PyWIP User's Guide

Nicholas Chapman

9 July 2013

Contents

1 Introduction

Around the time I was supposed to be writing my thesis (way back in 2007), I decided I was fed-up with a plotting package called WIP. In Astronomy, people generally use IDL, SM, WIP, or Gnuplot for their plotting needs. The first two cost money, a lot of money in the case of IDL. WIP has two main advantages over Gnuplot: it can read and understand FITS/MIRIAD data files with world coordinates, and I know how to use it. In most other ways WIP sucks. Big time.

So, like I said, I was supposed to be writing my thesis, which was going to require a ton of figures, and the thought of doing it all in WIP gave me nightmares. Right then and there, I decided it would be much easier to write a python wrapper around WIP to make it much easier to use. Thus, PyWIP was born.

The only other python plotting package I knew of at the time was matplotlib. I had played with that off and on, but was ultimately unsatisfied by it. It seemed to require too much overhead to make any sort of plot. Furthermore, it took major effort and more overhead to do some simple things like changing the tick marks on the axes. Lastly, matplotlib seemed to run very slowly, even on the simplest of plots. But, your mileage may vary. If you hate PyWIP, give it a try. matplotlib can do many things that PyWIP (and WIP) cannot. However, I do not need any of those features (usually). There is also an astronomy python plotting package based on matplotlib called APLpy.

2 Installation

PyWIP is a python package, so all the code, documentation, and examples are located inside a directory named `pywip`. It is easy to install. You only need two things, WIP and python. You likely already have python installed. If you have a semi-recent version, greater than say, around 2.4, that should be fine. WIP comes from installing MIRIAD.

The most convenient way to install PyWIP is to put it in some permanent location and make that location your PYTHONPATH. For this to work you need to set the PYTHONPATH environment variable, e.g.:

```
setenv PYTHONPATH /Users/nchapman/data1/Applications/nlcpython
```

3 Documentation

PyWIP comes with numerous examples in the `bin` directory. You should be able to run them all and get output postscript plots. Furthermore, you can find this users manual in LaTeX and pdf format in the `docs` directory. Lastly, you can easily create an HTML guide that lists all the commands with brief descriptions of their keywords:

```
pydoc -w pywip
pydoc -w pywip.tools
```

4 Individual Commands

This section will further describe individual commands. It will usually describe 'gotchas' and the like that may cause headaches.

4.1 axis

If you issue the `axis()` command without any arguments, then PyWIP will attempt to guess the appropriate axis to draw. This means if you set the `logx` or `logy` keywords in any of your plots, then the appropriate x- or y-axis will be labeled logarithmically. Furthermore, if you plotted an image with `head='rd'`, then PyWIP will label the axes with world coordinates. Finally, when you are plotting multiple panels that join, PyWIP will be smart enough to suppress number labeling where it would interfere with an adjacent panel.

This command isn't perfect, but it's good enough for Rock-and-Roll. Give it a whirl, and if it doesn't do it the way you like, then you can fall back on specifying some or of all the parameters manually as needed. Table 1 lists the allowed parameters.

4.2 blowup

This command is somewhat tricky. The input coordinates MUST be pixel values. I usually determine them from DS9. Secondly, if you have plotted only a sub-region of an image with halftone/contour, then you need to input the relative pixel values. For example, I plotted the image 'image.fits[172:194,136:158]'. Later, I wanted to draw a blow-up box determined by the pixel coordinates 181.156,183.018,145.567,146.802 (in the xmin, xmax,

Table 1: Keywords for the `axis()` command

Keyword	Meaning
box	Specifies the frame to be drawn. Defaults to ('bottom','left','top','right').
number	Where to put the numbers. Defaults to ('bottom','left') but `axis()` will attempt to change these depending if you have multiple panels. The allowed strings are: 'left', 'right', 'bottom', and 'top'. If you don't want any numbers, set this to ().
tickstyle	Style for tickmarks. Note, to control whether tickmarks are actually shown, see the `subtick` and `majortick` keywords below. The default style is ('inside','inside'), meaning the tickmarks for (x,y) axes are drawn inside of the frame. Either string can be set to 'inside', 'outside', or 'both'.
format	Format for the numbers. The default is ('auto','auto') for the (x,y) axes. If you are plotting an image, these are automatically changed to ('wcs','wcs'). Generally, you should never have to set this, but the other options are 'dec' to force decimal labeling, and 'exp' to force exponential labeling.
xinterval	A tuple of two numbers, where the first number is the distance between major xtick intervals, and the second number is the number of subticks between each major xtick. If you leave this unset, reasonable defaults will be chosen.
yinterval	A tuple of two numbers, where the first number is the distance between major ytick intervals, and the second number is the number of subticks between each major ytick. If you leave this unset, reasonable defaults will be chosen.

