pyPLUTO Documentation

Release 4-1.0

Bhargav Vaidya , Denis Stepanovs

November 02, 2012

CONTENTS

CHAPTER

ONE

INFORMATION

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

Version 4-1.0

Date November 02, 2012

TASK : Quick Tool for Visualization of PLUTO 4.0 code data (Mignone2007)

DESCRIPTION : The code is completely written using the Python Language. Further the GUI is developed with the Tkinter Interface.

FEATURES :

1. Completely based on Python and easy to work without need of any licenses like in IDL.

2. The GUI environment provides a tool for quick-check of data during the simulations runs.

3. The code is user friendly and allows the user to even do further plotting of contours, velocity vectors on the surface plot. Also the code can read user-defined variables.

CHAPTER

TWO

GETTING STARTED

2.1 Installation

After ensuring that all of the above pre-requisites are working and installed the user can download pyPLUTO-4-1.0.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 may require the user to have access to the root password:

- Unzip/Untar the source code : `unzip pyPLUTO-|version|.zip` or `tar -xvzf pyPLUTO-|version|.tar.gz`
- Enter into the directory : `cd pyPLUTO-|version|`
- Install the code in the default path : `python setup.py install`

2. **Local Install (Recommended)**

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/Untar the source code : `unzip pyPLUTO-|version|.zip` or `tar -xvzf pyPLUTO-|version|.tar.gz`
- Enter into the directory : `cd pyPLUTO-|version|`
- 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 =<path to MyPython_Modules>/lib/python<ver>/site-packages`

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

3. **GIT Repository**

This code is also available as a git repo and can be obtained as follows : `git clone git://github.com/coolastro/pyPLUTO.git`

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

2.2 Loading the Data

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

Inputs:

ns – Time step of the data, data.ns.dbl or data.ns.flt

w_dir – path to the directory which has the dbl.out(or flt.out) and the data

datatype – If the data is of 'float' type then datatype = 'float' else by default the datatype is set to 'double'.

Outputs:

A pyPLUTO.pload object having all the relevant information of the corresponding data file

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.

data(*datatype*)

This method loads the data from the file name "data.ns.dbl"(in case of single_file) or "varname.ns.dbl"(in case of multiple files).

Inputs:

datatype – If the data is of 'float' type then datatype = 'float' else by default the datatype is set to 'double'.

Outputs:

It returns a dictionary with all the variable names as the keys. **NOTE** All these variable names are set as attributes to the pyPLUTO.pload object like the grid() function.

geometry()

This method has the geometry information of the problem considered.

get_varinfo(*datatype*)

This method reads the dbl.out (or flt.out) and stores the information in a dictionary.

Inputs:

datatype – If the data is of 'float' type then datatype = 'float' else by default the datatype is set to 'double'.

Outputs:

A dictionary with the following keywords -

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 after reading the grid.out.

Outputs:

This function outputs a dictionary with following keys.

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

Usage:

```
import pyPLUTO as pp
D = pp.pload(1)
G = D.grid()
```

Now, G is a dictionary such that G['x1'] will give the x-array. **NOTE** - Even x1 is set as attribute for the pyPLUTO.pload object i.e., the x-array can be obtained as D.x1. Infact all the keys of this routine are set as attributes for the pyPLUTO.pload object D.

time_info(*datatype*)
This method returns a dictionary that has the time information for the step ns.

Inputs:

datatype – If the data is of 'float' type then datatype = 'float' else by default the datatype is set to 'double'.

Outputs:

A dictionary which has a following keys -

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.

2.3 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.

Inputs:

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

Outputs:

A list with 2 elements where the first element corresponds to the interpolate field point in 'x' direction and the second element is the 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).

Inputs:

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

Outputs:

This routine returns a dictionary with keys -

qx – list of the field points along the 'x' direction. qy – list of the field points along the 'y' direction.

Usage:

See the myfieldlines routine for the same.

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

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

Inputs:

Data – pyPLUTO.pload object

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

datatype – If the data is of 'float' type then datatype = 'float' else by default the datatype is set to 'double'.

Optional 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]).

Inputs:

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)

Optional 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.

Usage:

Assume that the magnetic field is a given as **B** = B0$hat{y}$. Then to show this field lines we have to define the x and y arrays of field foot points.

```
x0arr = linspace(0.0,10.0,20)
y0arr = linspace(0.0,0.0,20)
import pyPLUTO as pp
D = pp.pload(45)
I = pp.Image()
I.myfieldlines(D,x0arr,y0arr,colors='k',ls='--',lw=1.0)
```

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

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

Inputs:

var – 2D array that needs to be displayed.

Required Keywords:

x1 – The 'x' array

x2 – The 'y' array

Optional Keywords:

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*, *datatype=None*, ***kwargs*)

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

Inputs:

Data – pyPLUTO.pload object

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

datatype – If the data is of 'float' type then datatype = 'float' else by default the datatype is set to 'double'.

Required Keywords:

plvar – A string which represents the plot variable.

