


Branch: master ▾

Find file


Copy path

ma591spring2020 / material / matplotlib.ipynb




 **asaibab** Add files via upload  
098f5c4 12 days ago

1 contributor

<>



RawBlameHistory

671 lines (670 sloc) 633 KB

# Plotting with matplotlib ¶

Housekeeping:

- Will regularly post lecture videos online. Watch at your own pace.
- Office hours via zoom Fridays 1:55 PM - 3:30 PM.
- Homework 8, due Friday 4/3.
- Contact me regarding any issues (email/piazza/zoom)
- Thank you!

## Installing matplotlib

Jupyter notebooks should already come installed with matplotlib. Otherwise, one can execute the commands

```
pip install matplotlib
```

to install a working copy of matplotlib.

Some other plotting tools to use

- `mplot3d` for 3D plotting requirements
- `seaborn` for statistical data visualization.

Both these tools are based on `matplotlib` but `seaborn` requires additional installation.

Some other packages for visualization

- `vtk`
- `mayavi2`
- `paraview`

Reading:

- Chapter 5, Langtangen's "A primer on scientific programming with Python" .
- Chapter 4, [Python Data Science Handbook](https://jakevdp.github.io/PythonDataScienceHandbook/)  
(<https://jakevdp.github.io/PythonDataScienceHandbook/>).
- [matplotlib website](https://matplotlib.org/) (<https://matplotlib.org/>).

## Preliminaries

At the start of every jupyter notebook I typically add the following plotting commands.

```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline

        #Some code to beautify the plots
        plt.rcParams['figure.figsize'] = [10, 5]
        plt.rcParams['xtick.labelsize'] = 14
```

```
plt.rcParams['ytick.labelsize'] = 14
```

These two commands are specific to jupyter notebooks

- `%matplotlib inline` which plots static images
- `%matplotlib notebook` which plots interactive images

Without these commands, one has to type `plt.show()` each time to display an image.

The next three commands adjust the figure size, `xtick labelsize`, and `ytick labelsize`, which make the ticks more legible. These changes are applied to every plot but can be changed at a later time.

Type `print(plt.rcParams.keys())` to see a list of parameters that we can edit.

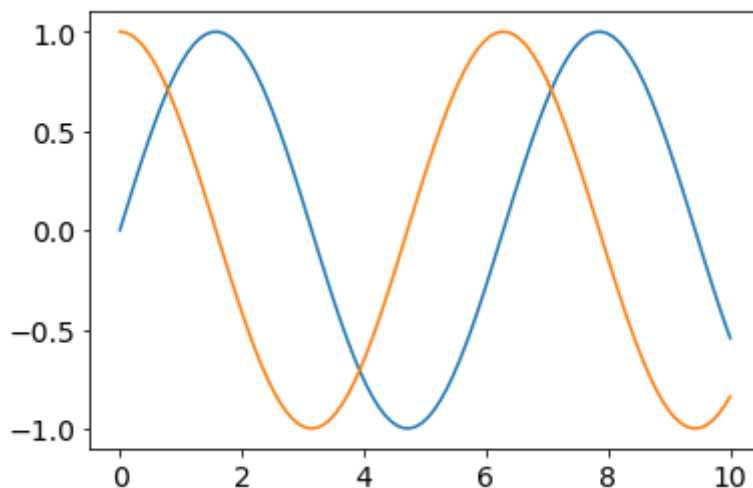
## Example 1: Plotting simple functions

```
In [2]: import numpy as np

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

Out[2]: [`<matplotlib.lines.Line2D at 0x7fb431ad8910>`]



To save this image, one can use the `savefig` command.

```
In [3]: !mkdir figs
plt.savefig('figs/mysincos.eps')      # May have to first create the directory figs
!ls -lh figs/mysincos.eps

