ASTR 119: Session 6
Matplotlib, Multipanel plots;
Line fitting

Outline

1) Homework due 10/22 at 8:00am
2) Attend your assigned lecture, and some comments about Slack
3) Visualization of the Day
4) Matplotlib basics
5) Our first figure, saving figures to a file
6) Multipanel figures, aspect ratios, legends
7) Line fitting
8) Save your work to GitHub

Homework, due Oct 22, 8:00am

1) Make a Jupyter notebook, import numpy and matplotlib, and in a single line use numpy to create an array x running x = [0, 2 * pi] inclusive with 1000 values. The use matplotlib to plot the following functions on a single plot, using the x range x= [0,2*pi] and the vertical range y = [-1, 10]:

    a). y(x) = 5.5 cos(2 * x) + 5.5
    b). y(x) = 0.02 * exp(x)
    c). y(x) = 0.25 * x^2 + 0.1 sin (10 * x)

2) Make the plot's y label "Measures of Awesomeness" and the x label "Time in ASTR / EART 119".

3) Create an issue for your repository and tag your TAs. CLEAR ALL THE CELLS BEFORE YOU COMMIT THE NOTEBOOK.

4) Your TA will clone your code and email you commented version of the code and a grade. To get the full grade possible, all the notebooks will need to run to completion without errors and produce the requested plots.

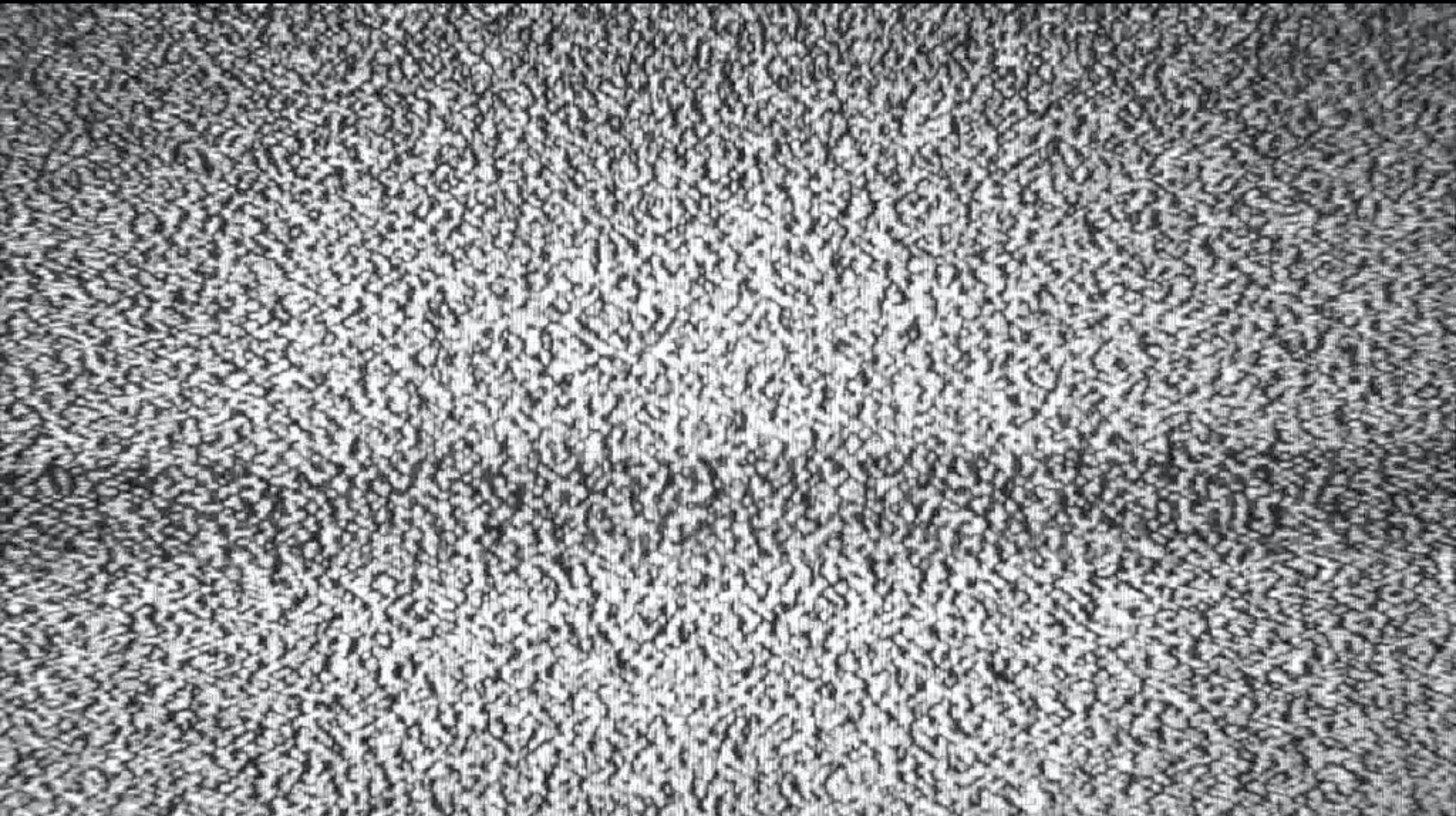
5) Call the repository "astr-119-hw-3" and the notebook "hw-3.ipynb".

