STOP FIGHTING MATPLOTLIB AND MAKE IT DO WHAT YOU WANT

Levi Naden
Hot Topics for Chodera Lab
June 29, 2017
DA RULES
OR
IF YOU TAKE NOTHING ELSE FROM THIS

1. Operate on the actual objects

2. Separate Calculation and Plotting scripts when possible

3. Always render figures at final-display size
MATPLOTLIB: THE (NOT SO) HUMBLE PLOTTING PROGRAM

- Somewhat easy to get installed and using
- (Nearly) Pure Python
- Does not look the best by default
- Has many “black box”-like functions. All optional
RULE 1: OPERATE ON THE ACTUAL OBJECTS
Let's clear up the namespace first

- So this import statement

- `matplotlib`: The main module, mostly just a house for the sub modules

- `pyplot`: The main workhorse module

- `pylab`: `pyplot + numpy`

  - Don't use this, it's confusing
THE HUMBLE FIGURE...

• It has dreams...

• Aspirations...

• No idea what its objects are...

• Hopes of being on the cover of Scienwaitwhat?
LETS PLAY “COUNT THE OBJECTS”

Hint: There are at least 4
LET'S PLAY “COUNT THE OBJECTS”

Hint: There are at least 4
WHAT ARE THE GOOD (AND BAD) WAYS TO MAKE A FIGURE?

# Method 1
fig = plt.figure()  # Creates the Figure object
ax = plt.subplot(111)  # Creates the Axes object, black box

# Method 2
fig, ax = plt.subplots()  # Creates both

# Method 3
fig = plt.figure()
ax = fig.add_subplot()

plt.anything(...)  

def anything(...):
    if anything in figure_methods:
        plt.gcf().anything(...)
    elif anything in axes_methods:
        plt.gca().anything(...)
    else:
        plt.actual_anything(...)

Hopefully you were done with that last plot!
MULTI-SUBPLOT SHORTCUTS...

```python
fig = plt.figure()
ax = fig.add_subplot(111)
```

- Number of Columns
- Number of Rows
- Axes index in that grid

Equivalent, More general

```
f.add_subplot(2,3,1)
```
MULTI-SUBPLOT SHORTCUTS

```python
f = plt.figure()
a = f.add_subplot(211)
b = f.add_subplot(212)
```

```python
f = plt.figure()
a = f.add_subplot(211)
b = f.add_subplot(212)
c = f.add_subplot(325)
```
MULTI-SUBPLOT SHORTCUTS

```python
f = plt.figure()
a = f.add_subplot(211)
b = f.add_subplot(212)
```

```python
f = plt.figure()
a = f.add_subplot(211)
b = f.add_subplot(212)
c = f.add_subplot(325)
```
MULTI-SUBPLOT ARRAYS:
CREATE ALL SUBPLOTS AT ONCE

```python
f, a = plt.subplots(1, 1)
print(a)
```
MULTI-SUBPLOT ARRAYS:
CREATE ALL SUBPLOTTS AT ONCE

```python
f, a = plt.subplots(1, 1)
print(a)

f, (a, b) = plt.subplots(2, 1)
print(a)
print(b)
```
MULTI-SUBPLOT ARRAYS:
CREATE ALL SUBPLOTS AT ONCE

```python
f, a = plt.subplots(1, 1)
print(a)

f, (a, b) = plt.subplots(2, 1)
print(a)
print(b)

f, a = plt.subplots(2, 2)
print(a)
```
MULTI-SUBPLOT ARRAYS:
CREATE ALL SUBPLOTS AT ONCE

- Index like a NumPy Array (Because it IS!)
- Dimensions are shape of grid
MULTI-SUBPLOT ARRAYS: CREATE ALL SUBPLOTS AT ONCE

```python
f, a = plt.subplots(1, 1)
print(a)

f, (a, b) = plt.subplots(2, 1)
print(a)
print(b)

f, a = plt.subplots(2, 2)
print(a)
print(type(a))
print(a[0, 1])

f, (a, b) = plt.subplots(2, 2)
print(a)
print(b)
```

`Axes((0.125, 0.11; 0.775x0.77))`

`Axes((0.125, 0.53; 0.775x0.35))`

`Axes((0.125, 0.11; 0.775x0.35))`

Not recommended (ambiguous)
Rule 2:
Run computations separately from figure generation*

*When you can
RULE 3A: ALWAYS RENDER PLOTS AT FINAL DISPLAY SIZE

RULE 3B: MAKE YOUR FONTS AND COLORS EASY TO MANAGE
SMALL TEXT = BAD FIGURE!

Original

figsize x2

Thanks to Josh for data and initial figure code
THERE IS NO UNIVERSAL FIX

This is much less of a pain if you are operating on the objects
OKAY, NO UNIVERSAL FIX. CAN I MAKE IT EASIER AT LEAST? YES!