mkdir: figs: File exists
-rw-r--r--@ 1 asaibab  108   638B Mar 21 15:47 figs/mysincos.eps
<Figure size 432x288 with 0 Axes>
```

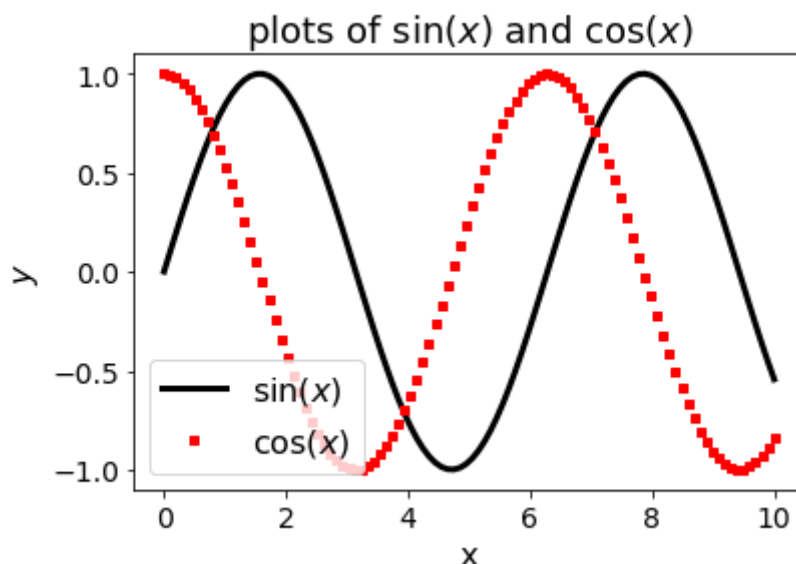
## Example 2: Editing the plot features

```
In [4]: x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x), 'k-', linewidth = 3.0)
plt.plot(x, np.cos(x), 'rs', lw = 3.0, markersize = 4)

plt.xlabel('x', fontsize = 16)
plt.ylabel('y', fontsize = 16)
plt.title('plots of $\sin(x)$ and $\cos(x)$', fontsize = 18)
plt.legend(( '$\sin(x)$', '$\cos(x)$' ), fontsize = 16, loc = 'best')
```

Out[4]: <matplotlib.legend.Legend at 0x7fb4207f02d0>



## Exercise

After studying a certain type of cancer, a researcher hypothesizes that in the short run the number ( $y$ ) of malignant cells in a particular tissue grows exponentially with time ( $t$ ). That is,  $y = \alpha_0 e^{\alpha_1 t}$ . The researcher records the data given below

.	.	.	.	.	.
t (days)	1	2	3	4	5
y (cells)	16	27	45	74	122

With some analysis, we find the coefficients of "best" fit

$$\alpha_0 = 9.7309 \quad \alpha_1 = 0.5071.$$

- Plot the data points (with black crosses) along with the "best" fit curve (as a solid red line).
- Predict at what time the number of cells reaches 100. Show this visually using a dashed green line.

The plots should have the following features

- Clearly label the x- and y-axes. All the labels should have fontsize 18.
- Title the figure 'Exponential line fit.' This should have fontsize 20
- Add a legend with three labels 'Data', 'Fit', and 'Prediction'. The fontsize for the labels should be 18.
- The markersize should be 10 pts, and the line-widths should be 4 pts.

```
In [5]: t = np.array([1,2,3,4,5])
        y = np.array([16,27,45,74,122])

        # Part (a)
        plt.figure()
        plt.plot(t,y, 'kx', markersize = 10)

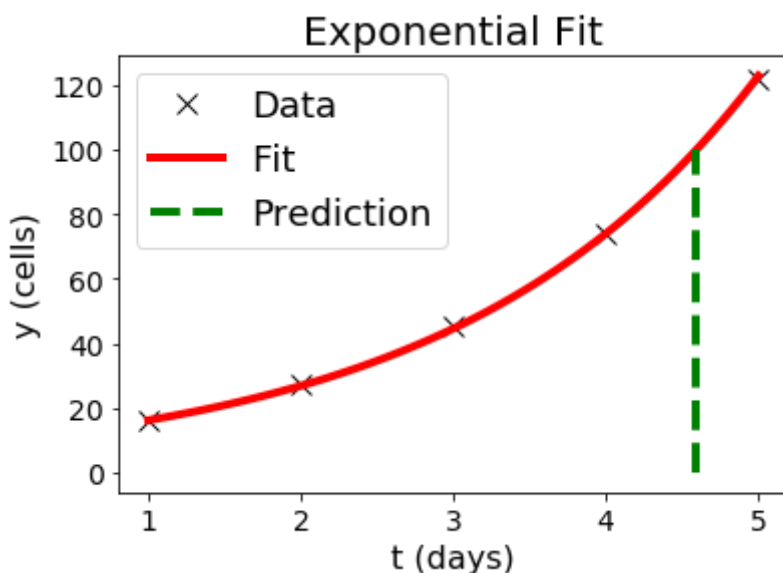
        tfit = np.linspace(1,5,100)
        yfit = 9.7303*np.exp(0.5071*tfit)

        plt.plot(tfit, yfit, 'r-', linewidth = 4.0)
        plt.xlabel('t (days)', fontsize = 16)
        plt.ylabel('y (cells)', fontsize = 16)
        plt.title('Exponential Fit', fontsize = 20)