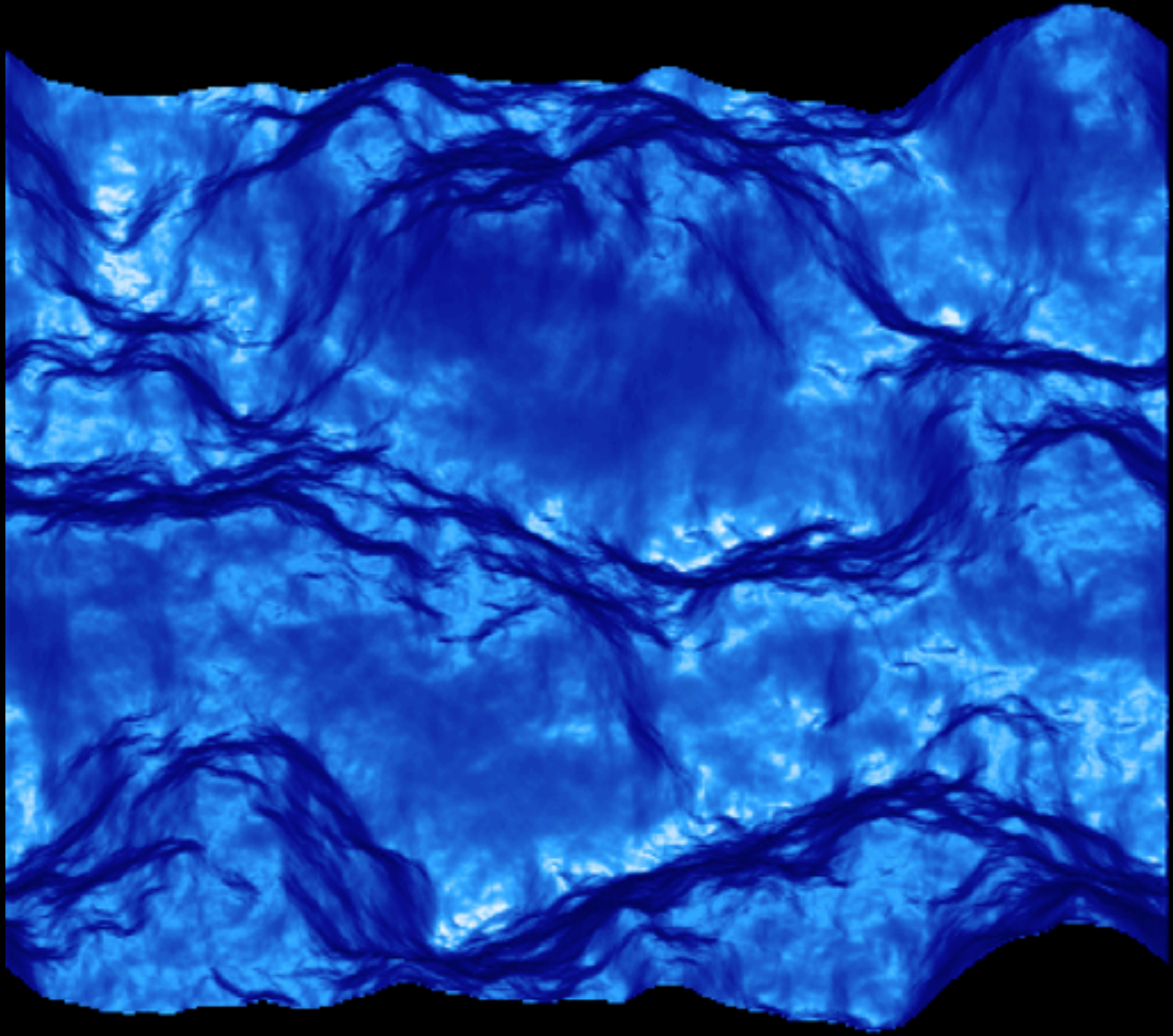
Lecture, Slack, and Help

1) You may only attend the lecture for which you are registered.