Table 1: Keywords for the `axis()` command (continued)

Keyword	Meaning
drawtickx, drawticky	If set to False, this is a shortcut to setting `subtickx=False` and `majortickx=False`. Note, setting `drawtick=False` while also setting the `majortick` or `subtick` keywords to True may cause weird behavior. You have been warned.
firstx, firsty	If set to False, write only the last part of the label for the first time tick on the axis. For example, if the full first label is 17 42 34.4, then only write 34.4. This option is useful for sub-panels that join each other. The default values are True.
gridx, gridy	Draw a grid of lines at the major tick positions for either x or y. Defaults are False, meaning no grid is shown.
logx, logy	Label x or y axis logarithmically. Defaults are set automatically depending on whether the user has specified log labeling with plot() or some other command. Note that setting this parameter will override any settings for xinterval and yinterval.
subtickx, subticky	Draw subticks? Defaults to True, meaning do draw subticks for both the x and y axes.
majortickx, majorticky	Draw majorticks? Defaults to True, meaning do draw major ticks for both the x and y axes.
verticaly	Rotate the numbers on the y axis to be vertical? The default is False, but it is set to True automatically if an image has been plotted. There is not equivalent command for the xaxis numbers.
zerox, zeroy	If set to False, omit leading zero in numbers < 10 in time labels. This option is only valid when format is 'wcs' for the given axis. Furthermore, it is ignored when verticaly=True because it becomes impossible to align the numbers properly. The default values are True.

ymin, ymax format). However, I could not enter these values for blowup() because they are incorrect. The correct values would be: 10.156, 12.018, 10.567, 11.802. (181.156 - 171, 183.018 - 171, 145.567 - 135, 146.802 - 135). I subtracted 171 and 135 for x and y respectively, because pixels in FITS images are counted from 1 not zero.

4.3 compass

A kludgy `compass()` command has been added which should allow one to plot the directions of galactic ℓ and b on an RA/DEC plot. It hasn't been well-tested.

4.4 legend

The legend command is used to make it easier to put legends on your plots. A legend is essentially a series of rows, where each row has two elements: a symbol or line representing the data set or curve and a text label. After vast amounts of work, I think I finally succeeded in making a general purpose `legend()` command that doesn't require lots of fiddling by the user. WIP does not come with such a command, so you should be awed by this achievement. Just give it x and y coordinates for the upper-left corner (given as a fraction of the plot width and height) and you should be good to go. Maybe someday I will get around to adding the option to have a box drawn around the legend automatically. The only caveat with `legend()` is that it should be the last thing plotted (well, I think `xlabel`,`ylabel`, and `title()` are okay).

Also note, the curves listed by `legend()` are accumulated for all panels as plots are made until a `legend()` command is issued. Then the list of plots is reset. By default, plots are not shown in the legend because they have `text=None` keyword.

4.5 vector

The `vector` command is used to draw arrow fields. So that the angles and lengths will always make sense when plotting arrows on a FITS image with WCS, internally `vector()` will reset the header to 'px' (absolute pixel value). What this means for the user is that `vector()` should be the last thing added to a plot (well, maybe before `legend()`, I'm not sure about that). Also, it means that you should always specify the x and y as pixels, not

wcs coordinates. Otherwise, your positions, lengths, and angles will be all wrong. Finally, you can use the `align` keyword to specify whether the input coordinates are the left (end), center, or right (tip) of the arrows.

You can also plot polarization 'sticks' (vectors without arrowheads) by setting `taper=0`.

5 Keywords

5.1 color,font,size,style,width

A number of the commands have some or all of these keywords as options. If some or all of these aren't specified, the default values for these will be used. The default color is black, font is roman, size is 1, style is solid lines, and width is 1. You can change the defaults at any time by running the `default` command. See Tables 2, 3, and 4 for details.

RGB colors can be used instead of the built-in defaults. This hasn't been fully tested and debugged, but seems to work well. To specify an rgb color, do it like so: `color='0,0.5,0'` which is 50% green, and no red or blue.

5.2 image

The `contour`, `halftone`, and `winadj` commands all have the `image` keyword. `image` is a string specifying the FITS/MIRIAD image name, and optionally the sub-image to plot. For example:
`image='Bob.fits'` Plot the entire image 'Bob.fits'
`image='Bob.fits[10:20,5:15]'` Plot 'Bob.fits' but only the subimage of pixels 10-20 in the x and 5-15 in the y. Finally, `image='Bob.fits(1)'` will plot plane 1 from 'Bob.fits'. You can combine the subimage and plane definitions in either order.

