Astronomical data analysis using Python

Lecture 7

Acknowledgements

Some slides used in this talk were prepared by Varun Bhalerao for an earlier version of this course.

Assignment 1

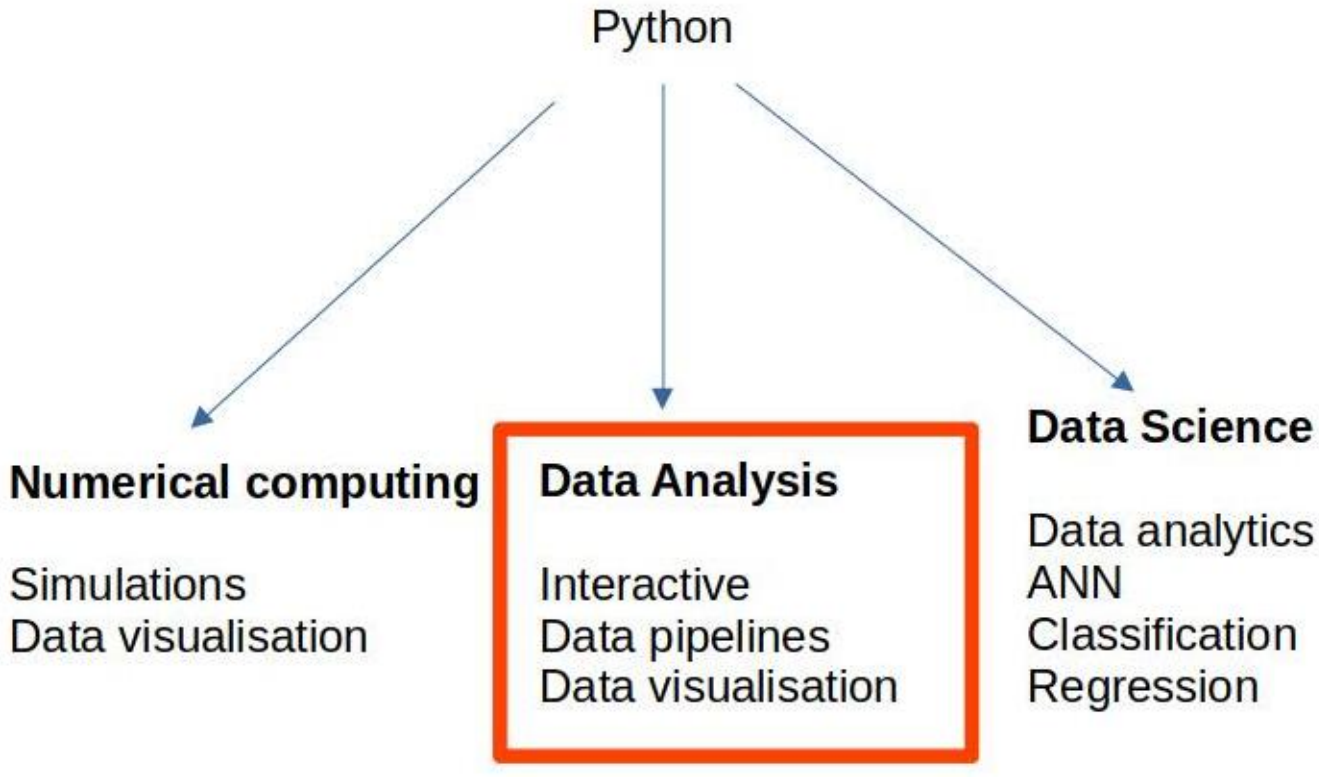
The next session to be conducted by Dr. Preetish Mishra on Dec 2, 2021 will be devoted to a discussion of the problems in Assignment 1. Please try to solve the problems by then.

Plotting with Python

- ppgplot/pygplot
- sm
- gnuplot
- R
- Matlab
- Mathematica
- IDL
- Octave

Interfaces available to all of these. Useful if you already know the syntax. Otherwise, just use `matplotlib` which we will learn about today.

Python usage in big data applications



Matplotlib - the standard plotting library for Python

Matplotlib has very powerful plotting facilities. So before we start with the details, let us take a look at https://matplotlib.org/stable/gallery/index.html (https://matplotlib.org/stable/gallery/index.html). It is a showcase of the capabilities of Matplotlib: starting from very simple graphs, to far more complicated ones.

When you get started with plotting, choose a plot that does most of what you want, and then modify it as per your needs.

Let’s get started!

Matplotlib commands are nearly identical to Matlab commands. So, in case you know Matlab, you won't have to learn much.

In [106]:

```
# %pylab inline

import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (5,3.5)
```

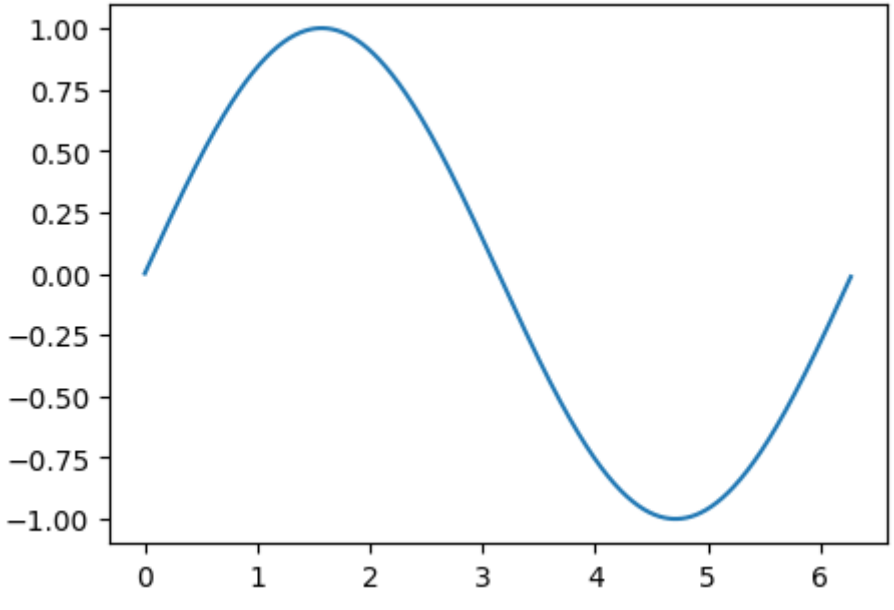
In [50]:

```
# Generate some test data
x = np.arange(0, 6.28, 0.01)
y = np.sin(x)
```