Optional Keywords:

logvar – [Default = False] Set it True for plotting the log of a variable. rphi – [Default = False - for plotting in r-theta plane] Set it True for plotting the variable in r-phi plane.

2.4 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.

Inputs:

u1 – 2D vector along x1 whose divergence is to be determined.

u2 – 2D vector along x2 whose divergence is to be determined.

x1 – The 'x' array

x2 – The 'y' array

dx1 – The grid spacing in 'x' direction.

dx2 – The grid spacing in 'y' direction.

geometry – The keyword *geometry* is by default set to 'cartesian'. It can be set to either one 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.

Outputs:

A 2D array with same shape as u1(or u2) having the values of divergence.

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

This method calculates the gradient of the 2D scalar phi.

Inputs:

phi – 2D scalar whose gradient is to be determined.

x1 – The 'x' array

x2 – The 'y' array

dx1 – The grid spacing in 'x' direction.

dx2 – The grid spacing in 'y' direction.

polar – The keyword should be set to True inorder to estimate the Gradient in polar co-ordinates. By default it is set to False.

Outputs:

This routine outputs a 3D array with shape = (len(x1),len(x2),2), such that [:,:,0] element corresponds to the gradient values of phi wrt to x1 and [:,:,1] are the gradient values of phi wrt to x2.

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

This method does the transformation from spherical coordinates to cylindrical ones.

Inputs:

R - 2D array of spherical radius coordinates.

Th - 2D array of spherical theta-angle coordinates.

X1 - 2D array of radial component of given vector

X2 - 2D array of thetoidal component of given vector

Outputs:

This routine outputs two 2D arrays after transformation.

Usage:

```
import pyPLUTO as pp
import numpy as np
D = pp.pload(0)
```

```
ppt=pp.Tools()

TH,R=np.meshgrid(D.x2,D.x1)

Br,Bz=ppt.RTh2Cyl(R,TH,D.bx1,D.bx2)
```

D.bx1 and D.bx2 should be vectors in spherical coordinates. After transformation (Br,Bz) corresponds to vector in cilindrical coordinates.

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.

Inputs:

a – The 2D array that needs resampling into new dimensions.

newdims – A tuple which represents the shape of the resampled data

method – This keyword decides the method used for interpolation.

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 – This keyword decides the positions of interpolation points.

True - interpolation points are at the centres of the bins

False - points are at the front edge of the bin

minusone – This prevents extrapolation one element beyond bounds of input array

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)

Outputs:

A 2D array with resampled data having a shape corresponding to newdims.

deriv(*Y*, *X=None*)

Calculates the derivative of Y with respect to X.

Inputs:

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.

Outputs:

This returns an 1-D array having the same no. of elements as Y (or X) and contains the values of dY/dX.

getUniformGrid(*r*, *th*, *rho*, *Nr*, *Nth*)

This method transforms data with non-uniform grid (stretched) to uniform. Useful for stretched grid calculations.

Inputs:

r - 1D vector of X1 coordinate (could be any, e.g D.x1).

th - 1D vector of X2 coordinate (could be any, e.g D.x2).

rho- 2D array of data.

Nr - new size of X1 vector.

Nth- new size of X2 vector.

Outputs:

This routine outputs 2D uniform array Nr x Nth dimension

Usage:

```
import pyPLUTO as pp
import numpy as np
D = pp.pload(0)
ppt=pp.Tools()
X1new, X2new, res = ppt.getUniformGrid(D.x1,D.x2,D.rho,20,30)
```

X1new - X1 interpolated grid len(X1new)=20 X2new - X2 interpolated grid len(X2new)=30 res - 2D array of interpolated variable

myInterpol(*RR*, *N*)

This method interpolates (linear interpolation) vector 1D vector RR to 1D N-length vector. Useful for stretched grid calculations.

Inputs:

RR - 1D array to interpolate.

N - Number of grids to interpolate to.

Outputs:

This routine outputs interpolated 1D array to the new grid (len=N).

Usage:

```
import pyPLUTO as pp
import numpy as np
D = pp.pload(0)
ppt=pp.Tools()
x=linspace(0,1,10) #len(x)=10
y=x*x
Ri,Ni=ppt.myInterpol(y,100) #len(Ri)=100
```

Ri - interpolated numbers; Ni - grid for Ri

sph2cyl (*D*, *Dx*, *rphi=None*, *theta0=None*)

This method transforms spherical data into cylindrical applying interpolation. Works for stretched grid as well, transforms poloidal (R-Theta) data by default. Fix theta and set rphi=True to get (R-Phi) transformation.

Inputs:

D - structure from 'pload' method.

Dx - variable to be transformed (D.rho for example).

Outputs:

This routine outputs transformed (sph->cyl) variable and grid.

Usage:

```
import pyPLUTO as pp

import numpy as np

D = pp.pload(0)

ppt=pp.Tools()

R,Z,res = ppt.sph2cyl(D,D.rho.transpose())
```

R - 2D array with cylindrical radius values Z - 2D array with cylindrical Z values res - 2D array of transformed variable

CHAPTER

THREE

PYPLUTO MODULE.