There are several other 'gotchas' to watch out for when plotting images. See § 7.3 for more details on these.

5.3 limits

The limits keyword can be have several different values. First, if you do not specify limits, then PyWIP will attempt to guess what the limits should be. If no limits have been set for the current panel, then PyWIP will use

Table 2: Valid Colors

Key	Description
w	white
k	black (default)
r	red
g	green
b	blue
c	cyan
m	magenta
y	yellow
o	orange
gy	green-yellow
gc	green-cyan
bc	blue-cyan
bm	blue-magenta
rm	red-magenta
dg	dark gray (approx gray65)
lg	light gray (aprox gray35)
gray1-100	Various shades of grayscale (1 is white, 100 is black)
r,g,b	rgb color as fractions from 0-1

Table 3: Valid Fonts

Key	Description
sf	sans-serif
rm	roman (default)
it	italics
cu	cursive

Table 4: Line Styles

Key	Description
-	solid (default)
--	dashed
.-	dot-dashed
:	dotted
-...	dash-dot-dot-dot

the min/max of the dataset for the current limits. Otherwise, the currently existing limits will be reused. Second, you can give a list/tuple of four values, which will become the new limits and override any previously existing ones. Third, you can set `limits='last'`, meaning PyWIP should use the last known limits (generally from the previous panel) as the new limits. Note, this will also carry over settings for `'logx'` and `'logy'`.

The fourth, and last possibility is to explicitly set `'limits=None'`. I haven't fully explored the ramifications of this, but it seems to force PyWIP to set new limits based on the min/max of the dataset or image, even if limits were already known for the current panel. I use it in `example11.py` to ensure that the blowup box is not resized to the limits of the entire region. I think it would work for non-image type plots as well.

5.4 palette

The `halftone()` command allows a palette to be a filename which contains an RGB lookup table. I had some grandiose plans when I first implemented this, but in reality it is quite limited. I think PGPLOT has a limit of 255 color levels, so you cannot make a large-scale RGB plot. This lookup file needs to have four columns given as red,green,blue,scol. The red, green, and blue values must be given as fractions between 0 and 1. The same is true for scol, which defines fractions of the maximum intensity. Linear interpolation for values in the image between specified levels will be performed.

To make a proper RGB plot, you would need to create a fake image (no larger than 16×16, and give each pixel an increasing intensity. Then, write a lookup table and for each intensity specify what rgb color it should have. You can see why this is a pain in the ass, and shouldn't be done. Easier to

just use DS9.

5.5 x,y

Several commands have the `x` and `y` keywords to specify a location on the plot. This can be given as a number, but it may be more convenient to specify a string in the case of world coordinates. For example, `x='3:30'` would move the cursor to 3 hours 30 minutes. Similarly, `x='52.5'` would also move the cursor to 3 hours 30 minutes since 52.5 degrees is the equivalent. Note the importance of giving the value as a string. Without the quotes, it will move the cursor to 52.5 hour-seconds, not 52.5 degrees, nor 3 hours 30 minutes. PyWIP assumes that `x` is for Right Ascension and `y` is for Declination.

6 Other Tools

PyWIP includes two additional commands located in the tools subpackage. They are `imextract` and `levelgen`. These tools require `numpy` and `pyfits` to use.

Furthermore, there are several stand-alone programs in the `pywip/bin` directory that may be useful. All of them use `readcmd.py` to read command line arguments. These are described below.

6.1 ds9towip

This program will read contour files created by DS9 and convert them to ones readable by WIP. It will create an output directory, and then inside that a directory a series of files, one for each contour level. Lastly, it will create a WIP file that can be processed using `inputwip()`. This script is useful because DS9's contouring algorithm is superior to WIP's in some circumstances (avoiding null pixels at the edges of maps).

6.2 qplot

`qplot.py` is a quick plotting script that can read a file from the command line and implements many common plotting commands. The idea is that if you just want to do a simple plot of your data, you could use `qplot.py` instead of writing a python script.

6.3 rotatevector

Because of projection effects, north on your map may not point exactly vertical. However, polarization vectors are usually defined as an angle east of north. Thus, `rotatevector.py` will rotate the vectors so that their angles will correspond to degrees east of vertical. This should never change your vector angles by more than a degree or two at most, so this is a really minor effect.

6.4 tickmarks

One of the many limitations of WIP is that it doesn't properly handle world coordinates for large regions on the sky. To fix this, I wrote `tickmarks.py`. You can use it to generate a file of raw WIP commands which can be used in your script with the `inputwip()` command. `tickmarks.py` manually draws the tickmarks and labels, and does a pretty good job of guessing things. The only drawback is that you have to manually add the label for the first tickmark along the x axis by editing the output WIP file created by `tickmarks.py`. It is pretty easy to do. Just scroll down to where the RA labels are defined.