Now we plot y versus x

In [108]:
```
plt.plot(x, y)
```

Out[108]:
```
[<matplotlib.lines.Line2D at 0x7f6cfd5e7630>]
```

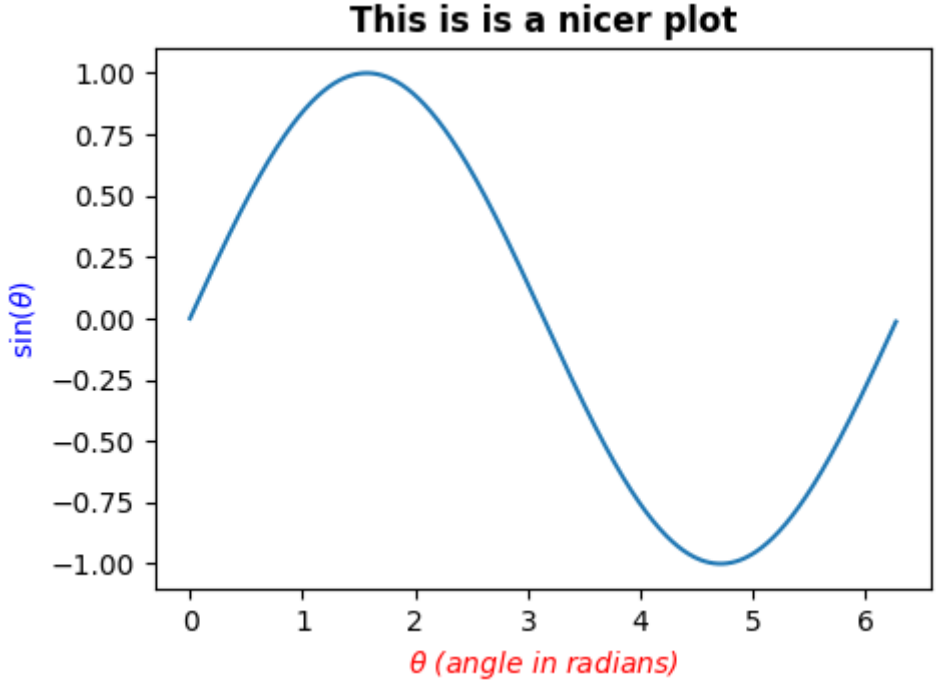


Adding Labels

That was simple: we have plotted y as a function of x. Now let us add axis labels and a title.

Fancier labels:

In [110]:
```
plt.plot(x, y)
xlab = plt.xlabel(r"$\theta$ (angle in radians)") # note the r before quotes
ylab = plt.ylabel(r'sin($\theta$)')
thetitle= plt.title("This is is a nicer plot");thetitle.set_fontweight('bold')
ylab.set_color('blue');xlab.set_color('red'); xlab.set_style('italic')
```

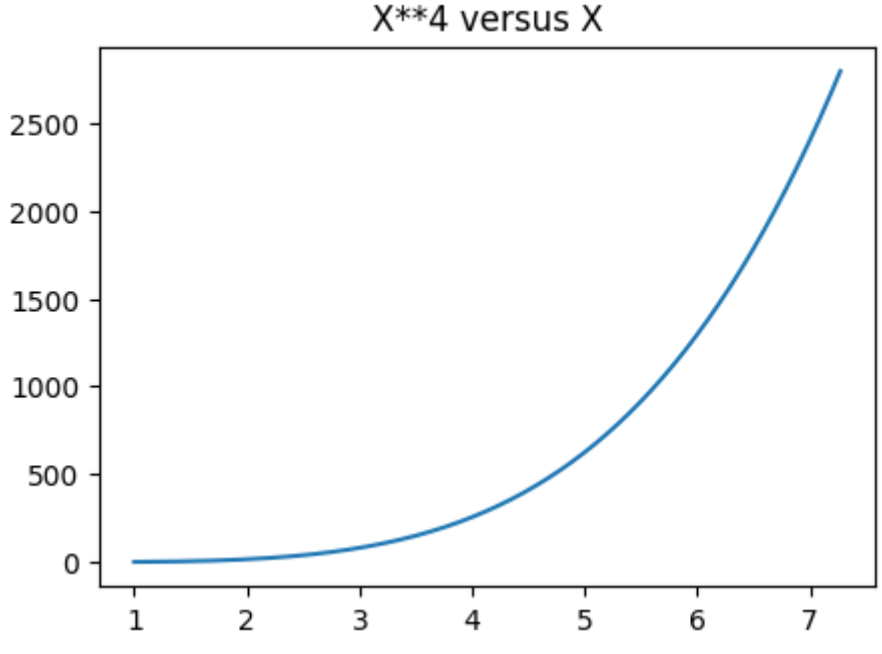


More examples at
https://matplotlib.org/stable/gallery/text_labels_and_annotations/fonts_demo.html
(https://matplotlib.org/stable/gallery/text_labels_and_annotations/fonts_demo.html)

Axis Range and scale

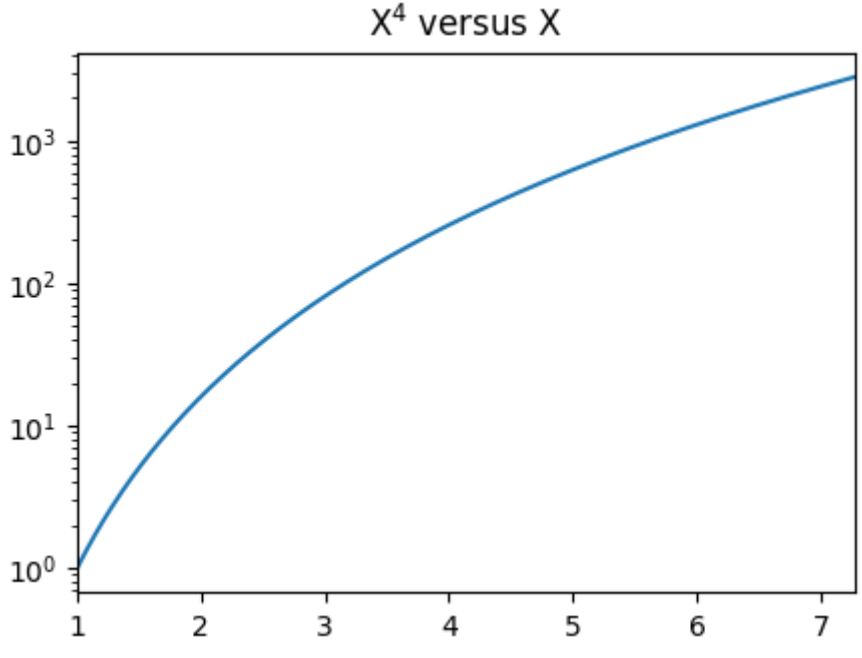
In [111]:
```
another_x = x+1
plt.plot(another_x,another_x**4)
plt.title('X**4 versus X')
```

Out[111]:
```
Text(0.5, 1.0, 'X**4 versus X')
```



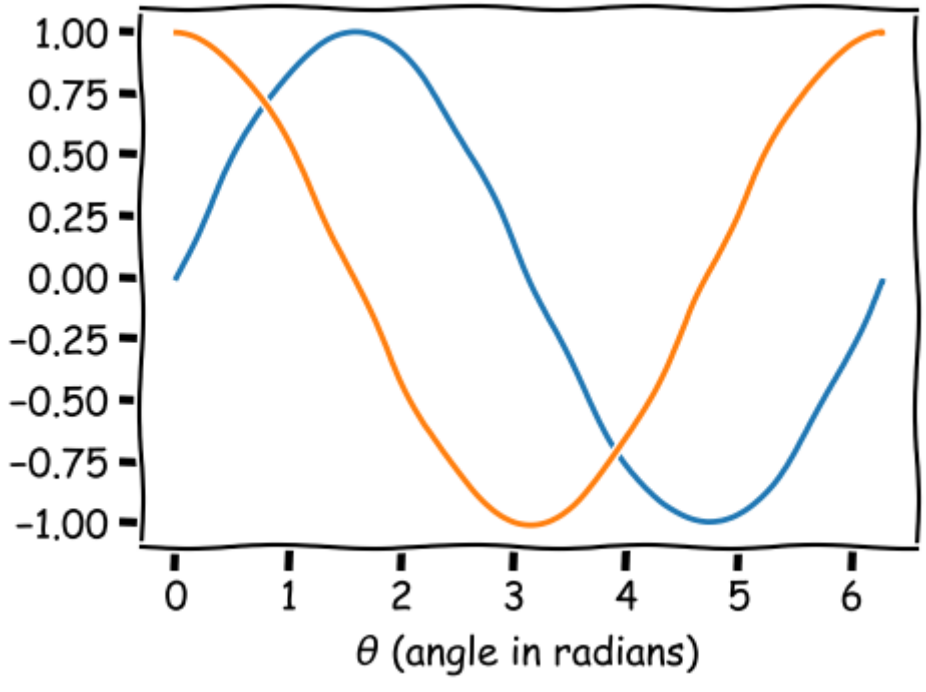
In [112]:

```
# Let's make it look a bit better
plt.plot(another_x,another_x**4)
plt.title(r'X$^4$ versus X') # The $ signs are for latex inputs
plt.xlim(min(another_x), max(another_x))
plt.yscale('log')
plt.show()
```