        # Part (b)
        yfinal = 100
        pred = np.log(yfinal/9.7303)/0.5071
        plt.plot(pred*np.ones((100,)), np.linspace(0,yf
            inal,100), 'g--', lw = 4.0)

        plt.legend(('Data', 'Fit', 'Prediction'), loc = 'upper left'
            , fontsize = 18)
```

Out[5]: <matplotlib.legend.Legend at 0x7fb420944650>



## Subplots

Consider the earlier example, now in the form of subplots.

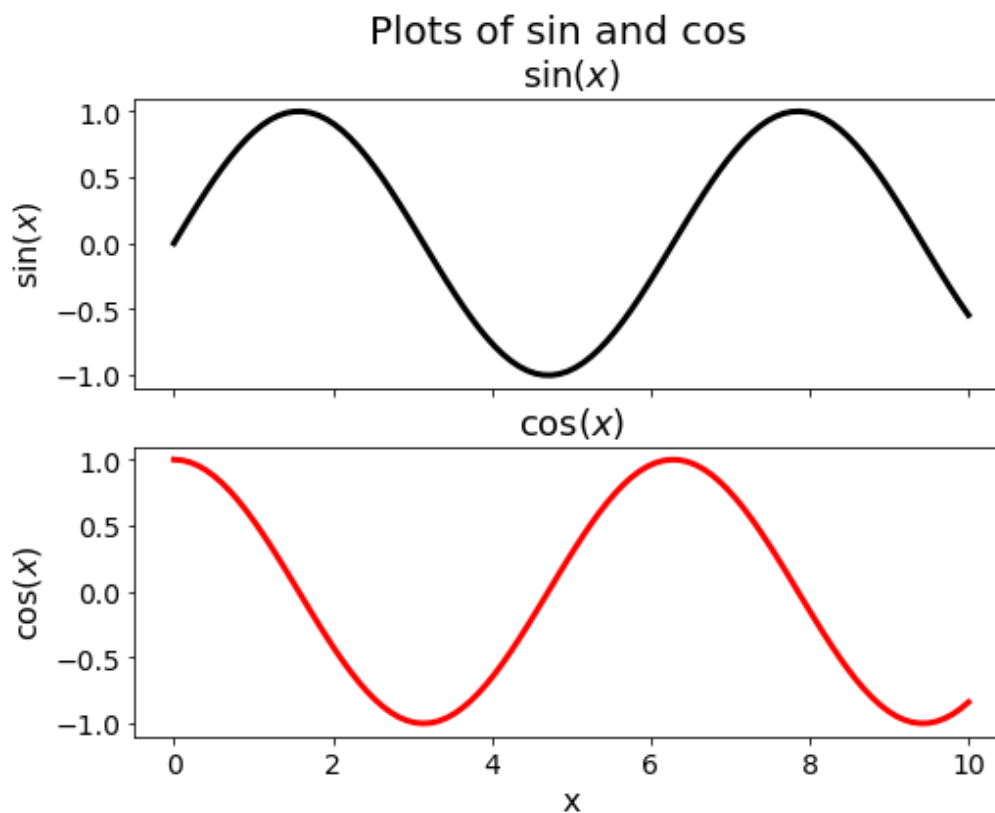
```
In [6]: f, (ax1, ax2) = plt.subplots(2,1, sharex = True, figsize = (
      8,6))

#Axes 1
ax1.plot(x, np.sin(x), 'k-', lw = 3.0)
ax1.set_title('$\sin(x)$', fontsize = 18)
ax1.set_ylabel('$\sin(x)$', fontsize = 16)

#Axes 2
ax2.plot(x, np.cos(x), 'r-', lw = 3.0)
ax2.set_title('$\cos(x)$', fontsize = 18)
ax2.set_ylabel('$\cos(x)$', fontsize = 16)
ax2.set_xlabel('x', fontsize = 16)

# Title for an overall plot
f.suptitle('Plots of sin and cos', fontsize = 20)
```

```
Out[6]: Text(0.5, 0.98, 'Plots of sin and cos')
```



Some features to keep in mind

- `sharex` shares the same x-axis between the two plots.
- `ax1`, `ax2` are two objects that give very fine scale control over each subplot.
- When you use axes rather than the figure you need slightly different commands. For example, note that 'xlabel' has changes to 'set\_xlabel'.

We can add a grid of subplots as follows.