6.5 worldpos

When I first wrote this, I was ecstatic. At the time it was the only pure python implementation of pixel $\leftrightarrow$ world coordinate conversion. You could still use it for that purpose if you like, and in fact `tickmarks.py` and `rotatevector.py` do so. However, there is now an "official" python package, `pywcs` that does the same thing and is more sophisticated about handling rotations.

7 Tips & Tricks

7.1 Using hidden variables

Say you want to make a series of plots in a for loop, and have each plot be a different color. You don't need to manually type in all the colors. Instead, they are part of a hidden variable that can be imported. Hidden variables the user may want to access are listed in Table 5. They can be imported and

Table 5: Hidden Variables

Variable	Meaning
_palettes	The palettes used for the `halftone` command
_colors	The valid colors for fonts, symbols, lines, etc. (see Table 2)
_fills	The available fill styles
_fonts	The available fonts (See Table 3)
_lstyles	The available line styles (Table 4)
_symbols	The available symbols to plot points

used like so:

```
from pywip import _colors as pycolors
```

7.2 Multi-Variable Axes

To plot one variable (with numbers and tickmarks) on the bottom and a second variable on the top (with a different set of numbers and tickmarks), you have to use the `style=None` keyword in the `plot()` command. This generates a phantom plot. PyWIP goes through the motions of setting limits, but won't actually plot any points. Then, just plot the new axis: `axis(box=['top'],number=['top'])`. You will also have to change the axis command for the first plot so that it doesn't interfere with the second `axis()` command, e.g. `axis(box=['bottom','left','right'])`.

I did this, and it seems like it would work, so long as variable two is directly proportional to variable1. For example, $var2 = 3 \times var1$ should be okay. However, I had a case where $var2 = 1/var1$, and that did NOT work. I think it is a problem with WIP however.

7.3 Futzing with images

WIP is particular about the images it will accept. Images in PyWIP will always have x increasing to the right and y increasing upwards. If you are plotting an image with a world coordinate system (WCS), it is up to you to make sure the image is rotated properly. In astronomy you typically want Declination upwards and Right Ascension increasing to the left. You may need to rotate your image with IRAF's `rotate` command or the `regrid` task

in Miriad to achieve the correct rotation. If you neglect to do this, PyWIP will still do something, but the WCS will probably be wrong.

Secondly, at least one world coordinate projection (gls) is not allowed. If you get an error about "unsupported projection" then you may need to fix your image. This can be done with the Miriad task `regrid`. Thanks to Jorge Pineda for this tip:

```
Assume you have file.fits
fits in=file.fits op=xyin out=file.mir
regrid in=file.mir out=file_tan.mir project=tan
fits in=file_tan.mir op=xyout out=file_tan.fits
```

Lastly, for very large fields, I think the WCS displayed by WIP will be off. This can easily be seen if you plot the ra/dec, then plot their equivalent x/y positions on an image (get the x/y through something like `sky2xy` or `ds9`). They will *not* line up.

7.4 World Coordinates from a data file

When reading from files, WIP needs ra/dec to be in hour-seconds and degree-seconds (ra * 240, when ra is in degrees, and dec * 3600 when dec is in degrees).

8 The Future

PyWIP is starting to show its age. It is based on PGPLOT and WIP, which means many of the underlying problems cannot be fixed. Since I first started PyWIP, I have learned `numpy` and `pyfits`, and use them both extensively, which eliminates any of the speed advantage of PyWIP over matplotlib. But, I still think the syntax of PyWIP rules, so I am learning matplotlib and implementing wrappers around their commands to make them easier to use. All in all, the need for PyWIP is less than it used to be. As a result, this is the last version of PyWIP. It is possible I made make some bug fixes in the future, but there are no guarantees

For historical reasons, here is my final list of PyWIP limitations:

- The colors should allow hexcodes, grayscale ranges, and rgb values (partially fixed)

- The axis() command should control logx, logy, and plot range limits
- Cannot make shaded contour plots
- Cannot make translucent fill styles (I don't think PGPLOT allows transparency)
- Only a limited number of filled symbols are available
- The only output formats are .ps and .gif. (PGPLOT problem again)
- The legend() command is quite nice, but is still a kludge
- No ability to make RGB plots like DS9 (limited to 255 levels)
- WCS projections do not seem correct for large regions
- No way to vertically align text (in a left, center, right format)
- Doesn't understand numpy arrays
- you should be able to specify whether a given set of coordinates are viewport, wcs, or whatever.
- filled histograms
- an easy scalebar (i.e. draw a line and specify distance in pc, whatever)