Overplot

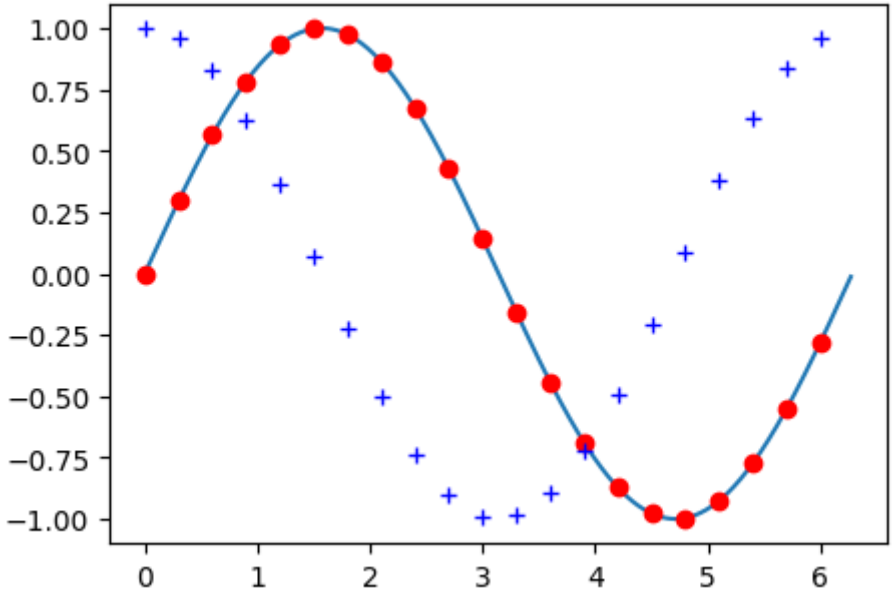
In [123]:
```
plt.rcParams['figure.figsize'] = (5,3.5)
plt.xkcd()
plt.plot(x, y)
plt.plot(x, np.cos(x)) # Second curve in the same figure
xlab = plt.xlabel(r"$\theta$ (angle in radians)")
plt.rcdefaults()
plt.rcParams['figure.figsize'] = (5,3.5)
```



Styles, symbols and colours

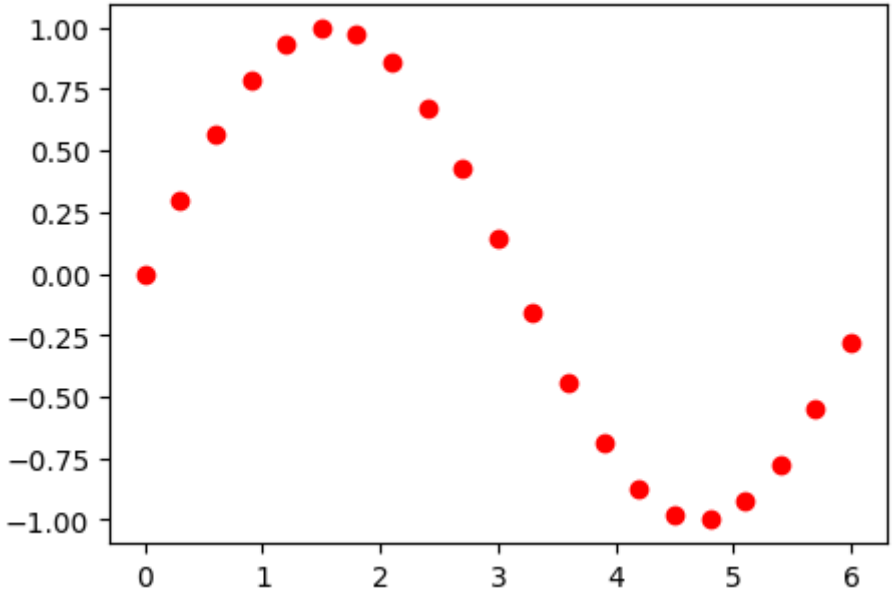
In [124]:
```
plt.plot(x, y)
x_spaced = np.arange(0, 2.0*np.pi, 0.3)
plt.plot(x_spaced, np.sin(x_spaced), marker='o', color='red', linestyle='None')
plt.plot(x_spaced, np.cos(x_spaced), 'b+')
```

Out[124]:
```
[<matplotlib.lines.Line2D at 0x7f6cfdc05c88>]
```



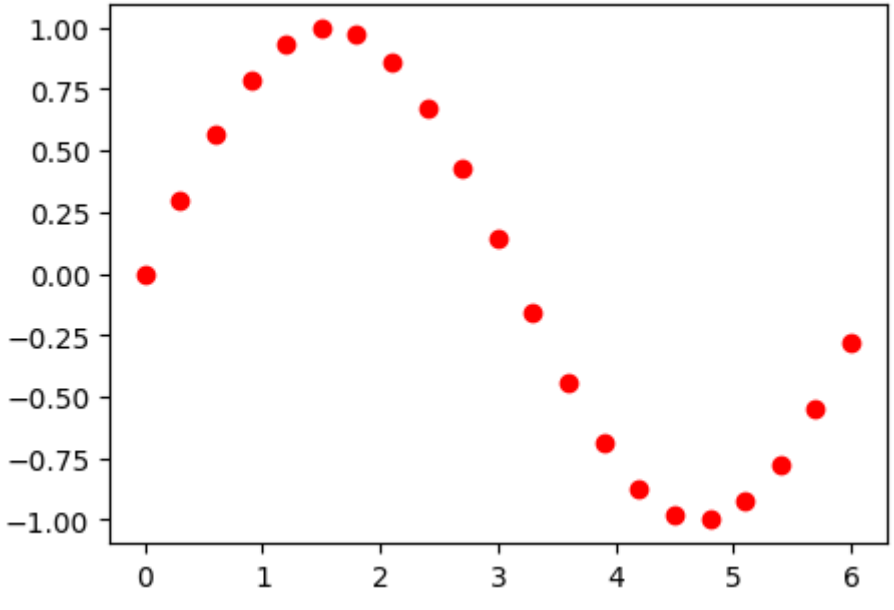
In [126]:
```
plt.plot(x_spaced, np.sin(x_spaced), marker='o', color='red', linestyle='None')
```