2) Technical and homework help over Slack will be limited. Please come to office hours and attend your section for better support.

3) When asking for help, please provide as much detail as possible about the issue you encounter. Is there an error message? Did you do anything "custom" when configuring your software?





matplotlib.pyplot

Version 3.0.0

Fork me on GitHub

home | examples | tutorials | API | docs » The Matplotlib API » Pyplot function overview »

previous | next | modules | index

matplotlib.pyplot

matplotlib.pyplot is a state-based interface to matplotlib. It provides a MATLAB-like way of plotting.

pyplot is mainly intended for interactive plots and simple cases of programmatic plot generation:

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

x = np.arange(0, 5, 0.1)
y = np.sin(x)
plt.plot(x, y)
```

The object-oriented API is recommended for more complex plots.

Functions

acorr(x, *[, data])	Plot the autocorrelation of x.
angle_spectrum(x[, Fs, Fc, window, pad_to, ...])	Plot the angle spectrum.
annotate(text, xy, *args, **kwargs)	Annotate the point xy with text s.
arrow(x, y, dx, dy, **kwargs)	Add an arrow to the axes.
autoscale([enable, axis, tight])	Autoscale the axis view to the data (toggle).

Quick search

Go

Table Of Contents

matplotlib.pyplot

- Functions

Related Topics

Documentation overview

- The Matplotlib API
  - Pyplot function overview
    - Previous: Pyplot function overview
    - Next: matplotlib.pyplot.acorr

Show Page Source

jupyter test_matplotlib Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Trusted

Python 3

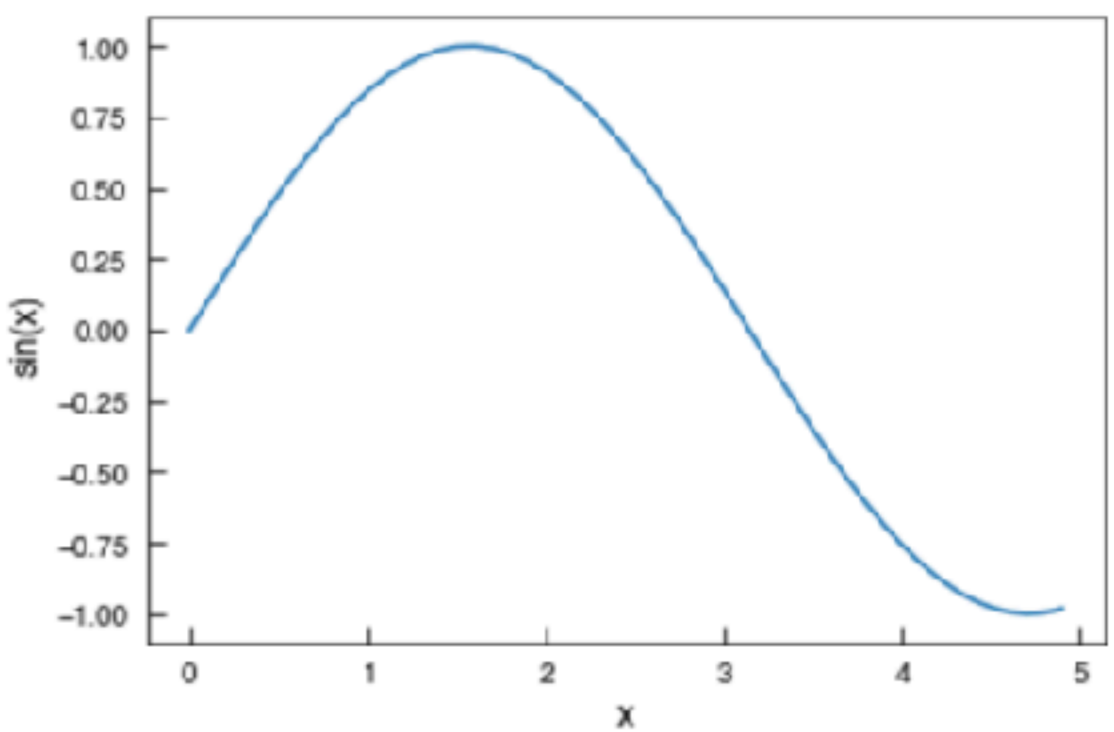
Run Code

In [1]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:

```
x = np.arange(0, 5, 0.1)  #create x=[0..5] in 0.1 increments
y = np.sin(x)             # y = sin(x)
plt.plot(x,y)             # make a plot
plt.xlabel('x')           # label x axis
plt.ylabel('sin(x)')      # label y axis
plt.show()                # show the plot on the screen
```



Matplotlib.pyplot

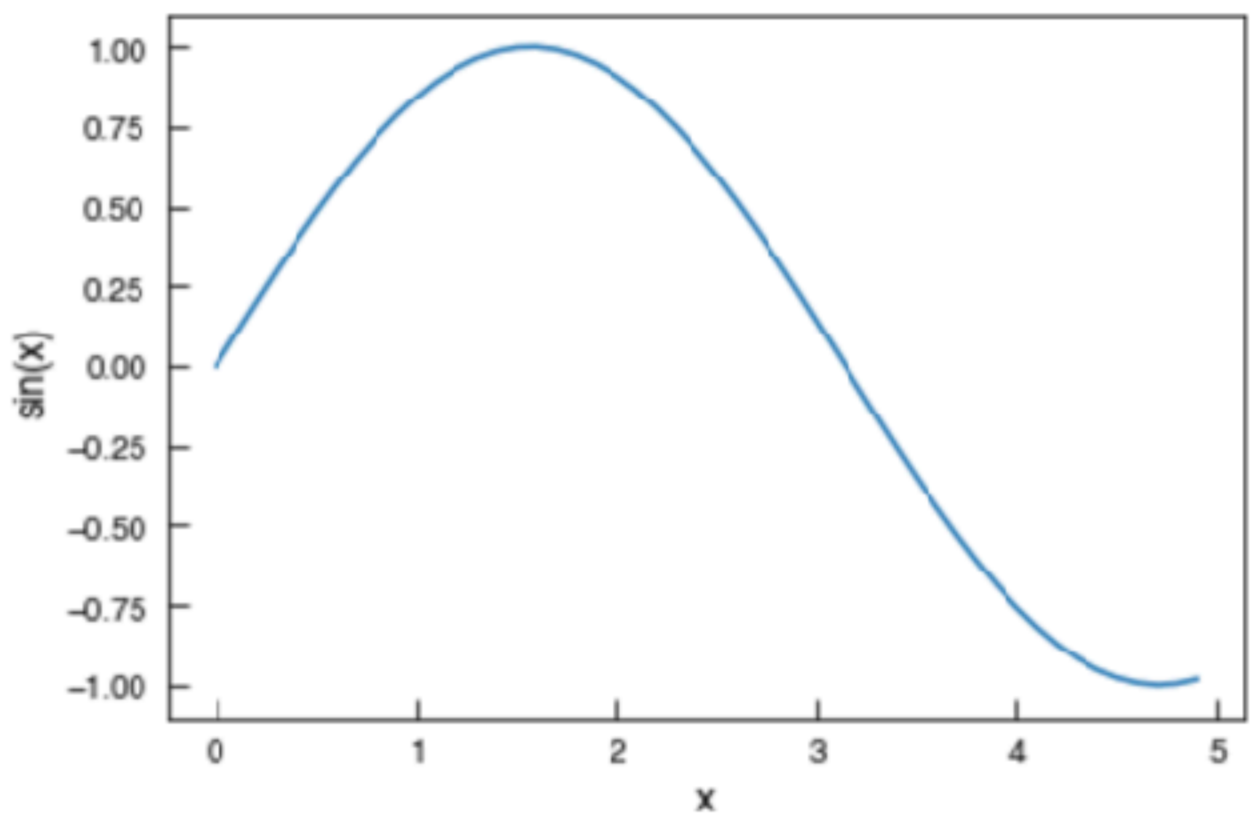
jupyter test_matplotlib Last Checkpoint: 14 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Help

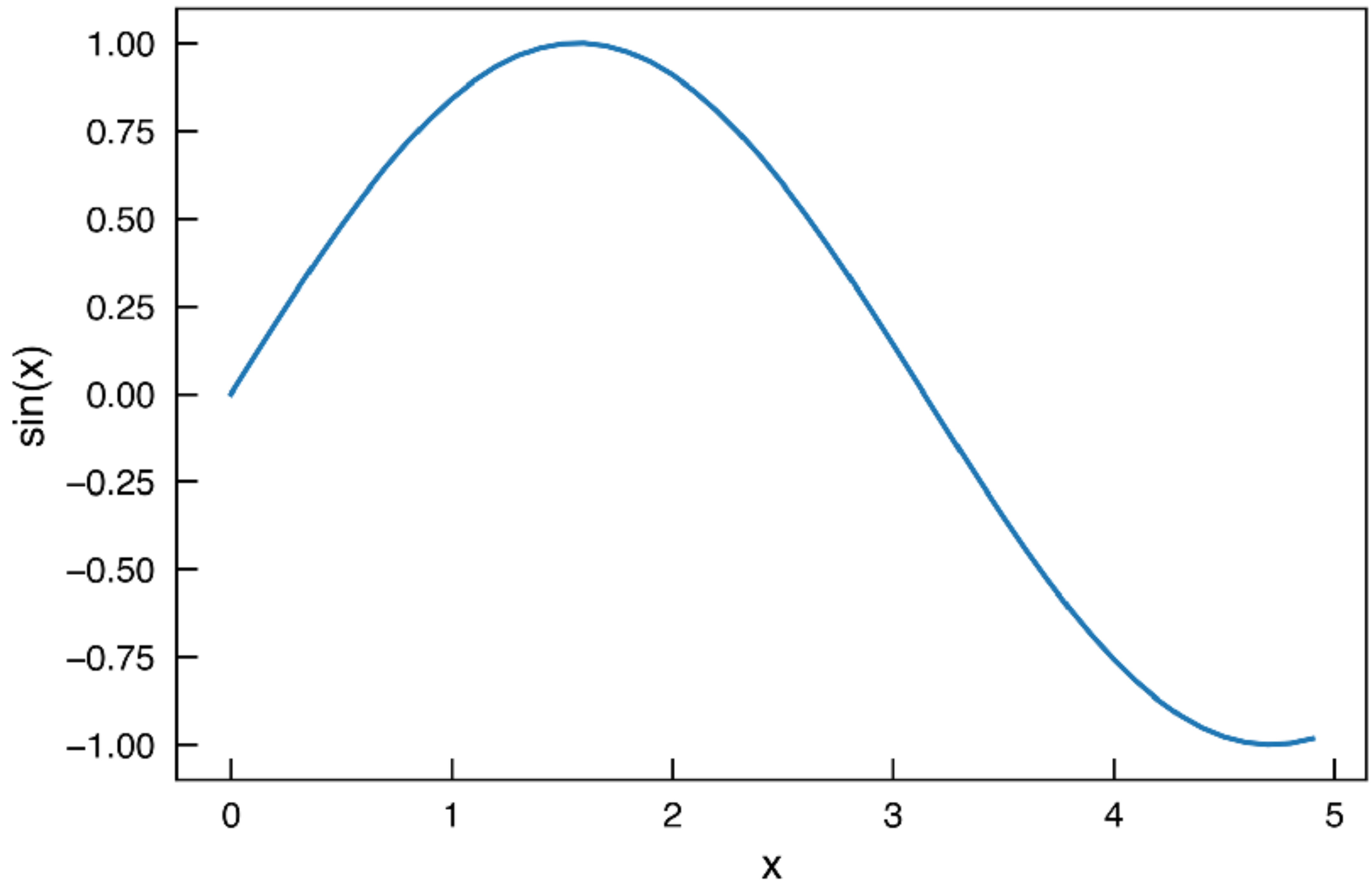
Run Code

In [5]:

```
x = np.arange(0, 5, 0.1)   #create x=[0..5] in 0.1 increments
y = np.sin(x)              # y = sin(x)
plt.plot(x,y)              # make a plot
plt.xlabel('x')            # label x axis
plt.ylabel('sin(x)')       # label y axis
plt.savefig('sinx.png',bbox_inches="tight",dpi=600)
```



plt.savefig()



plt.savefig()

Multipanel Figures

Using Matplotlib, we can make multi panel figures easily.

Making multipanel plots with matplotlib

First, we import numpy and matplot lib as usual

In [2]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Then we define an array of angles, and their sines and cosines using numpy. This time we will use linspace

In [30]:
```
x = np.linspace(0,2*np.pi,100)
print(x[-1],2*np.pi)

y = np.sin(x)
z = np.cos(x)
w = np.sin(4*x)
v = np.cos(4*x)
```

```
6.28318530718 6.283185307179586
```

Multipanel Figures

Using Matplotlib, we can make multi panel figures easily.

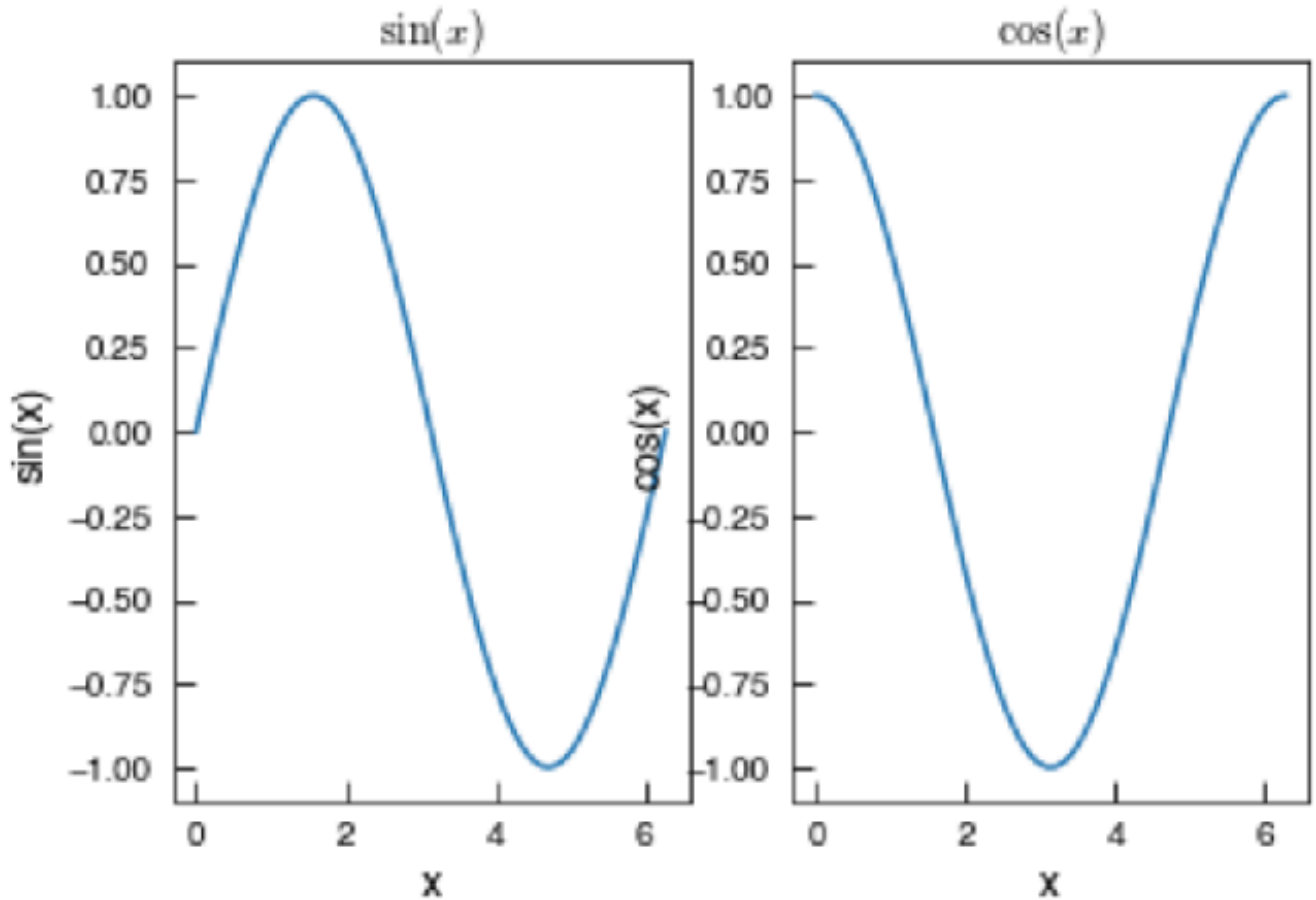
Now, let's make a two panel plot side-by-side

In [ ]:
```
#call subplots to generate a multipanel figure.  This means 1 row, 2 columns of figures
f, axarr = plt.subplots(1, 2)

#treat axarr as an array, from left to right

#first panel
axarr[0].plot(x, y)
axarr[0].set_xlabel('x')
axarr[0].set_ylabel('sin(x)')
axarr[0].set_title(r'$\sin(x)$')
#second panel
axarr[1].plot(x, z)
axarr[1].set_xlabel('x')
axarr[1].set_ylabel('cos(x)')
axarr[1].set_title(r'$\cos(x)$')
```

Multipanel Figures

Using Matplotlib, we can make multi panel figures easily.

