Pygplot

API Documentation

February 17, 2005

Contents

1 Module pygplot

Pygplot: A Python Interface to Pgplot

Version: 0.91

Author: Chris Burns

Requires: Pgplot (*http://www.astro.caltech.edu/~tjp/pgplot/*)

You can get the source for pygplot here: *http://astro.swarthmore.edu/~burns/pygplot_downloads*.

There is the beginnings of a manual here: *http://astro.swarthmore.edu/~burns/pygplot.pdf* .

This python module provides an object-oriented interface to the Pgplot library of routines. Most (though not all) of the functionality of Pgplot is included:

1. 1-D plots using lines, symbols and error-bars
2. 2-D plots using color (or gray-scale) images and contours
3. Access to all the pgplot primitives through the ppgplot extension module.

1.1 Functions

columns(*filename*)

Read in the specified file and try to extract columns. So far, it will recognise space-, comma-, tab-, or semi-colon-delimited columns.

Parameters

`filename`: the name of the file to read
(type=string)

Return Value

a 2-D array: one row for each column in the file

Linear(*low*, *high*, *n*)

Convenience function for computing *n* numbers linearly interpolated between *low* and *high*. Used as the default function for the *cfunc* parameter to the *contour* function.

list_devices()

Print out the list of available devices.

list_palettes()

Print the currently installed palettes to the terminal.

pgspline(*xp*, *yp*, *x*, *slope0*=`None`, *slope1*=`None`)

Compute the cubic spline for a set of data

Parameters

`xp:`	An array of x-coordinates for the control points. *(type=float array)*
`yp:`	An array of y-coordinates for the control points. *(type=float array)*
`x:`	An array of x-coordinates where you want interpolations to be done *(type=float array)*
`slope0:`	Optionally specify the slope at the beginning of the interval *(type=float)*
`slope1:`	Optionally specify the slope at the end of the interval *(type=float)*

1.2 Variables

Name	Description
descriptions	**Value:** `{'/NULL': ['(Null device, no output)', 0], '/VG-IF': ['(Graphics Interchange F...` *(type=`dict`)*
devices	**Value:** `['/GIF', '/VGIF', '/NULL', '/PPM', '/VPPM', '/P-S', '/VPS', '/CPS', '/VCPS', '...` *(type=`list`)*
n_devices	**Value:** `15` *(type=`int`)*

1.3 Class MPlot

Class used to create a multi-panel plot with separate bounding boxes. This is in the traditional sense, where a page is split into (n x m) panels and each panel has a full plot: box, axes, tick lables, axis labels, titles, etc. Required parameters are the number of columns and rows. The user then adds Plot instances which will be ploted in the sub-panels in order.

Example:

```
>>> M = MPlot(2,2,device="test.ps/PS")    # create a 2 x 2  multi-plot
>>> M.add(plot1)     # plot1, plot2, plot3 and plot4 are Plot
>>> M.add(plot2)     #  instances
>>> M.add(plot3)
>>> M.add(plot4)
>>> M.plot()
>>> M.close()
```

1.3.1 Methods

__init__(*self*, *nx*, *ny*, *device*=`'/XWINDOW'`, *aspect*=`None`)

Initialization routine for the MPlot class.

Parameters

`nx:`	number of columns *(type=int)*
`ny:`	number of rows *(type=int)*
`device:`	Pgplot device to use (default: 1st available device) *(type=string)*
`aspect:`	The aspect ratio (height/width) (default: device dependent). *(type=float)*

add(*self*, *plot*)

Add a Plot instance to the MPlot. The instances are plotted in the order they are added, filling each row and then switching to the next.

close(*self*)

Close the pgplot device corresponding to the MPlot.

plot(*self*)

Plot the Mplot. Each sub-panel is plotted by calling that instance's plot() method after calling pgpage() to advance to the next panel.

select(*self*, *i*=`0`, *j*=`0`)

Make the instance's device the current device and select the given panel.