Out[126]:
```
[<matplotlib.lines.Line2D at 0x7f6cfd379d68>]
```



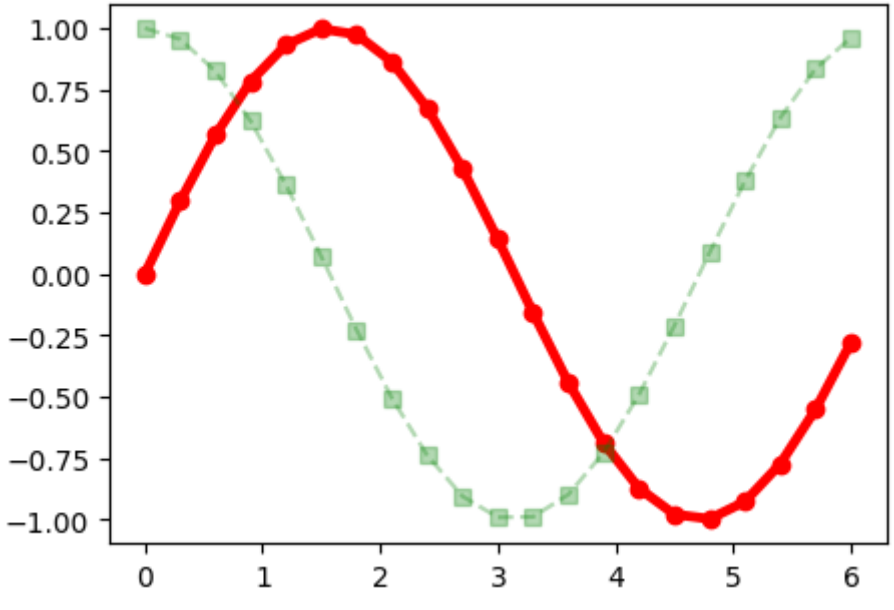
In [127]:
```
plt.plot(x_spaced, np.sin(x_spaced), 'ro') #specify color and marker in one go!
```

Out[127]:
```
[<matplotlib.lines.Line2D at 0x7f6cfd3494e0>]
```



In [128]:
```
plt.plot(x_spaced, np.sin(x_spaced), 'ro', ls='-',lw=3.5)
plt.plot(x_spaced, np.cos(x_spaced), 'gs', ls='--',alpha=0.3)
```

Out[128]:
```
[<matplotlib.lines.Line2D at 0x7f6cfd377828>]
```



Line Styles

- '-' solid line style
- '--' dashed line style
- '-.' dash-dot line style
- ':' dotted line style

Point symbols

- '.' 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

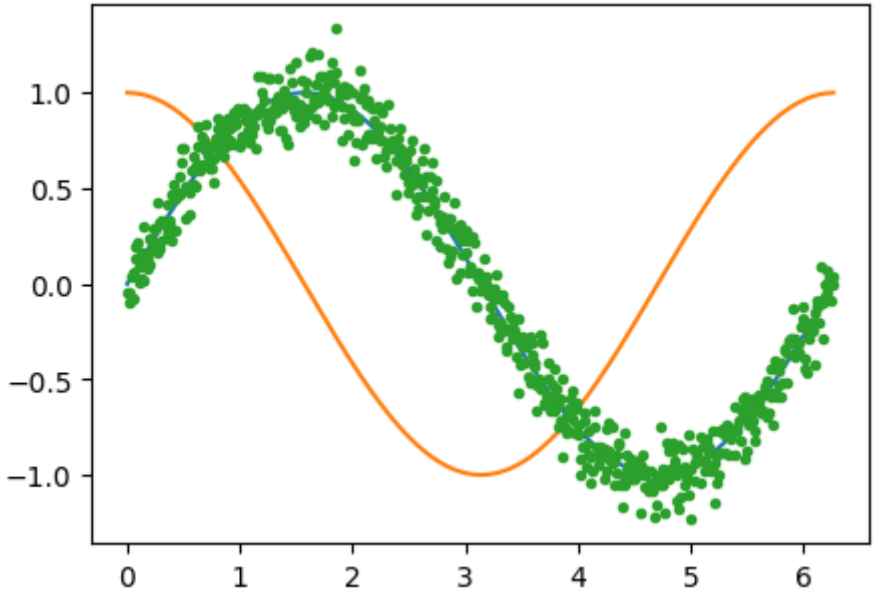
Colours

- 'b' blue
- 'g' green
- 'r' red
- 'c' cyan
- 'm' magenta
- 'y' yellow
- 'k' black
- 'w' white
- '#ff0000' RGB codes
- '(0,1,0,1)' RGB + alpha (transparency)

Adding a Legend

In [129]:
```
plt.plot(x, y, label='Sine(x)')
plt.plot(x, np.cos(x))
y_scatter = y+np.random.normal(0.0, 0.1, len(x))
plt.plot(x, y_scatter, ls='None', marker='.', label='Fake data with scatter')
```

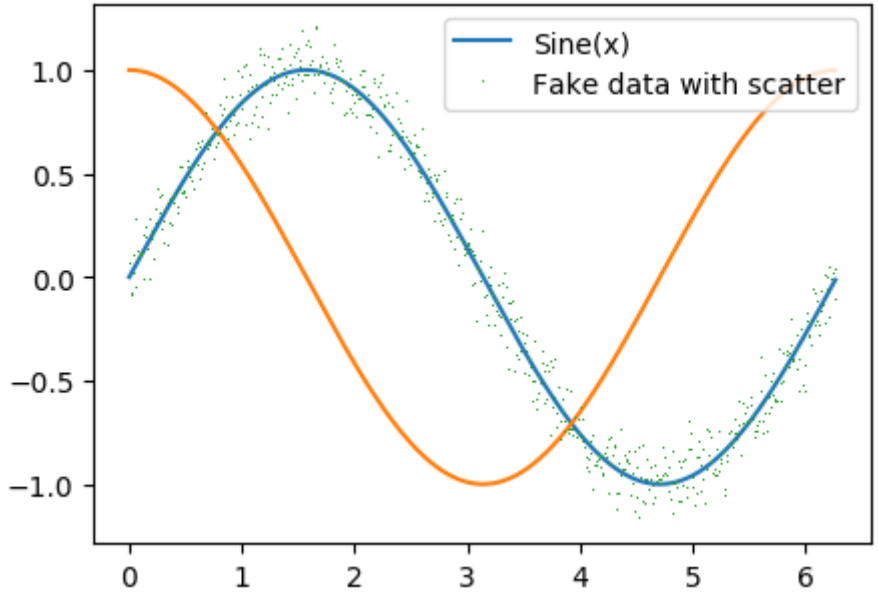
Out[129]:
```
[<matplotlib.lines.Line2D at 0x7f6cfe4f12b0>]
```



np.random.normal() generated len(x) data points with mean 0.0 and sigma 0.1.