Set a font scalar
Feed into all fontsize=fsc

```
scale = 2
fig = plt.figure(figsize=(10*scale, 2*scale))
default_font = mpl.rcParams['font.size']
fsc = default_font * scale
```

Programmatically loop over objects
Rule 1!

```
for i, datum in enumerate(data):
    ax = fig.add_subplot(1, 5, i + 1)
    ax.set_ylabel("Text!", fontsize=fsc)
    for axis in [ax.xaxis, ax.yaxis]:
        axis.set_tick_params(labelsize=fsc)
```

Do not adjust global font setting, each figure is different (usually)
THAT DIDN’T FIX IT...

- Keep adjusting the font sizes until it looks right!
- This is why Rule 2 is helpful. Fast figure generation!
- Remember that on paper, and on screen look different
- May need to recreate figure (Rule 2)
- Starting with your figure at final display size helps reduce this process
MATPLOTLIB COLOR BASICS

- Rule 3B: Define your “blue”, “red”, “green”, etc. at the top of your script
- Accepts Scalar, Hex, RGB tuples, and RGBA tuples
- Remember: Your image will look different on different screens, mediums, may need to adjust to get it right
  - Speed saved from Rule 2, running image creation separately.
- Handy sites:
  - http://colorbrewer2.org/
  - http://paletton.com/
  - Google: “Shades of X hex”
MATPLOTLIB

COLORMAPS

• https://matplotlib.org/examples/color/colormaps_reference.html

• Matplotlib tries to guess what your scale is based on data.
  • Normalizes everything into a [0,1] range by default

• Every built in colormap has a REVERSE, “cmap_r” e.g. “Blues” and “Blues_r”

• You can make your own CMAPs, that is really its own full talk, but I can show later
ADVANCED TOPICS
HEAT MAPS

- Want to recreate this in Matplotlib? Not hard!
- Image provided by John
HEAT MAPS: DATA

- Constant scaled by Normal distribution and noise

- What you need:
  - X Dimension (1D)
  - Y Dimension (1D)
  - Z (data) 2D of shape (Nx Ny), entries are observed datum
HEAT MAPS: AS AN IMAGE

```python
f, a = plt.subplots(1, 1)
a.set_xlim(Tx.min(), Tx.max())
a.set_ylim(Ty.max(), Ty.min())
imgdata = a.imshow(Z[:, :, ::-1],
                   cmap='hot_r',
                   extent=[Tx.min(), Tx.max(), Ty.min(), Ty.max()],
                   vmin=0,
                   vmax=-10)
colorbar = f.colorbar(imgdata, ticks=np.linspace(0, -10, 11))
```
HEAT MAPS: AS AN IMAGE

```python
f, a = plt.subplots(1, 1)
a.set_xlim(Tx.min(), Tx.max())
a.set_ylim(Ty.max(), Ty.min())
imgdata = a.imshow(Z[:, :, ::-1],
    cmap='hot_r',
    extent=[Tx.min(), Tx.max(), Ty.min(), Ty.max()],
    vmin=0,
    vmax=-10)

colorbar = f.colorbar(imgdata, ticks=np.linspace(0, -10, 11))
```
HEAT MAPS: AS A COLOREMPHESH

```
g, b = plt.subplots(1, 1)
b.set_xlim(Tx.min(), Tx.max())
b.set_ylim(Ty.max(), Ty.min())
TX, TY = np.meshgrid(Tx, Ty)
pcm = b.pcolormesh(TX, TY, Z,
                   cmap='hot_r',
                   vmin=-10,
                   vmax=0)
boundaries = np.linspace(-10, 0, 256)
cbar = plt.colorbar(pcm, ax=b, orientation='vertical',
                    boundaries=boundaries,
                    values=boundaries[1:])
cbar.set_ticks(np.linspace(0, -10, 11))
```
HEAT MAPS: AS A COLOREMESH

- CAST 2, 1-D -> 2 2D
- Matplotlib oddities....

Make colorbar from scratch

Original

From code
Overlapping 2 Axes to share an X-axis

```python
x = np.linspace(0, 1, 100)
x2 = np.sin(x*np.pi*2)
y = x
y2 = x**2

f = plt.figure()
a = f.add_subplot(1, 1, 1)
# Create another Axes which shares an X
b = a.twinx()
a.plot(x, y, '-r')
a.yaxis.set_tick_params(colors='r')
b.plot(x, y2, '.g')
b.yaxis.set_tick_params(colors='g')
```
Multi-Plot-Fu

Overlapping 2 Axes to share an X-axis

- Create a new empty Axes, \( b \), in the same Figure and location as \( a \)
- Set \( b \)'s \( xaxis \) to the same object as in \( a \)
- Create a new \( yaxis \) for \( b \) and move it to the other side from where \( a \)'s \( yaxis \) is
- Define all other bounding boxes the same as \( a \)
MULTI- PLOT- FU