```
In [7]: f, axarray = plt.subplots(2,3, sharex = True, sharey = True,
```

```

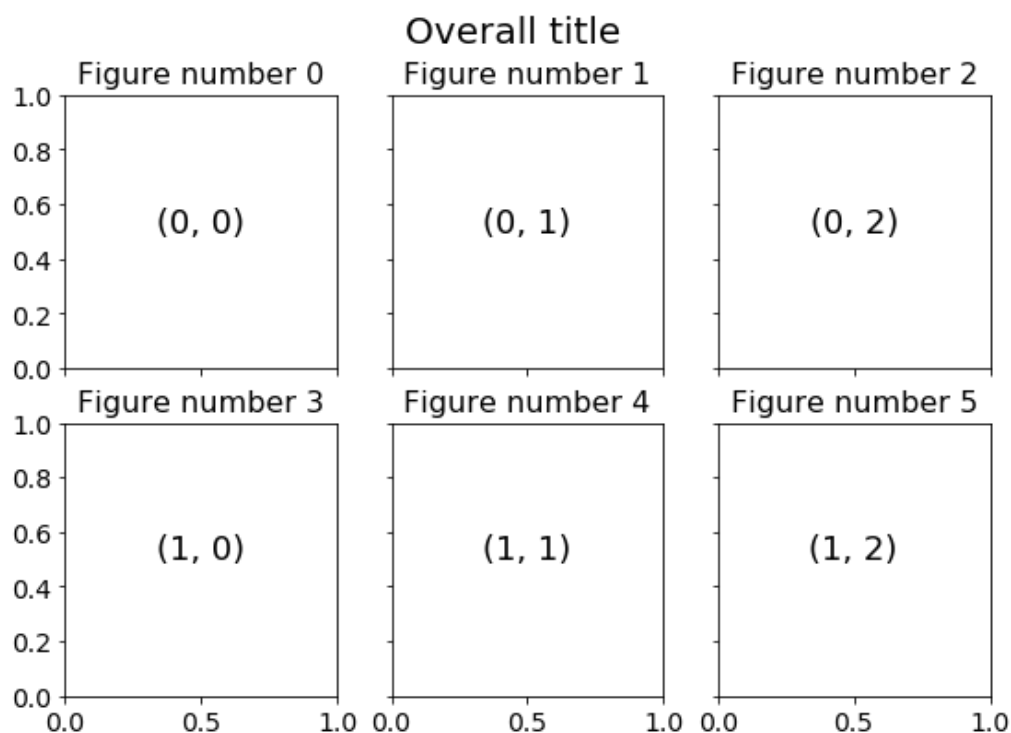
figsize = (9,6))

# We can iterate through the images
for i in range(2):
    for j in range(3):
        axarray[i,j].text(0.5, 0.5, str((i, j)), fontsize=18
, ha='center')
        axarray[i,j].set_title('Figure number %d' %(3*i+j),
fontsize = 16)

f.suptitle('Overall title', fontsize = 20)

```

Out[7]: Text(0.5, 0.98, 'Overall title')



## Plotting images

There are several functions for plotting images. `imshow` is good for grayscale and color images. We can use this function with many different colormaps. Here are three examples but a longer list is available [here \(https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html\)](https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html).

```

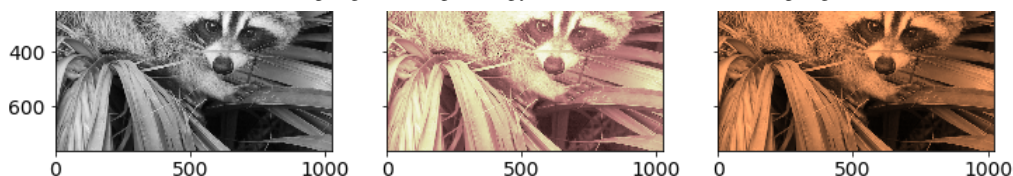
In [8]: from scipy import misc
face = misc.face(gray=True)

f, (ax1,ax2,ax3) = plt.subplots(1,3, sharey = True, figsize
= (12,4))
ax1.imshow(face, cmap=plt.cm.gray)
ax2.imshow(face, cmap=plt.cm.pink)
ax3.imshow(face, cmap=plt.cm.copper)

```

Out[8]: <matplotlib.image.AxesImage at 0x7fb431eacc50>





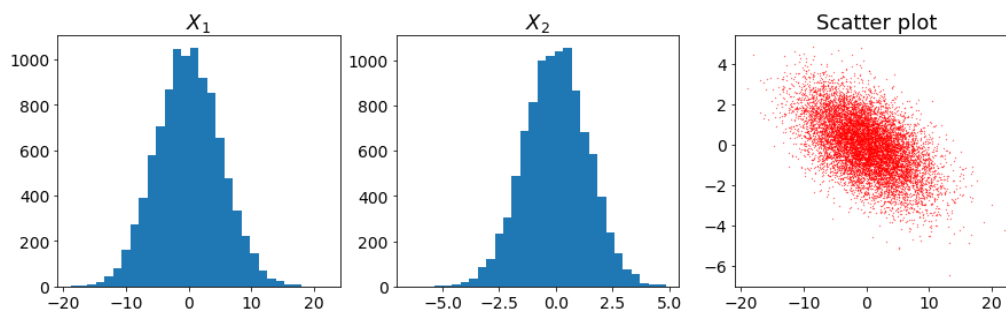
## Histograms