In [130]:
```
plt.plot(x, y, label='Sine(x)'); plt.plot(x, np.cos(x))
y_scatter = y+np.random.normal(0.0, 0.1, len(x))
plt.plot(x, y_scatter, ls='None', marker=',', label='Fake data with scatter')
plt.legend()
```

Out[130]:
```
<matplotlib.legend.Legend at 0x7f6cfd23f550>
```

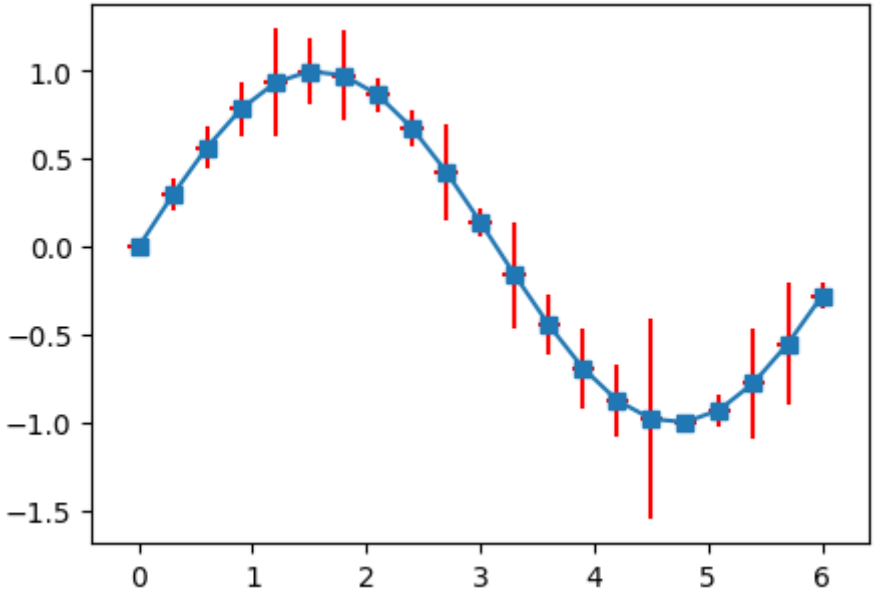


Note that we did not add a "label" keyword in the cos(x) plot, so that is not listed in the legend

Error bars

In [131]:
```
y_error = np.random.normal(0.0, 0.2, len(x_spaced))
x_error = np.repeat(0.1, len(x_spaced))
plt.errorbar(x_spaced, np.sin(x_spaced), yerr=y_error,xerr=x_error,marker='s',e
color='r')
```

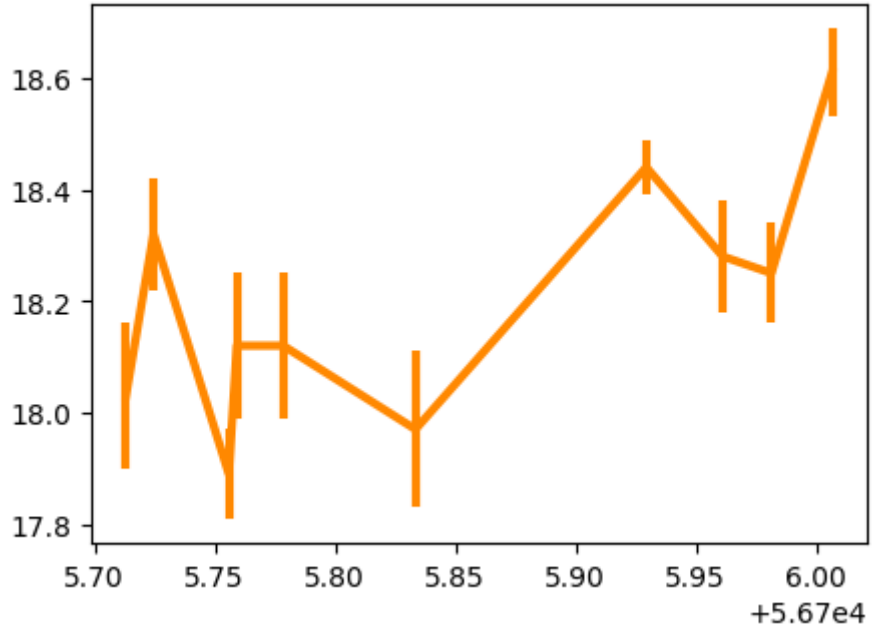
Out[131]:
```
<ErrorbarContainer object of 3 artists>
```



Autoscaling of axes

In [132]:
```
mjd = [56705.7130824, 56705.7245194, 56705.7558403, 56705.7591719, 56705.778438
0,56705.8334423,
       56705.9293922, 56705.9611691, 56705.9814212,  56706.0066355]
# continuing on next line is okay
magnitude = [18.03, 18.32, 17.89, 18.12, 18.12, 17.97, 18.44, 18.28, 18.25, 18.
61]
magerr = [0.13, 0.10, 0.08, 0.13, 0.13, 0.14, 0.05, 0.10, 0.09, 0.08]
plt.errorbar(mjd, magnitude, yerr=magerr,color='#ff8800',lw=3)
```

Out[132]:
```
<ErrorbarContainer object of 3 artists>
```



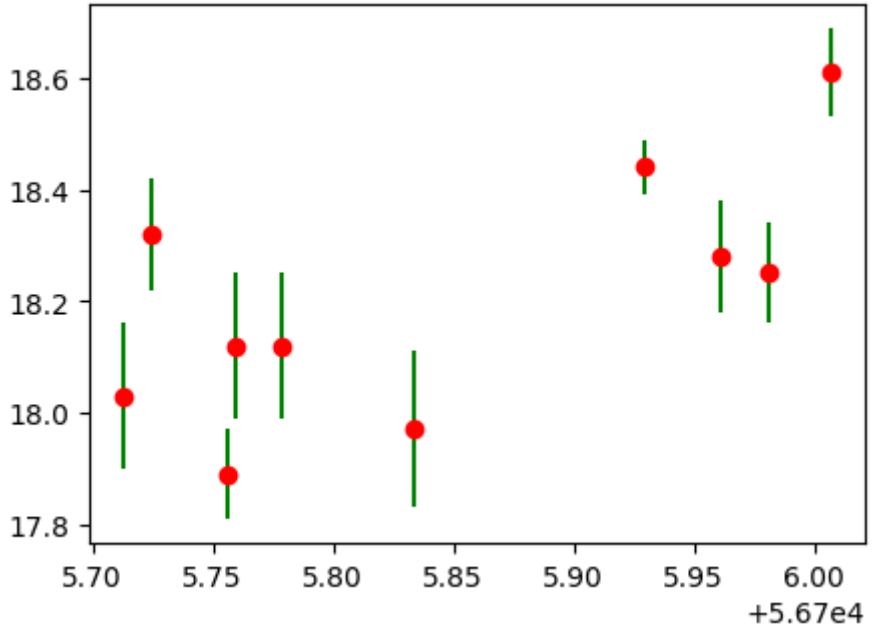
Note the X axis: Matplotlib has automatically subtracted a constant and made the plot more readable.

