Appendix C

More Matplotlib

Lab Objective: *Display relevant matplotlib information*

The documentation for matplotlib can be frustrating to maneuver and relevant information is sometimes difficult to find. For this reason, we have supplied some of the more applicable and useful information here. For a more general introduction to matplotlib, see lab 3.

Colours

By default, every plot is assigned a different color specified by a "color cycle". It can be overwritten by specifying what color is desired. `matplotlib` recognizes some basic built-in colours.

color	command
black	k
green	g
blue	b
red	r
cyan	c
magenta	m
yellow	y
white	w

The following displays how these colours can be implemented. All the built-in colours are displayed for your convenience in Figure C.1

```
import numpy as np
from matplotlib import pyplot as plt

colours = np.array(["k", "g", "b", "r", "c", "m", "y", "w"])
x = np.linspace(0, 5, 1000)
y = np.ones(1000)

for i in xrange(8):
    plt.plot(x, i*y, colours[i])

plt.ylim([-1, 8])
plt.show()
```

colours.py

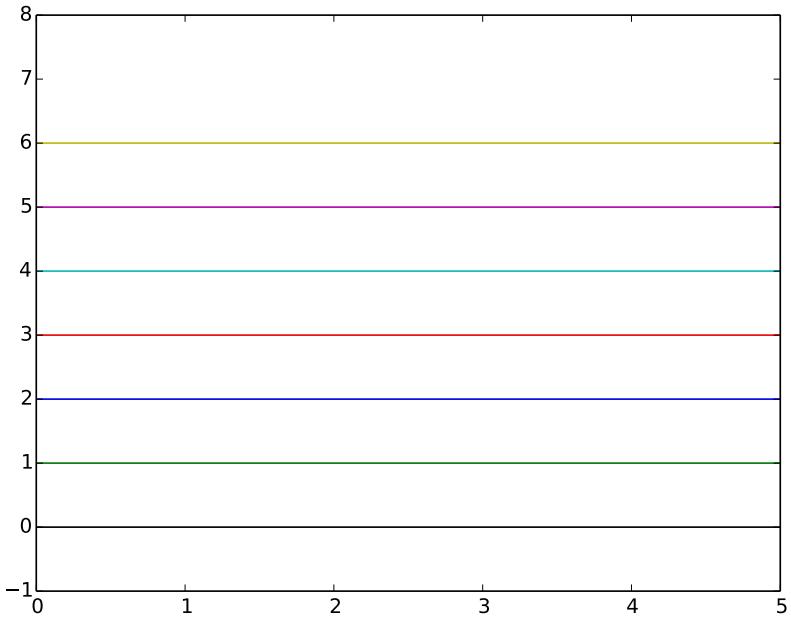

Figure C.1: A display of all the built-in colours.

There are many other ways to specific colours. A popular method to access colours that are not built-in is to use a `RGB` tuple. Another way is to specify the color using an html hex string or its associated html color name like `DarkOliveGreen, FireBrick, LemonChiffon , MidnightBle, PapayaWhip,`, or `SeaGreen`.

Axes

You may have noticed the use of `plt.ylim([ymin, ymax])` in the previous code. This explicitly sets the boundary of the y-axis. Similarily, `plt.xlim([xmin, xmax])` can be used to set the boundary of the x-axis. Doing both commands simultaneously is possible with the `plt.axis([xmin, xmax, ymin, ymax])`. Remember that these commands must be executed after the plot.

Another popular command is `plt.axis("tight")` which adjusts the axes so all data is visible and centered.

lines

width

You may have noticed that the width of the lines above seemed thin considering we wanted to inspect the line colour. `linewidth` is a keyword argument that is defaulted to be `None` but can be given any real number to adjust the line width.

The following displays how `linewidth` is implemented. It is displayed in Figure C.2

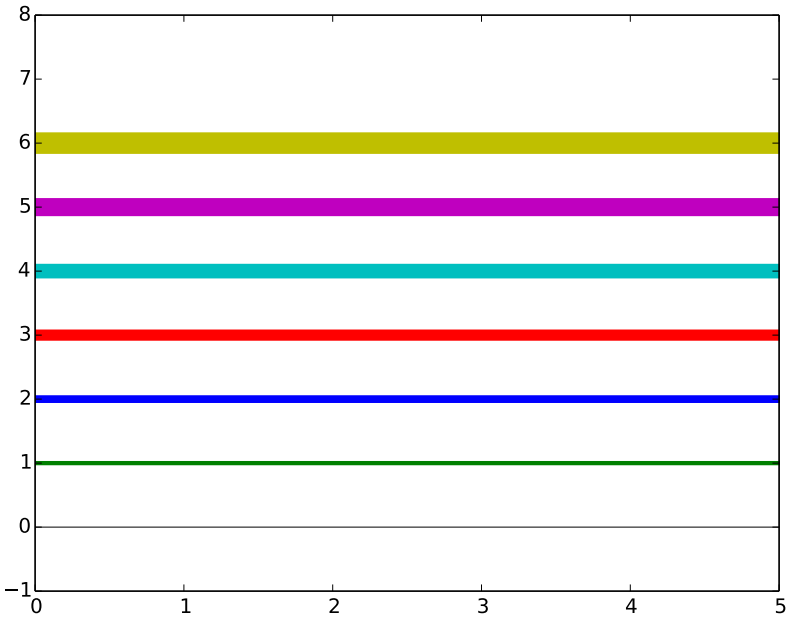

Figure C.2: plot of varying linewidths.

```
lw = np.linspace(.5, 15, 8)

for i in xrange(8):
    plt.plot(x, i*y, colours[i], linewidth=lw[i])

plt.ylim([-1, 8])
plt.show()
```

linewidth.py

style

By default, plots are executed with a solid line style. We can easily change that. The following are accepted format string characters to indicate line style.

character	description
-	solid line style
--	dashed line style
-.	dash-dot line style
:	dotted line style
.	point marker
,	pixel marker
o	circle marker
v	triangle_down marker
^	triangle_up marker
<	triangle_left marker
>	triangle_right marker
1	tri_down marker
2	tri_up marker
3	tri_left marker
4	tri_right marker
s	square marker
p	pentagon marker
*	star marker
h	hexagon1 marker
H	hexagon2 marker
+	plus marker
x	x marker
D	diamond marker
d	thin_diamond marker
|	vline marker
_	hline marker

The following displays how `linestyle` can be implemented.

