pygmyplot Documentation

Release 0.2.0

James Stroud

August 29, 2012

CONTENTS

CHAPTER

ONE

INTRODUCTION

Pygmyplot is a simplified wrapper around matplotlib, designed to make it easier to embed specific types of matplotlib plots into Tkinter applications while retaining matplotlib levels of configurability. The idea behind pygmyplot is similar to that of python megawidgets (PMW), although the interface to pygmyplot is not intended to be similar to PMW. The four types of plots currently supported by pygmyplot are:

- X-Y Plots
- Scatter Plots
- Histograms
- Heat Maps

CHAPTER

TWO

CHANGES

2.1 0.2.6rc

- Updated packaging to use distribute, which allows for proper handling of dependencies with PyPI

2.2 0.2.2rc

- Fixed bug with heatmap labels

2.3 0.2.1rc

- Solidified public API for distribution
- Created documentation in sphinx

2.4 From 0.2.0

- Added cushion for scatter: better limits for scatter plots.

CHAPTER

THREE

API

3.1 Overview

At present, pygmyplot has four stable convenience functions as part of the API. These functions take minimal arguments to produce plots and return instances of `MyPlot` subclasses (e.g. `MyXYPlot`) through which the matplotlib API is accessible. Because of this accessibility, pygmyplot plots are as configurable as matplotlib plots.

Although `MyPlot` subclasses are defined for each type of plot, these classes are not presently part of the official pygmyplot API, so use of these classes directly is not yet supported and may affect the forward-compatibility of your code.

Several examples of pygmyplot usage follow descriptions of the convenience functions.

3.2 Convenience Functions

The four convenience functions currently in the official pygmyplot API are:

- `xy_plot()`
- `scatter()`
- `histogram()`
- `xy_heat()`

In all four functions, the `master` keyword argument designates the Tkinter widget that serves as the container for the resulting plot. Use of the `master` keyword argument is given below in the Examples. Other keyword arguments for these functions will be documented in the future.

`pygmyplot.`**`xy_plot`**(*x*, **args*, ***kwargs*)
: Returns an instance of *MyXYPlot* with the data in *x* and **args* (if supplied) plotted. This function mostly mirrors the behavior of pyplot.plot.

`pygmyplot.`**`scatter`**(*x*, *y*, ***kwargs*)
: Returns an instance of *MyXYPlot* with the data in *x* and *y* plotted. This function mostly mirrors the behavior of matplotlib.axes.scatter.

`pygmyplot.`**`histogram`**(*data*, ***kwargs*)
: Returns an instance of *MyHistogram* with the *data* (a sequence) binned and plotted.

  Use the *bins* keword argument to control the number of bins.

pygmyplot.**xy_heat**(*coords*, *vals*, *coarsen=None*, *heat_up=None*, ***kwargs*)

Returns a heat-map (`MyHeatMap` intance) with the values (`vals`) plotted at the positions indicated by `coords`, which is organized as pairs containing (x,y) positions, e.g:

```
coords = [(0, 0), (0, 1), (1, 0), (1, 2)]
vals = [5, 2, 4, 3]
```

For each (x,y) coordinate of data, the corresponding numerical value is plotted on the heat map at that position. All unfilled positions take the minimum value of the data.

The grid is coarsened by the pair (e.g. a 2-tuple) `coarsen` relative to the (x,y) positions if it is provided. This makes it possible for the (x,y) postions to have a multiplicity (e.g. 0, 4, 8 versus 0, 1, 2 in x, etc.). Thus, `coarsen` should divide evenly into the coordinates.

For example if `coarsen` is `(3, 2)`, then the coordinates could be:

```
[(0,0), (0,2), (0,4), (3,0), (3,2), (3,4), (9,0), (9,2), (9,4)]
```

If the `heat_up` keyword argument is provided, then the data is heated up by that much to make the unfilled values have more contrast. This has the side-effect of reducing the range of colors in the map.

The `**kwargs` are passed directly to the `MyHeatMap` initializer.

3.3 Plots

Plots are instances of `MyPlot` subclasses. The most useful attribute of plots is the `axes` attribute, which exposes the matplotlib.axes API. See the *X-Y Plot* example for how the `axes` attribute is accessed and used to modify the appearance of the plot.

CHAPTER

FOUR

EXAMPLES

The following examples are *complete* Tkinter programs, so may look complicated. However, only one or two lines in each example go to constructing the actual plot. To see the results, just cut-and-paste an example into a python interactive interpreter or make a `.py` file and run it as a python script (e.g. `python xyplot_example.py`).

Note that you will need matplotlib installed.

4.1 X-Y Plot

Here is an example that puts an x-y plot in a toplevel window:

```
import Tkinter as TK
import pygmyplot
tk = TK.Tk()
tk.title("My Plot")
x, y = ([1, 2, 3, 4, 5, 6], [1, 4, 8, 16, 32, 64])
p = pygmyplot.xy_plot(x, y, 'ro-', master=tk)
p.axes.set_ylabel('ordinate')
TK.Button(tk, text="Quit", command=tk.destroy).pack()
tk.mainloop()
```

4.2 Scatter and Histogram Combo

Here, both a scatter plot and a histogram populate the same toplevel window:

```
import random
import Tkinter as TK
import pygmyplot
tk = TK.Tk()
tk.title("Scatter and Histogram")
data = [random.normalvariate(10, 5) for i in range(100)]
pygmyplot.scatter(sorted(data[:50]), sorted(data[50:]), master=tk)
pygmyplot.histogram(data, master=tk, bins=10)
TK.Button(tk, text="Quit", command=tk.destroy).pack()
tk.mainloop()
```

4.3 Heatmap

Here is an example of a heatmap in a toplevel window. Note that the coordinate and value corresponding to central position, `(2, 2)`, is popped, leaving it undefined and thus filled with the implicit background color (white in this example). Also note that the `heat_up` is a fraction (`0.5`), giving the map more dynamic range in color, but reducing the contrast of the positions colored in the background color with the rest of the map. Finally, the value of the `cmap` option is `Greens`, which is a Matplotlib colormap. Other colormaps can be specified to improve contrast or appearance.

```
import random
import Tkinter as TK
import pygmyplot
tk = TK.Tk()
tk.title("A Heatmap")
# make some data
coords = []
vals = []
for i in xrange(5):
  for j in xrange(5):
    rnum = random.normalvariate(10, 5)
    coords.append((i, j))
    vals.append(rnum)
idx_to_pop = coords.index((2, 2))
coords.pop(idx_to_pop)
vals.pop(idx_to_pop)
hm = pygmyplot.xy_heat(coords, vals, heat_up=0.2,
                       figsize=(4, 3.5),
                       xlabs=["A", "B", "C", "D", "E"],
                       ylabs=["i", "j", "k", "l", "m"],
                       cmap="Greens",
                       master=tk)
TK.Button(tk, text="Quit", command=tk.destroy).pack()
tk.mainloop()
```

INDEX