In [133]:
```
plt.errorbar(mjd, magnitude, yerr=magerr, 'ro') # does not work
```

```
  File "<ipython-input-133-615e366ae5b6>", line 1
    plt.errorbar(mjd, magnitude, yerr=magerr, 'ro') # does not work
                                              ^
SyntaxError: positional argument follows keyword argument
```

In [134]:
```
# instead of shorthand, give the full command
plt.errorbar(mjd, magnitude, yerr=magerr, marker='o', color='r', ecolor='g', ls
='None')
```

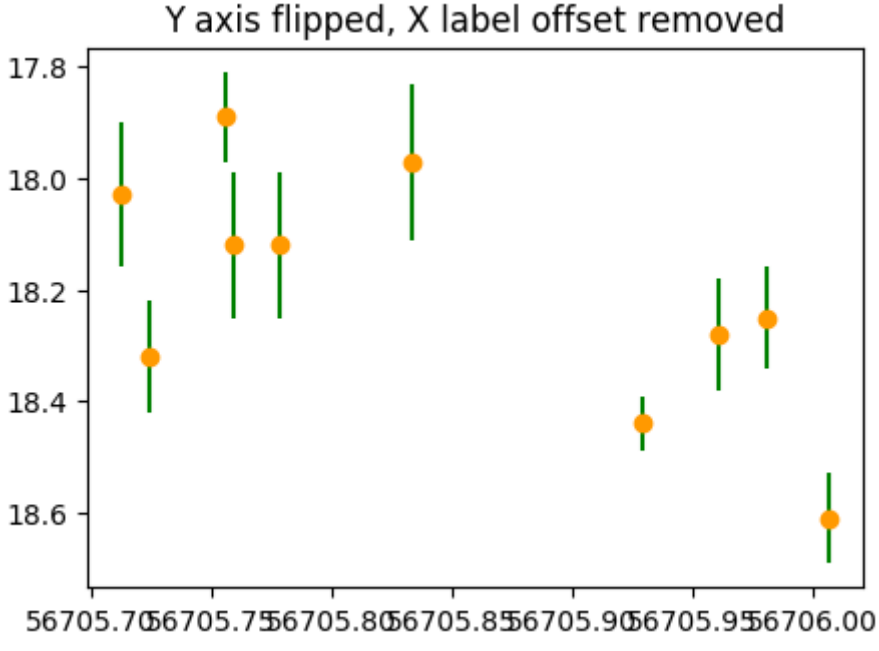
Out[134]:
```
<ErrorbarContainer object of 3 artists>
```



Note: Y axis is still flipped. Higher magnitude is fainter, so it should be plotted below.

In [135]:
```
plt.errorbar(mjd, magnitude, yerr=magerr, marker='o', color='#ff9900', ecolor=
'g', ls='None')
ymin, ymax = plt.ylim()
plt.ylim(ymax, ymin)
plt.ticklabel_format(useOffset=False)
plt.title("Y axis flipped, X label offset removed")
```

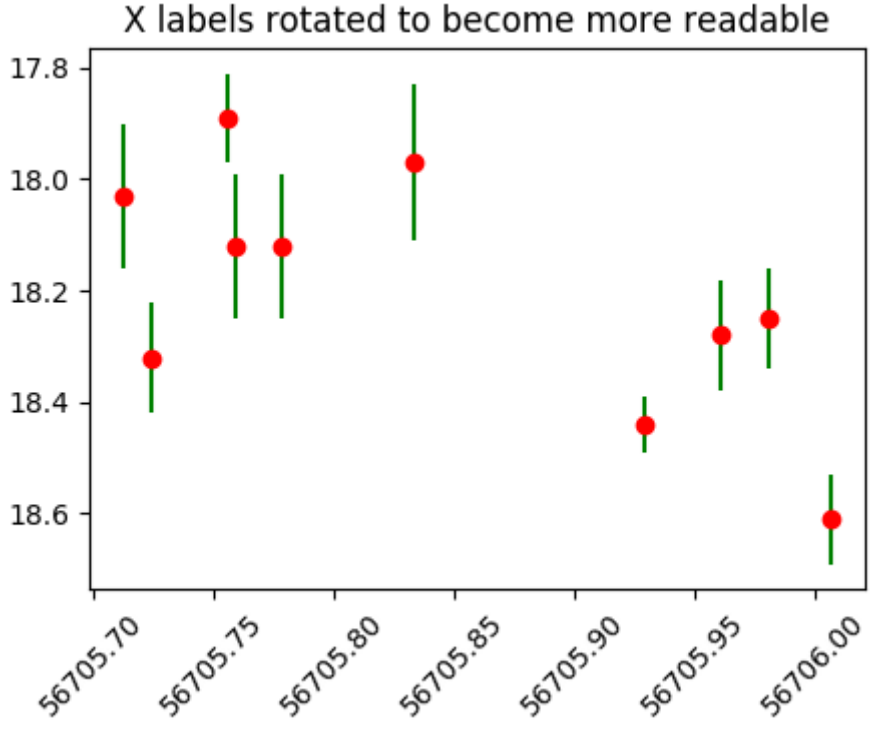
Out[135]:
```
Text(0.5, 1.0, 'Y axis flipped, X label offset removed')
```



X axis labels: the offset is not subtracted anymore, but now labels are hard to read.

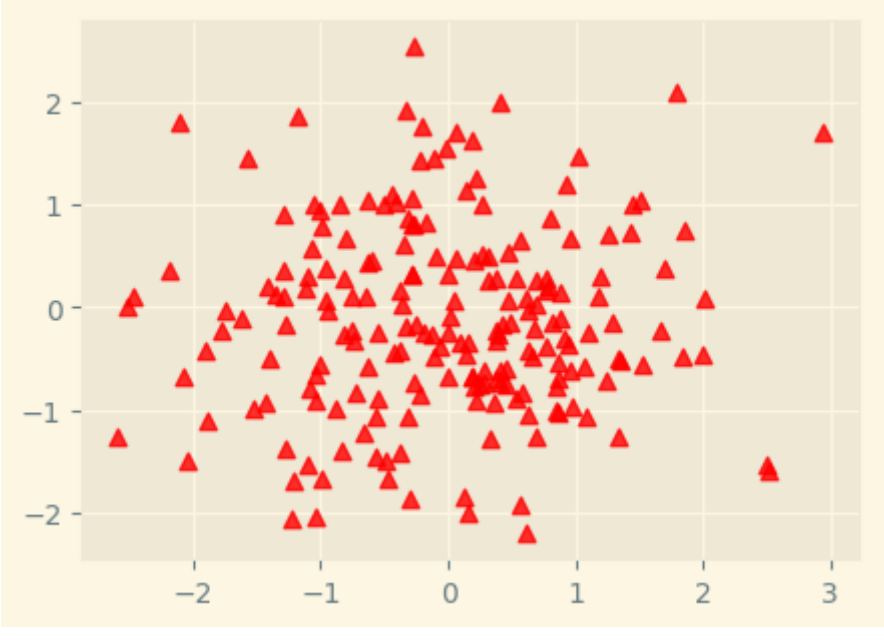
In [136]:
```
plt.errorbar(mjd, magnitude, yerr=magerr, marker='o', color='r', ecolor='g', ls
='None')
ymin, ymax = plt.ylim()
plt.ylim(ymax, ymin)
plt.ticklabel_format(useOffset=False)
plt.xticks(rotation=45)
plt.title("X labels rotated to become more readable")
```