```
Out[54]: Text(0.5,1,'$\\cos(x)$')
```



Multipanel Figures

subplots_adjust() enables us to move the panels further apart

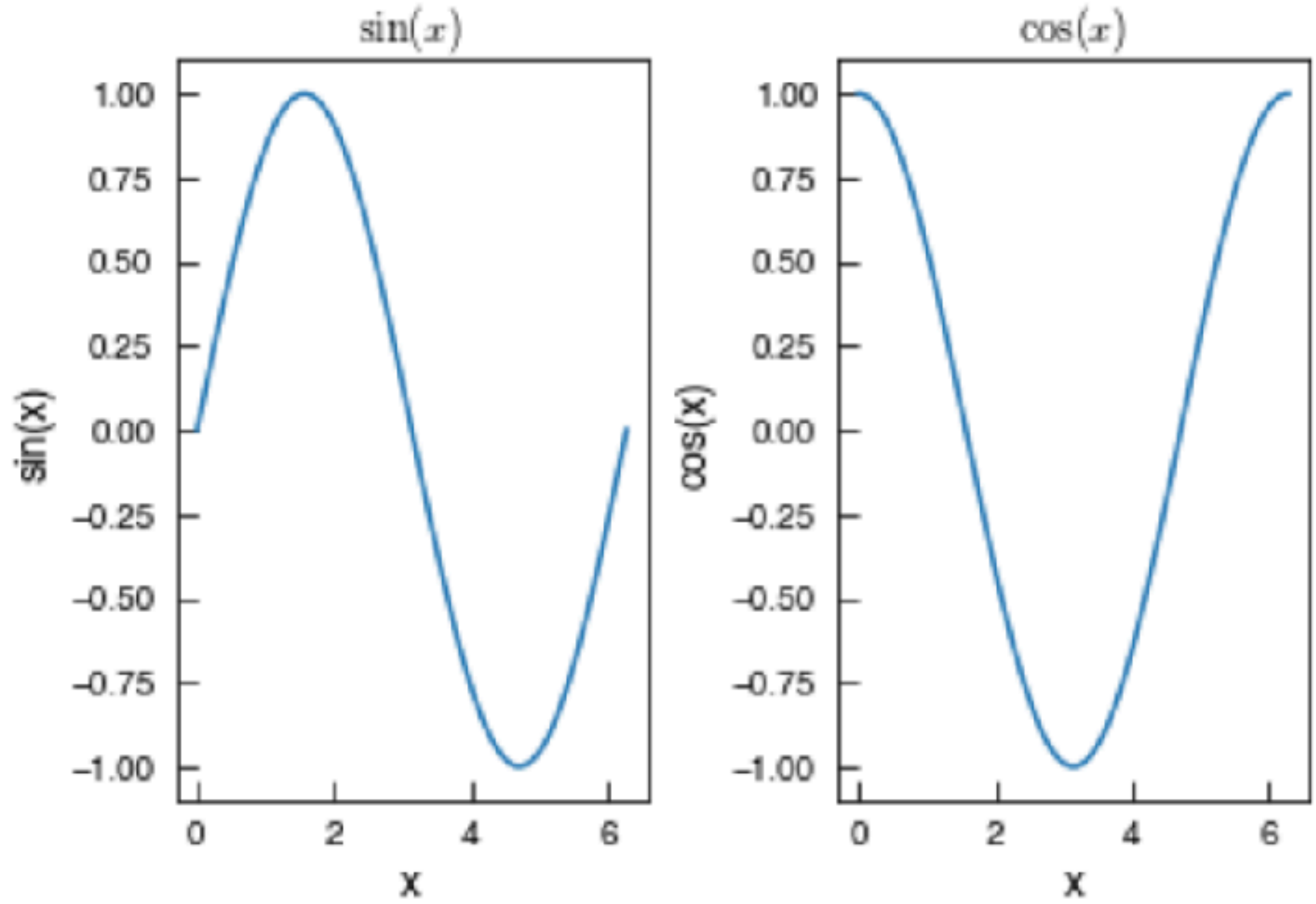
Here we can see that matplotlib has the panels too close together.

We can adjust that using the subplots_adjust() function.

In [ ]:
```
#call subplots to generate a multipanel figure.  This means 1 row, 2 columns of figures
f, axarr = plt.subplots(1, 2)