Parameters

`i:`	which column to select (default: 0) *(type=int)*
`j:`	which row to select (default: 0) *(type=int)*

1.3.2 Class Variables

Name	Description
nx	**Value:** `1` *(type=`int`)*
ny	**Value:** `1` *(type=`int`)*

1.4 Class Panel

Class used to create a multi-panel plot with a single bounding box. Unlike MPlot, these panels are made to look like one plot with contiguous sub-plots. For example, if the plot were split into 3 x 3 panels, the central panel would have no labels outside it's box, which would abutt the sides of the adjoining panels. Other than this property, this class works the same as MPlot.

1.4.1 Methods

__init__(*self*, *nx*, *ny*, *device*=`'/XWINDOW'`, *aspect*=`None`)

Initialization routine for the MPlot class.

Parameters

`nx:`	number of columns *(type=int)*
`ny:`	number of rows *(type=int)*
`device:`	Pgplot device to use (default: 1st available device) *(type=string)*
`aspect:`	The aspect ratio (height/width) (default: device dependent). *(type=float)*

add(*self*, *plot*)

Add a Plot instance to the MPlot. The instances are plotted in the order they are added, filling each row and then switching to the next.

close(*self*)

Close the pgplot device corresponding to the MPlot.

plot(*self*)

Plot the Mplot. Each sub-panel is plotted by calling that instance's plot() method after calling pgpage() to advance to the next panel.

select(*self*, *i*=`0`, *j*=`0`)

Make the instance's device the current device and select the given panel.

Parameters

`i:`	which column to select (default: 0) *(type=int)*
`j:`	which row to select (default: 0) *(type=int)*

1.4.2 Class Variables

Name	Description
nx	**Value:** `1` *(type=`int`)*
ny	**Value:** `1` *(type=`int`)*

1.5 Class Plot

Class to make a single 2-D plot. Use this class to produce a single frame plot, with axes, lines, points, etc. See constructor for options.

Example:

```
>>> plot = Pgplot.Plot(device='test.ps/PS', xrange=[0,1], yrange=[0,10])
>>> plot.point(x,y)
>>> plot.line(x,y)
>>> plot.plot()
>>> plot.close()
```

1.5.1 Methods

__init__(*self*, ***kw*)

Initialization routine. The Plot class has the following options which can be specified:

- **aspect**: (*float*) The aspect ratio (h/w). Defaults to device's default
- **color**: (*int/string*) Color with which objects are drawn (lines, symbols, etc) Defaults to foreground color.
- **device**: (*string*) Pgplot device to which the plot is sent. Default: First available interactive device
- **font**: (*int*) Font with which to print labels. Default: 1
- **fsize**: (*float*) Font size with which to print labels. Default: 1.0
- **id**: (*bool*) Print the user ID? Default: 0
- **linestyle**: (*int*) Line style with which to draw data lines. Default: 1
- **linewidth**: (*int*) Line width with which to draw data lines. Default: 1
- **noleftlabel**: (*bool*) Suppress first x-axis tick label? Default: 0
- **norightlabel**: (*bool*) Suppress last x-axis tick label? Default: 0
- **notoplabel**: (*bool*) Suppress last y-axis tick label? Default: 0
- **nobottomlabel**: (*bool*) Suppress first y-axis tick label? Default: 0
- **symbol**: (*int*) Symbol with which to plot points. Default: 1
- **ticks**: (*string*) Draw ticks 'in', 'out', or 'both'? Default: 'in'
- **title**: (*string*) Title of the plot. Default: None.
- **width**: (*string*) Width with which to draw all other lines besides the data lines. Default: 1
- **[x|y]axis**: (*bool*) Draw the [x|y] axis ([y|x]=0)? Default: 0
- **[x|y]format**: (*string*) Format of the [x|y]-axis labels. 'hms' produces 'hh:mm:ss' format, 'dms' produces 'dd:mm:ss' and 'dec' is the usual decimal format. Default: 'dec'.
- **[x2|y2]format**: (*string*) Format of the secondary [x|y]-axis labels. 'hms' produces 'hh:mm:ss' format, 'dms' produces 'dd:mm:ss' and 'dec' is the usual decimal format. Default: 'dec'.
- **[x|y]grid**: (*int/string*) Whether to plot the grid lines for the x or y axes and what color to use. Set x/ygrid to the color (string or integer) you want plotted. Default: no grid is drawn.
- **nl[xy]l** (*boolean*) no leading [xy] label on the axis if using "hms" or "dms" format. Default: 0
- **[x|y]log**: (*bool*) Use logarithmic [x|y] axis? Default: 0
- **x2log**: (*bool*) Use logarithmic [x2|y2] axis? Default: 0
- **[x|y]range**: (*2-tuple*) [x|y] range. Default: auto-range.
- **[x|y]2range**: (*2-tuple*) [x|y] range of second axis. Default: auto-range.
- **[x|y]label**: (*string*) Label on [x|y] axis. Default: None
- **[x|y]2label**: (*string*) Label on second [x|y] axis. Default: None