You can do it for Shared Y as well

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 1, 100)
x2 = np.sin(x*np.pi*2)
y = x

g = plt.figure()
c = g.add_subplot(1, 1, 1)
d = c.twiny()
c.plot(x, y, '-r')
c.xaxis.set_tick_params(colors='r')
d.plot(x2, y, '.g')
d.xaxis.set_tick_params(colors='g')
```
**More Multi-Plot-Fu**

```
import matplotlib.pyplot as plt
from matplotlib import gridspec

Nr = 5
Nc = 3
grid = gridspec.GridSpec(Nr, Nc,
                        wspace=0.5,
                        hspace=0.7)

f = plt.figure()
# Full dim plot
f.add_subplot(grid[::, 0])
# Across 1 dim
f.add_subplot(grid[0, 1:])
# Across both row and col
f.add_subplot(grid[1:3, 1:])
# Individual plots
f.add_subplot(grid[3, 1])
f.add_subplot(grid[3, 2])
# Skip a cell
f.add_subplot(grid[4, 2])
```

GridSpec: Define a grid of your figure
MORE MULTI-PLOT-FU

Check the Matplotlib site for how to make this
PRACTICAL MULTI-PLOT-FU

• Some porous membrane.

• Inject some tracer in regular pulses, flows and decays with time

• Known probability of tracer residency as function of thickness

• Want: Membrane conc. as function of time and thickness

• Also show bulk conc. and probability
```python
def scalefont(ax, fs):
    for axis in [ax.xaxis, ax.yaxis]:
        axis.set_tick_params(labelsize=fs)

timedim = 40
pdim = 10
subdim = 2
cbdimm = 1

fs = 20
grid = gridspec.GridSpec(pdim+subdim, timedim+subdim+cbdimm, hspace=0, wspace=0)
f = plt.figure(figsize=(timedim+subdim, pdim+subdim))

prob = f.add_subplot(grid[subdim:, :subdim])
prob.xaxis.set_ticklabels([])
prob.set_xlabel('p(x)', fontsize=fs)
prob.set_xlim(1, 0)
prob.set_ylabel('Membrane Distance (A)', fontsize=fs)
prob.set_ylim(-memsize, 0)
prob.plot(probability, depth)

mix = f.add_subplot(grid[subdim:, subdim:subdim+timedim])
mix.set_xlim(0, ntime)
mix.yaxis.tick_right()
mix.set_ylim(-memsize, 0)
mix.set_xlabel('time (s)', fontsize=fs)
mix.yaxis.set_visible(False)
X, Y = np.meshgrid(t[lead:], depth)
pcolor = mix.pcolormesh(X, Y, Z.T, cmap='Blues')
cbarax = f.add_subplot(grid[subdim:, -cbdimm:])
colorbar = f.colorbar(pcolor, cax=cbarax)
cbarax.set_ylabel('Membrane Conc.' , fontsize=fs)

time = f.add_subplot(grid[:subdim, subdim:-cbdimm])
time.plot(t, conc)
time.set_xlim(0, ntime)
time.xaxis.set_visible(False)
time.set_ylim(0, conc.max())
time.yaxis.set_ticklabels([])
time.set_ylabel('Bulk
Conc.', fontsize=fs)

for ax in [prob, mix, time, cbarax]:
    scalefont(ax, fs)
```

"Kiai!"
PRACTICAL MULTI- PLOT-FU BREAKDOWN

```python
def scalefont(ax, fs):
    for axis in [ax.xaxis, ax.yaxis]:
        axis.set_tick_params(labelsize=fs)

timedi = 40
pdim = 10
subdim = 2
cbdim = 1

fs = 20
g = gridspec.GridSpec(pdim+subdim, timedi+subdim+cbdrid, hspace=0, wspace=0)
f = plt.figure(figsize=(timedi+subdim, pdim+subdim))
```

Font Scaling

GridSpec layout

“Kiai!”
PRACTICAL
MULTI- PLOT-FU BREAKDOWN

Programmatic area selection from GridSpec

Simple 2D plot

“Kiai!”

Hide X axis ticks to reduce clutter
PRACTICAL MULTI- PLOT- FU BREAKDOWN

2D GridSpec Selection

Meshgrid and pcolormesh heat maps

Color bar created and assigned to empty GridSpec plot

“Kiai!”
PRACTICAL MULTI-PLOT-FU

wh-CREEPER!
YOUR REQUESTS, THAT DID NOT FIT IN EARLIER
Fitting the image in the box

Pretty much always set bbox_inches

```python
f.savefig('no_box.png', bbox_inches=False)
f.savefig('yes_box.png', bbox_inches='tight')
```
SPEAKING OF WAFFLES—
I MEAN IMAGES...

```
img = plt.imread('waffles.png')
```

This makes an RGB array
Can then use imshow
DON’T BE AFRAID TO USE OTHER PROGRAMS TO MAKE FINAL IMAGE!
LEGEND PLACEMENT

• Remember to keep acting on the objects themselves

• I cannot do better than the online guide
  
  • https://matplotlib.org/users/legend_guide.html

• For better control, manually call the legend object, place it with absolute coordinates in the figure, adjust as needed