Out[136]:
```
Text(0.5, 1.0, 'X labels rotated to become more readable')
```



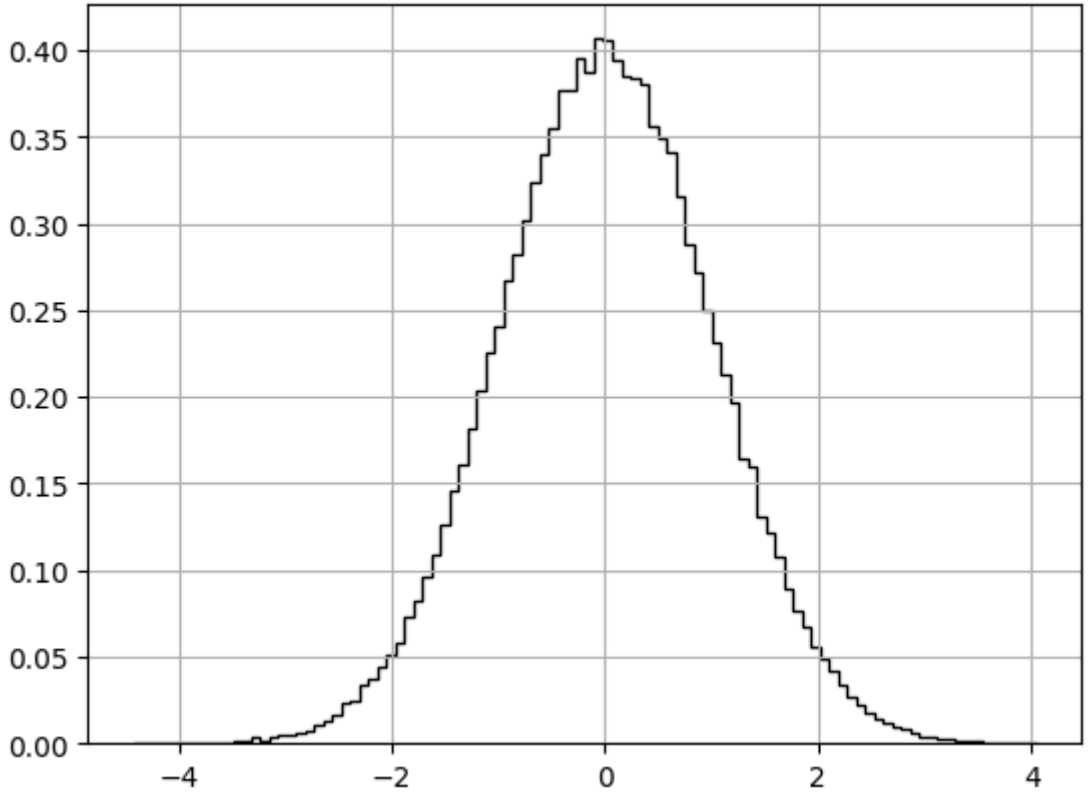
Scatter plots

In [137]:
```
#print (plt.style.available)
plt.style.use('Solarize_Light2')
dist1=np.random.normal(0.,1.,200)
dist2=np.random.normal(0.,1.,200)
plt.scatter(dist1,dist2,c='r',marker='^',alpha=0.8)
plt.style.use('default')
```



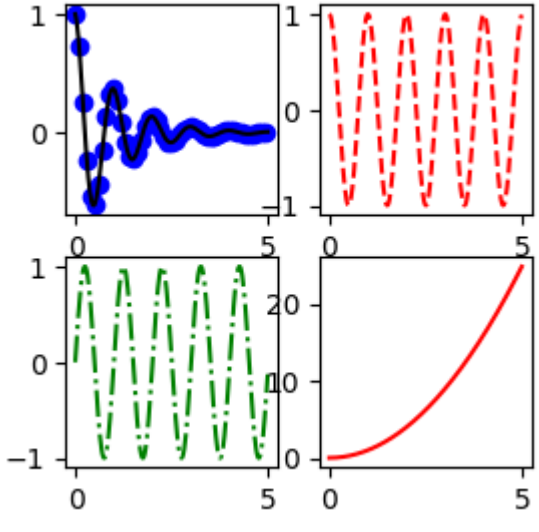
Histogram

In [138]:
```
dist1 = np.random.normal(0.,1.,100000)  #mean 0., std. devn = 1., N sized array
n,bins,patches= plt.hist(dist1,bins=100,color='k',density=True,histtype='step')
plt.grid()
plt.savefig('normaldistribution.png')
```



Subplots

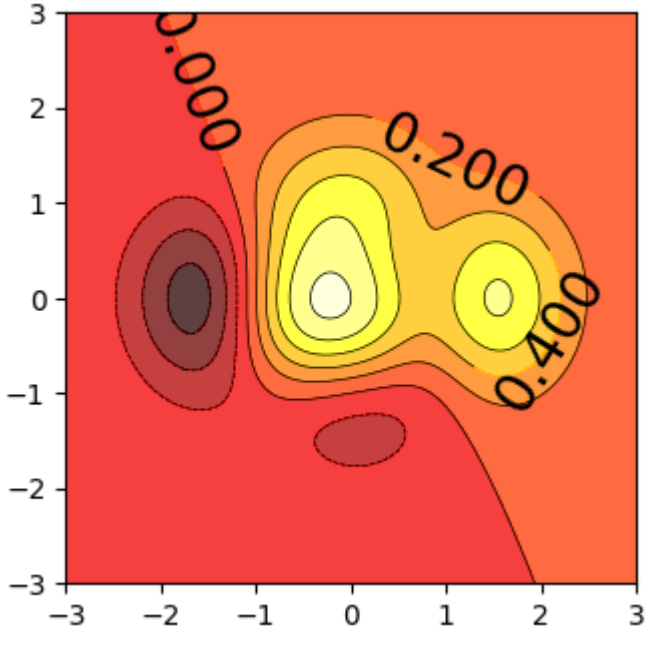
In [139]:
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.rcParams['figure.figsize'] = (3,3)
plt.subplot(221); plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(222); plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.subplot(223); plt.plot(t2,np.sin(2.*np.pi*t2),'g-.')
plt.subplot(224); plt.plot(t2,t2**2,'r-')

plt.savefig('subplots.png'); plt.savefig('subplots.eps'); plt.savefig('subplot
s.svg')
```



Contour plots

In [140]:
```
def f(x,y):
    return (1 - x / 2 + x**5 + y**3) * np.exp(-x**2 -y**2)
n = 256; x = np.linspace(-3, 3, n); y = np.linspace(-3, 3, n)
X,Y = np.meshgrid(x, y)
plt.axes([0.025, 0.025, 0.95, 0.95])
plt.contourf(X, Y, f(X, Y), 8, alpha=.75, cmap=plt.cm.hot)
C = plt.contour(X, Y, f(X, Y), 8, colors='black', linewidths=.5)
plt.clabel(C, inline=1, fontsize=20)
```

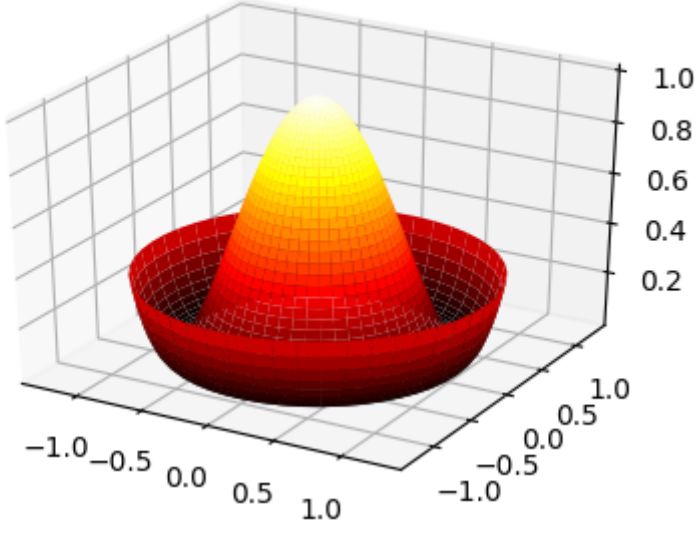
Out[140]:
```
<a list of 3 text.Text objects>
```



3D plotting with mplot3d

In [144]:
```
#See: https://matplotlib.org/stable/gallery/mplot3d/surface3d_radial.html
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(5, 3.5)) # OOP syntax
ax = fig.add_subplot(111, projection='3d')
r = np.linspace(0,1.25,50) # create supporting points in polar coordinates
p = np.linspace(0,2*np.pi,50)
R,P = np.meshgrid(r,p)
X,Y = R*np.cos(P),R*np.sin(P) # transform them to cartesian system
Z = ((R**2 - 1)**2)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.hot)
```

Out[144]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7f6cfcd10b70>
```



OOP versus functional style matplotlib calls

In this talk, I have mostly used the functional programming style syntax, since that may be more familiar to those of you with programming experience with other languages. But you may want to use the OOP style if you prefer.