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

axis(*self*, *x1*, *y1*, *x2*, *y2*, *v1*, *v2*, *log*=`0`, *label*=`0`, *dec*=`0`, *exp*=`0`, *step*=`0`, *nsub*=`0`, *dmajl*=`0.5`, *dmajr*=`0.5`, *fmin*=`0.5`, *disp*=`0.29999999999999999`, *orient*=`0.0`, *fsize*=`None`, *font*=`None`, *linestyle*=`None`, *linewidth*=`None`, *color*=`None`)

Add an alternate axis to the plot. This does not alter the viewport nor change the way points are plotted; it is merely a convenient way to add another axis.

Parameters

`x1:`	x world coordinate of the beginning of the axis *(type=float)*
`y1:`	y world coordinate of the beginning of the axis *(type=float)*
`x2:`	x world coordinate of the end of the axis *(type=float)*
`y2:`	y world coordinate of the end of the axis *(type=float)*
`v1:`	value of the axis at (x1,y1) *(type=float)*
`v2:`	x value of the axis at (x2,y2) *(type=float)*
`step:`	major tick spacing along the axis. *(type=float)*
`nsub:`	number of minor ticks per major tick interval *(type=int)*
`dmajl:`	length of the major tick to the left of the axis in units of character height *(type=float)*
`dmajr:`	length of the major tick to the right of the axis in units of character height *(type=float)*
`fmin:`	length of the minor ticks as fraction of the major tick. *(type=float)*
`disp:`	displacement of numeric labels away from the axis. In units of character height. *(type=float)*
`orient:`	angle at which to print labels, relative to the axis. (default 0.0) *(type=float)*
`fsize:`	font size to use (defaults to the Plot's global value) *(type=float)*
`font:`	font with which to print the label (defaults to the Plot's global value) *(type=int)*
`linestyle:`	line style to use (defaults to the Plot's global value) *(type=int)*
`linewidth:`	line width to use (defaults to the Plot's global value) *(type=int)*
`color:`	Color with which to print the label (defaults to the Plot's global value) *(type=string/int)*

close(*self*)

Close the plot device. Only effective if the Plot instance is not part of a MPlot or Panel instance.

color_point(*self*, *x*, *y*, *z*, *symbol*=`None`, *size*=`None`, *palette*=`'real'`, *transfer*=`'linear'`)

Add a data set that will be drawn with symbols. Unlike the usual point, here the value of z is used to determine an intensity of each point and, based on the chosen palette, this is drawn with differen colors.

Parameters

`x:`	array of x world coordinates *(type=numarray)*
`y:`	array of y world coordinates *(type=numarray)*
`z:`	array of intensities for the x,y points *(type=numarray)*
`symbol:`	symbol to use (defaults to parent class default) *(type=int)*
`size:`	size of symbols *(type=float)*
`palette:`	color palette to use. See list_palettes(). *(type=string)*
`transfer:`	transfer function (intensity) to use. See list_palettes(). *(type=string)*

contour(*self*, *data*, ***kw*)

Contour plot data. The only required argument is a 2 dimensional array. The rest of the arguments are as follows:

- **clab**: (*bool*): Draw contour labels? Default: 0
- **clabc**: (*int/string*) Color for the contour labels. Defaults to foreground color.
- **clabi**: (*int*) Spacing along the contour between labels in grid cells.
- **clabm**: (*float*) Contours that cross less than clabm grid cells will not be labeled.
- **color**: (*int/string*) Color of the contours.
- **contours**: (*numarray*) The contour levels to draw. If not specified, then they will be generated automatically based on 'low', 'high', 'ncontours', and cfunc (see below)
- **cfunc**: (*function*) A function that returns contour levels. The function will be run by plot command automatically if the contours aren't specified explicitly. The function takes 3 arguments: the value of the low contour, the value of the high contour and the number of contours requested. The default is Linear(), a function that gives ncontour contours equally spaced between low and high.
- **font**: (*int*) Font with which to print labels. Default: inherited from the Plot class.
- **fsize**: (*float*) Font size with which to print labels. Default: inherited from the Plot class.
- **high**: (*float*) The largest pixel value to contour.
- **linestyle**: (*int*) Line style with which to draw contours. Default: inherited from Plot class.
- **linewidth**: (*int*) Line width with which to draw contours. Default: inherited from Plot class.
- **low**: (*float*) Lowest pixel value to contour.
- **ncontours**: (*int*) Number of contours to plot (default 10).
- **tr**: (*6-tuple*) The transformation matrix from pixel coordinates to world coordinates. If px and py are the pixel coordinates, then the world coordinates x and y are given by: x = tr[0] + tr[1]*px + tr[2]*py y = tr[3] + tr[4]*px + tr[5]*py This option overrides [x|y]range.
- **[x|y]range**: (*2-tuple*) [x|y] range. The range of the pixel data. This is used to figure out the pixel-to-world coordinate transformaton (see tr above). Default: auto-range.

error(*self*, *x*, *y*, *dx1*=`None`, *dy1*=`None`, *dx2*=`None`, *dy2*=`None`, *linestyle*=`None`, *linewidth*=`None`, *color*=`None`)

Add error bars to the data points. By specifying one or two error limits, you get symmetric or non-symmetric error bars, respectively.

Parameters

`x:`	Array of x-values (midpoints) *(type=float array)*
`y:`	Array of y-values (midpoints) *(type=float array)*
`dx1:`	Array of x error lengths. If dx2 is not specified, errorbars are drawn at +/- dx1, otherwise it is an upper error. *(type=float array)*
`dy1:`	Same as dx1 but an upper limit in the y-direction *(type=float array)*
`dx2:`	Array of lower error limits in the x-direction *(type=float array)*
`dy2:`	Array of lower error limits in the y-direction *(type=float array)*

histogram(*self*, *data*, ***kw*)

Plot a histogram of unbinned data. The bins are computed automatically by the routine from the input data. The other optional arguments are

- **data**: (*float*) Data to contour.
- **linestyle**: (*int*) Linestyle to use when drawing the boxes (defaults to global value)
- **linewidth**: (*int*) Linewidth to use for drawing the boxes (defaults to global value)
- **color**: (*string,int*) The color use (defaults to global color)
- **min**: (*float*) Minimum bin value. Data < bin is not counted. Default: min of data
- **max**: (*float*) Maximum bin value. Data > bin is not counted. Default: max of data
- **nbin**: (*int*) Number of bins. Default: 10
- **fill**: (*int*) Which fill style to use: 1- solid 2- outline 3- hatch 4- cross-hatch. Default: 1

image(*self*, *data*, ***kw*)

Plot a grey-scale or color image representation of data on the background of the plot. The only required argument is data, a 2 dimensional array. The other options are as follows:

- **autocut**: (*float*) If specified, the **low** and **high** will be chosen automatically so that **autocut** percent of the data is included.
- **high**: (*float*) Pixels with values higher than 'high' are drawn with the maxinum value in the color table. Default value is the highest value in the data.
- **low**: (*float*) Pixel values lower than 'low' are drawn with the lowest color in the color table. Default value is the lowest pixel value in the data.
- **palette**: (*string*) The color palette to use. Use the function list_palettes() to see a list of available palettes. If palette is not specified, the default grey-scale will be used.
- **transfer**: (*string*) The transfer function (intensity) to use for the color palette (default is "real" i.e., linear).
- **tr**: (*6-tuple*) The transformation matrix from pixel coordinates to world coordinates. If px and py are the pixel coordinates, then the world coordinates x and y are given by: x = tr[0] + tr[1]*px + tr[2]*py y = tr[3] + tr[4]*px + tr[5]*py This option overrides [x|y]range.
- **[x|y]range**: (*2-tuple*) [x|y] range. The range of the pixel data. This is used to figure out the pixel-to-world coordinate transformaton (see tr above). Default: auto-range.