The pyPLUTO module can be loaded as follows.

```
import pyPLUTO as pp
```

The functions associated with this module are listed below :

pyPLUTO.**get_nstepstr**(*ns*)

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

Inputs:

ns – Integer number that represents the time step number. E.g., The ns for data.0001.dbl is 1.

Outputs:

Returns the string that would be used to complete the data file name. E.g., for data.0001.dbl, ns = 1 and pyPLUTO.get_nstepstr(1) returns '0001'

pyPLUTO.**nlast_info**(*w_dir=None*, *datatype=None*)

Prints the information of the last step of the simulation as obtained from dbl.out or flt.out

Inputs:

w_dir – path to the directory which has the dbl.out(or flt.out) and the data

datatype – If the data is of 'float' type then datatype = 'float' else by default the datatype is set to 'double'.

Outputs:

This function returns a dictionary with following keywords -

nlast – The ns for the last file saved.

time – The simulation time for the last file saved.

dt – The time step dt for the last file.

Nstep – The Nstep value for the last file saved.

Usage:

In case the data is 'float'.

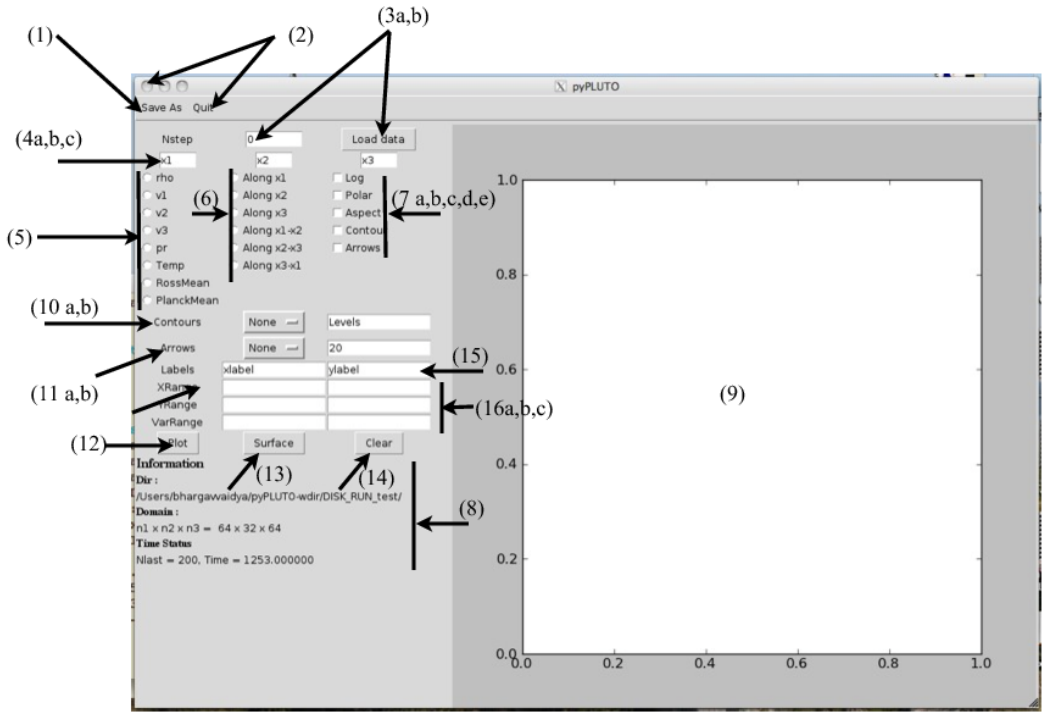
```
wdir = /path/to/data/directory
import pyPLUTO as pp
A = pp.nlast_info(w_dir=wdir,datatype='float')
```

CHAPTER

FOUR

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.

Usage :

On installing the pyPLUTO source code via local install process, the executable for the GUI version of the code will be present in `<path to MyPython_Modules>/bin`. In order to use directly from the command line append the `$PATH` variable as follows:

`export PATH=<path to MyPython_Modules>/bin:$PATH`, after which data can be visualized using following commands in the directory which has the relevant data -

`GUI_pyPLUTO.py` - This will work in case the data is in double format.

`GUI_pyPLUTO.py --float` – For data in float format one has to use the flag *–float*

The various functionalities in the Graphic User Interface (GUI) of the pyPLUTO code are marked with numbers as shown in the figure and 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'. **NOTE** - In case of surface images, it is recommended to save as 'png' and/or 'jpg' and then convert as the way the code plots surface is using pcolormesh (to account for non-uniform grid) and its been a known fact that rendering of pcolormesh images in eps takes loads of time.

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* (or *flt.out*) are enlisted in this column. The code will have a disabled *Plot* and *Surface*, until a variable is chosen in this column. **NOTE** - If the user wish to plot a userdef variable and also saves the vector potential then it is recommended to save data as multiple_files instead of single_file in pluto.ini.

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*. In case 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

FIVE

INDICES AND TABLES

- *genindex*
- *search*

PYTHON MODULE INDEX

p

INDEX