```
In [9]: Y = np.random.randn(10000,2)

x = Y[:,0] + 5*Y[:,1]
y = Y[:,0] - Y[:,1]

f, (ax1,ax2, ax3) = plt.subplots(1,3, figsize = (15,4))
ax1.hist(x, bins = 30)
ax1.set_title('$X_1$', fontsize = 18)
ax2.hist(y, bins = 30)
ax2.set_title('$X_2$', fontsize = 18)
ax3.plot(x,y, '.r', markersize = 0.5)
ax3.set_title('Scatter plot', fontsize = 18)
```

Out[9]: Text(0.5, 1.0, 'Scatter plot')



The package seaborn also has excellent tools for statistical visualization.

## Visualizing two dimensional functions

Consider the following function

$$f(x, y) = \sin^{10}(x) + \cos(10 + xy)\cos(x) \quad x, y = [0, 5]^2.$$

We consider three different ways of plotting the function.

```
In [10]: def f(x, y):
          return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)

x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

```
In [11]: f, (ax1,ax2,ax3) = plt.subplots(1,3, sharey = True, figsize
```



```

= (12,4))

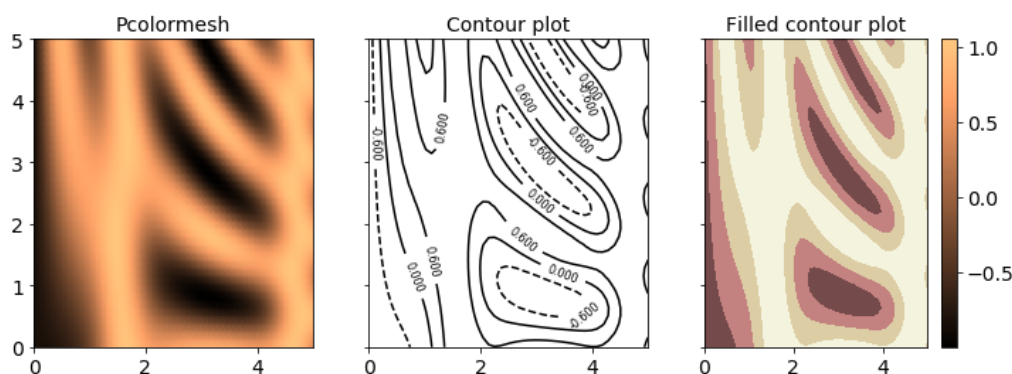
#Pcolormesh - similar to imshow with different orientation
c1 = ax1.pcolormesh(X,Y,Z, shading = 'gouraud', cmap = plt.cm.copper)
plt.colorbar(c1)
ax1.set_title('Pcolormesh', fontsize = 14)

#Contourplot
contours = ax2.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)
ax2.set_title('Contour plot', fontsize = 14)

#Filled countour plot
ax3.contourf(X, Y, Z, 3, cmap = plt.cm.pink)
ax3.set_title('Filled contour plot', fontsize = 14)

```

Out[11]: Text(0.5, 1.0, 'Filled contour plot')



## Animations

There are several options for animations

1. Save several files for each frame of the video fig0001.png to fig0060.png. Then run an animator (e.g. ImageMagick's convert).
2. Use matplotlib.animation tool to create a video.

```

In [ ]: """
        Matplotlib Animation Example

        author: Jake Vanderplas
        email: vanderplas@astro.washington.edu
        website: http://jakevdp.github.com
        license: BSD
        Please feel free to use and modify this, but keep the above
        information. Thanks!
        """

        from matplotlib import animation

        # First set up the figure, the axis, and the plot element we
        # want to animate
        fig = plt.figure()
        ax = plt.axes(xlim=(0, 2), ylim=(-2, 2))

```

```
line, = ax.plot([], [], lw=2)
```

```
# initialization function: plot the background of each frame
```