label(*self*, *x*, *y*, *string*, *angle*=`0`, *just*=`0`, *fsize*=`None`, *font*=`None`, *color*=`None`, *width*=`None`, *vjust*=`0`, *bbox*=`None`)

Add a label to the plot.

Parameters

`x:`	x world-coordinate for the label *(type=float)*
`y:`	y world-coordinate for the label *(type=float)*
`string:`	the label *(type=string)*
`angle:`	angle of the label relative to the x-axis (default: 0) *(type=float)*
`just:`	Justification of the string relative to the (x,y) coordinates: just = 0.0 –> left justified (default), just = 1.0 –> right justified, just=0.5 –> centered. *(type=float)*
`fsize:`	font size to use (defaults to the Plot's global value) *(type=float)*
`font:`	font with which to print the label (defaults to the Plot's global value) *(type=int)*
`color:`	Color with which to print the label (defaults to the Plot's global value) *(type=string/int)*

legend(*self*, *height*=`None`, *width*=`None`, *position*=`'ur'`, *dx*=`None`, *dy*=`None`, *bbox*=`None`, *fsize*=`1.0`, *font*=`1`, *color*=`'white'`)

Draw a legend for the data. You can control the height of the bounding box in world coordinates if you want to, but that will over-ride the font size (fsize has no effect). The position can be ur (upper right), lr (lower right), ul (upper left), or ll (lower left). dx and dy allow you to offset the legend from the axis (positive number brings it inside the graph). If bbox is not None, a bounding box will be drawn.

- **height**: max height of the legend in world coordinates (default: as high as needed given the fsize)
- **widht**: max width of the lengend in world coordinates (default: as wide as needed given fsize
- **position**: vertical and horizontal alignment (ul, ur, ll, lr)
- **dx,dy**: horizontal and vertical offsets from the axis
- **bbox**: boolean: draw a bounding box around the legend and over-write what's underneath?
- **fsize**: font size. This over-rides height and width
- **font**: font to use
- **color**: color for the text

line(*self*, *x*, *y*, *linestyle*=`None`, *linewidth*=`None`, *color*=`None`, *label*=`None`)

Add a data set that are connected with lines

Parameters

`x:`	array of x world coordinates *(type=numarray)*
`y:`	array of y world coordinates *(type=numarray)*
`linestyle:`	Linestyle to use when drawing the line (defaults to parent class default) *(type=int)*
`linewidth:`	Linewidth to use for drawing the line (defaults to parent class default) *(type=int)*
`color:`	The color use (defaults to parent class global) *(type=int/string)*
`label:`	optional label to be used in a legend *(type=string)*

plot(*self*, *just*=`0`)

Plot the plot object. This is called if the Plot instance is not part of a MPlot or Panel. Otherwise, MPlot or Panel will be responsible for calling it. The optional just parameter is used internally.

point(*self*, *x*, *y*, *symbol*=`None`, *size*=`None`, *color*=`None`, *label*=`None`)

Add a data set that will be drawn with symbols.

Parameters

`x:`	array of x world coordinates *(type=numarray)*
`y:`	array of y world coordinates *(type=numarray)*
`symbol:`	symbol to use (defaults to parent class default) *(type=int)*
`size:`	size of symbols *(type=float)*
`color:`	color of the symbols *(type=int/string)*
`label:`	optional label to be used in a legend *(type=string)*

replot(*self*)

Select the plot's device, erase and redraw the plot. Use this in the interactive mode to make changes and replot.

select(*self*)

This will make the instance's device the selected device (and make it's panel the current pane if it is part of an Mplot or Panel). All ppgplot raw commands will then be directed to this plot.

1.5.2 Class Variables

Name	Description
options	**Value:** {} *(type=`dict`)*