It is displayed in Figure C.3

```
x = np.linspace(0, 5, 10)
y = np.ones(10)
ls = np.array(['-.', ':', 'o', 's', '*', 'H', 'x', 'D'])

for i in xrange(8):
    plt.plot(x, i*y, colours[i]+ls[i])

plt.axis([-1, 6, -1, 8])
plt.show()
```

linestyle.py

Text

It is also possible to add text to your plots.

To label your axes, the `plt.xlabel()` and the `plt.ylabel()` can both be used.

The function `plt.title()` will add a title to a plot. If you are working with subplots, this command will add a title to the subplot you are currently modifying. To add a title above the entire figure, use `plt.suptitle()`.

All of the `text()` commands can be customized with `fontsize` and `color` keyword arguments.

We can add these elements to our previous example.

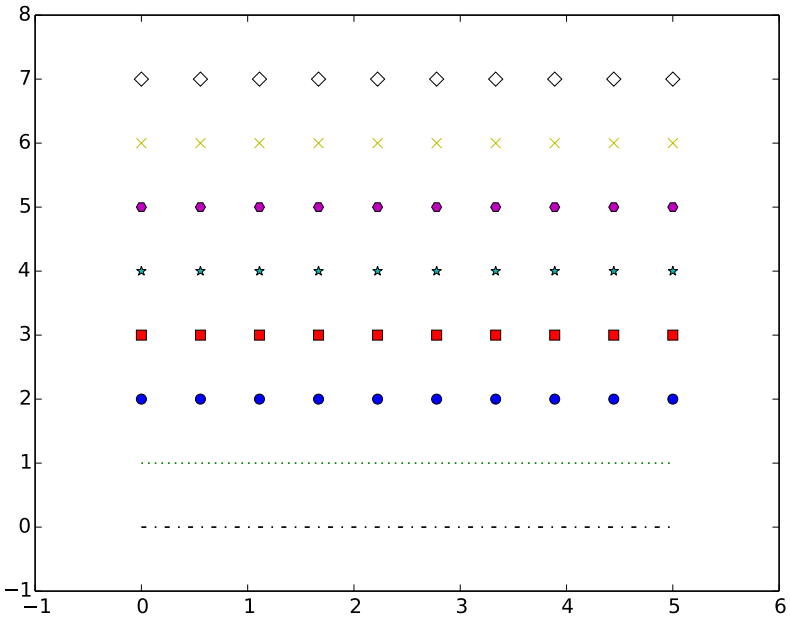

Figure C.3: plot of varying linestyles.

It is displayed in Figure C.4

```
for i in xrange(8):
    plt.plot(x, i*y, colours[i]+ls[i])

plt.title("My Plot of Varying Linestyles", fontsize = 20, color = "gold")
plt.xlabel("x-axis", fontsize = 10, color = "darkcyan")
plt.ylabel("y-axis", fontsize = 10, color = "darkcyan")

plt.axis([-1, 6, -1, 8])
plt.show()
```

text.py

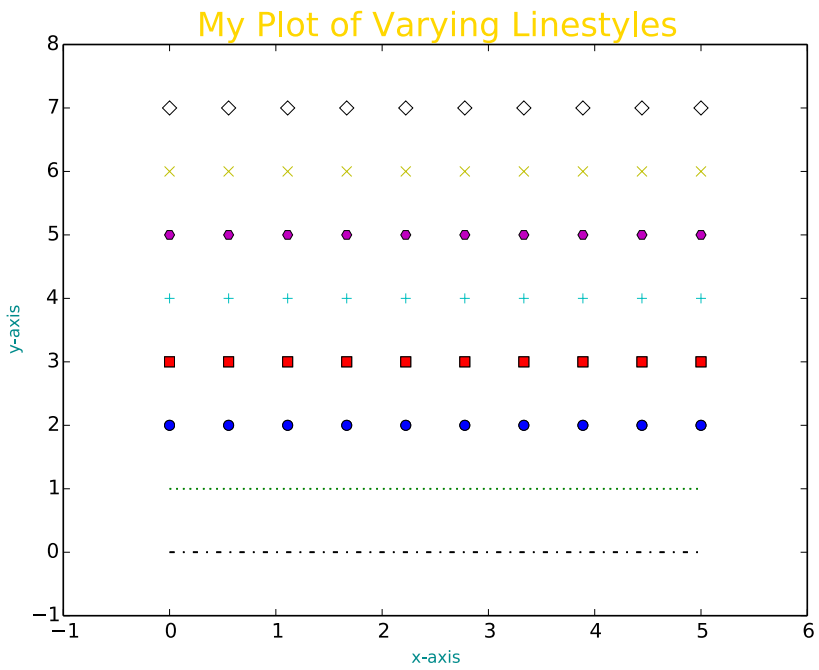

Figure C.4: plot of varying linestyles using text labels.