can plot logarithmic values using this option.

   b. *Polar* : The user can use this option to project the data from Spherical coordinates to Cylindrical coordinates. In the present version, the code handles the projection in the r-θ plane (Along x1-x2) and the r-ϕ plane (Along x3-x1).

   c. *Aspect* : Choosing this option allows the user to ensure that the image (only the Surface) produced has proper aspect ratio. By default the aspect ratio is set to 'auto'.

   d. *Contour* : The user can choose this option to overplot the surface plot with contours [see (10 a,b)].

   e. *Arrow* : The user can choose this option to overplot the surface plot with vector arrows [see (11 a,b)].

8. *Information Panel* : The user can view the basic information regarding the problem under consideration. In this panel, the user can get information of the current working directory, along with the domain of the numerical problem and finally also the final time step of that problem.

9. *Figure Window* : In this panel the corresponding *Plot* or the *Surface* (image with colorbar) will be displayed. In order that the GUI fits into the whole screen, the user can play with the size of the figure by modifying the code and specifying the size of the figure window. The default value is : figsize=(7,7).

10. *Contours* : The user can activate this option by choosing the *Contour* tab [7(d)].

    a. *Contour Variable* : The 2D contours of listed variables can be plotted on the surface plot. The user can choose among all the variables that are available for plotting along with additional provision [only for the MHD problem] of plotting the *Current* [x1*b3] and *Magnetic field lines* [x1*A3]. The Magnetic field lines contour will only work if the code data consists information on the vector potential 'A3'. The list

by default is set to 'None' and thus will give error is the user tries to plot contours without choosing the appropriate variable from the drop down list.

b. *Contour Levels* : The user can provide the contours levels which is required separated by ','. If the user does not wish to provide the levels then by default a total of 5 contours of automatically chosen levels will be displayed. Further, the user can also choose to display logarithmic contours by having the first entry in this panels as *log*. For example,

- log,-1.5,-1.8,-2.0 : This plots the logarithmic contours for the chosen Contour variable at levels marked by 10^{-1.5}, 10^{-1.8} and 10^{-2.0}
- 1.0,2.0,3.0 : This will display normal contours for the chosen Contour variable at levels marked by 1.0, 2.0, and 3.0
- Blank [Default]: 5 contours of automatically chosen levels will be displayed

There is no limit to the number of levels that can be inputed in this field. By convention all negative contours will be shown as *dashes*. Incase of logarithmic contours, the *dashes* would represent contours for levels less than unity.

11. *Arrows* : The user can activate this option by choosing the 'Arrows' tab [7(e)].

a. *Arrow Variable* : The vector arrows of velocity field [Vp and Vp_norm] and the magnetic field [Bp and Bp_norm] (only in MHD) can be displayed. The options of . The variables with '_norm' indicate that the arrows are normalized, i.e. each arrow will have unit length. The list by default is set to 'None' and thus will give error is the user tries to plot arrows without choosing the appropriate variable from the drop down list.

b. *Arrow Spacing* : The user can provide the spacing value which indicates the size of the congrid matrix used to create the vector plots. Higher values will produce a more "dense" plot. Default is set to 20.

12. *Plot* : This button is by default deactivated and it will be active only when the user has fulfilled the conditions of valid loading of data [(3) a,b], choosing the variable [(5)] and choosing to plot either Along x1 or Along x2 or Along x3 [(6)]. Only in case of 1-D numerical problems this button will be activated by default.

13. *Surface* : This button is by default deactivated and it will be active only when the user has fulfilled the conditions of valid loading of data [(3) a,b], choosing the variable [(5)] and choosing to plot either Along x1-x2 or Along x2-x3 or Along x3-x1 [(6)]. Consecutively pressing this button will clear the existing image and create a new one.

14. *Clear* : This button allows the user to clear the plot.

15. *Labels* : The user can choose the x and y labels for their plots/images. The user can also use standard TeX symbols within the '$' sign. By default they are set to 'xlabel' and 'ylabel' respectively.

16. *Ranges* : The user can choose range of values to be displayed from these panels.

a. *Xrange* : To set the minimum (left entry) and maximum (right entry) range of the X axis. A 'blank' entry would mean that by default the minimum and the maximum of the X vector will be shown (i.e. the full range).

b. *Yrange* : To set the minimum (left entry) and maximum (right entry) range of the Y axis. A 'blank' entry would mean that by default the minimum and the maximum of the Y vector will be shown (i.e. the full range). This becomes ineffective in case if the user wants to plots a line. To set the range for the Y axis in case of 'Plot', the user should use the VarRange option.

c. *VarRange* : To set the minimum (left entry) and maximum (right entry) range of the Variable. A 'blank' entry would mean that by default the minimum and the maximum of the Variable will be shown (i.e. the full range). In case the user wants a 'Surface' image then with this option the user can choose the maximum and the minimum of the image. In case of 'Plot', this becomes the Y axis range.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *search*

PYTHON MODULE INDEX

p

INDEX