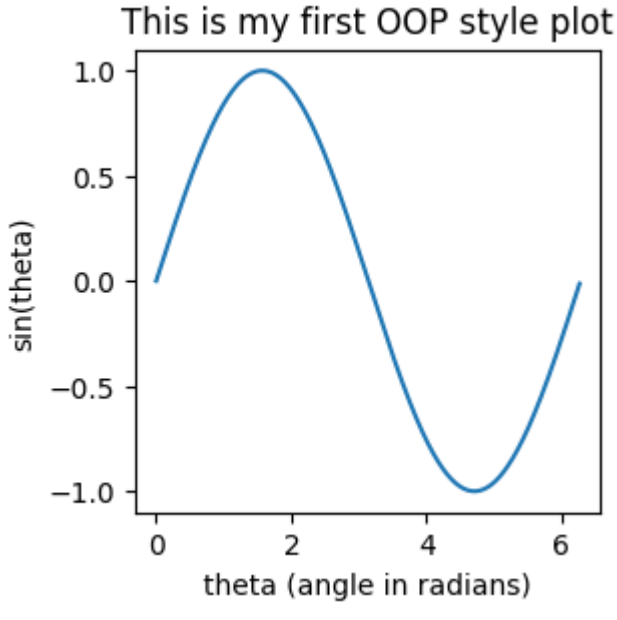
The OOP approach will definitely make life easier if you have very complicated plots eg. a plot with many subfigures.

In [142]:

```
# Generate some test data
x = np.arange(0, 6.28, 0.01)
y = np.sin(x)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x,y)
ax.set_xlabel("theta (angle in radians)")
ax.set_ylabel('sin(theta)')
ax.set_title("This is my first OOP style plot")
```

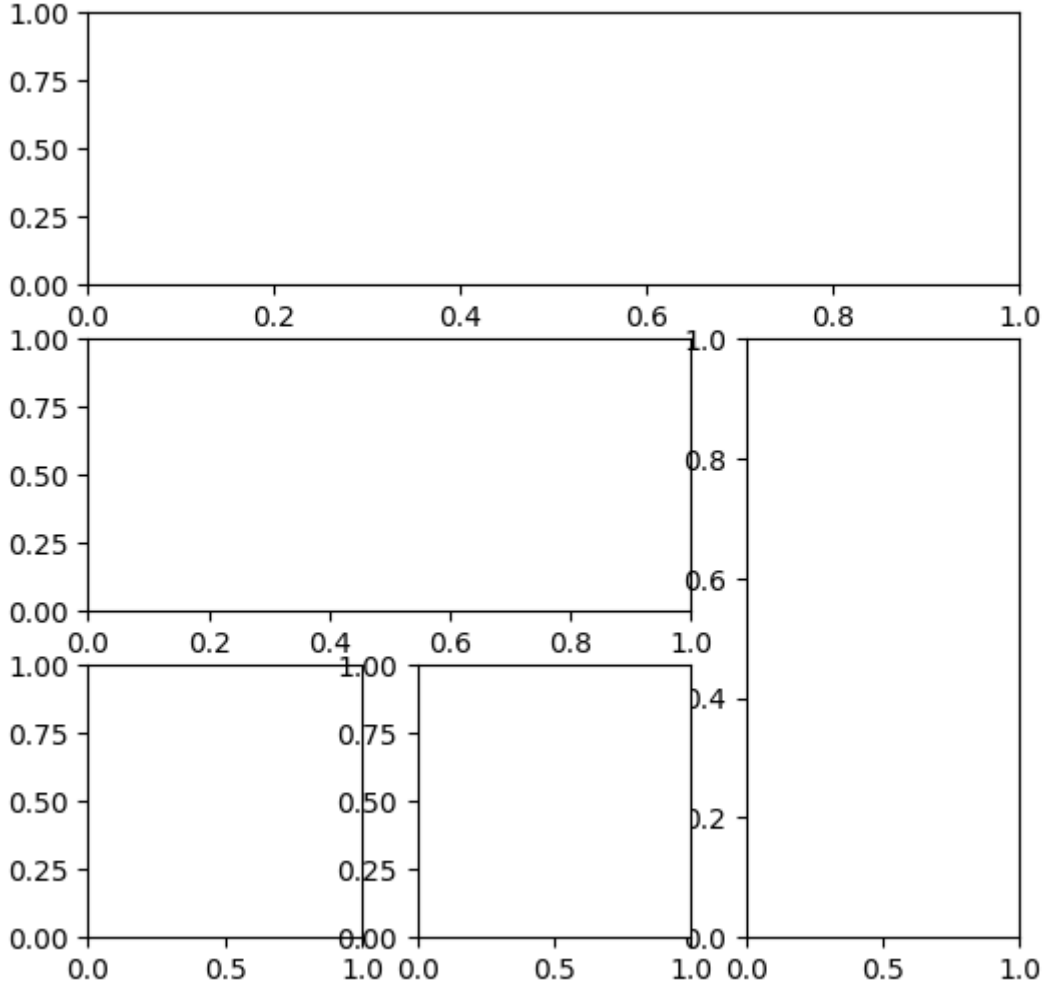
Out[142]:

```
Text(0.5, 1.0, 'This is my first OOP style plot')
```



In [143]:

```
fig = plt.figure(figsize=(6, 6))
gs = plt.GridSpec(3, 3)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, :2])
ax3 = fig.add_subplot(gs[1:, 2])
ax4 = fig.add_subplot(gs[2, 0])
ax5 = fig.add_subplot(gs[2, 1])
```



This talk was a quick tour of matplotlib

Many aspects could not be covered in the short time available. Matplotlib is learnt quickest by trying the examples.

1. Browse demos here: https://matplotlib.org/stable/gallery/index.html (https://matplotlib.org/stable/gallery/index.html)
2. Find the plot that is closest to your requirement
3. Modify the code to suit your requirement
4. **Remember: making good plots takes a lot of time and effort. Look at nice examples in papers you read.**

What I have not covered: **aplpy** APLpy (the Astronomical Plotting Library in Python) is a Python module aimed at producing publication-quality plots of astronomical imaging data in FITS format. For complete control of complicated astronomical images use **WCSAxes** from astropy

Seaborn is a powerful module built on top of matplotlib and creates very aesthetic figures. You could use it to produce the final figures for your paper.