#treat axarr as an array, from left to right

#first panel
axarr[0].plot(x, y)
axarr[0].set_xlabel('x')
axarr[0].set_ylabel('sin(x)')
axarr[0].set_title(r'$\sin(x)$')
#second panel
axarr[1].plot(x, z)
axarr[1].set_xlabel('x')
axarr[1].set_ylabel('cos(x)')
axarr[1].set_title(r'$\cos(x)$')

#add more space between the figures
f.subplots_adjust(wspace=0.4)
```

Multipanel Figures

subplots_adjust() enables us to move the panels further apart



Multipanel Figures

We can use set_aspect() to change the axis ratio.

OK, but the axis ratios are all squished. Let's fix that too.

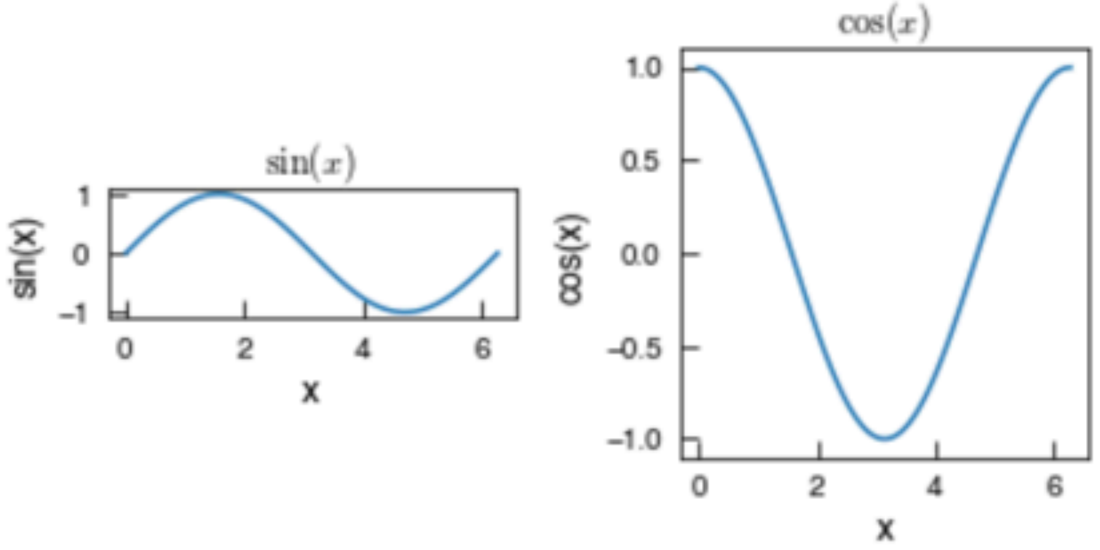
In [ ]:
```
#call subplots to generate a multipanel figure.  This means 1 row, 2 columns of figures
f, axarr = plt.subplots(1, 2)

#treat axarr as an array, from left to right

#first panel
axarr[0].plot(x, y)
axarr[0].set_xlabel('x')
axarr[0].set_ylabel('sin(x)')
axarr[0].set_title(r'$\sin(x)$')
#second panel
axarr[1].plot(x, z)
axarr[1].set_xlabel('x')
axarr[1].set_ylabel('cos(x)')
axarr[1].set_title(r'$\cos(x)$')

#add more space between the figures
f.subplots_adjust(wspace=0.4)

#fix the axis ratio
#here are two possible options
axarr[0].set_aspect('equal')  #make the ratio of the tick units equal, a bit counter intuitive
axarr[1].set_aspect(np.pi)    #make a square by setting the aspect to be the ratio of the tick unit range
```

UC SANTA CRUZ

Multipanel Figures

We can use set_aspect() to change the axis ratio.

```
#fix the axis ratio
#here are two possible options
axarr[0].set_aspect('equal')  #make the ratio of the tick units equal, a bit counter intuitive
axarr[1].set_aspect(np.pi)    #make a square by setting the aspect to be the ratio of the tick unit range
```



Legends

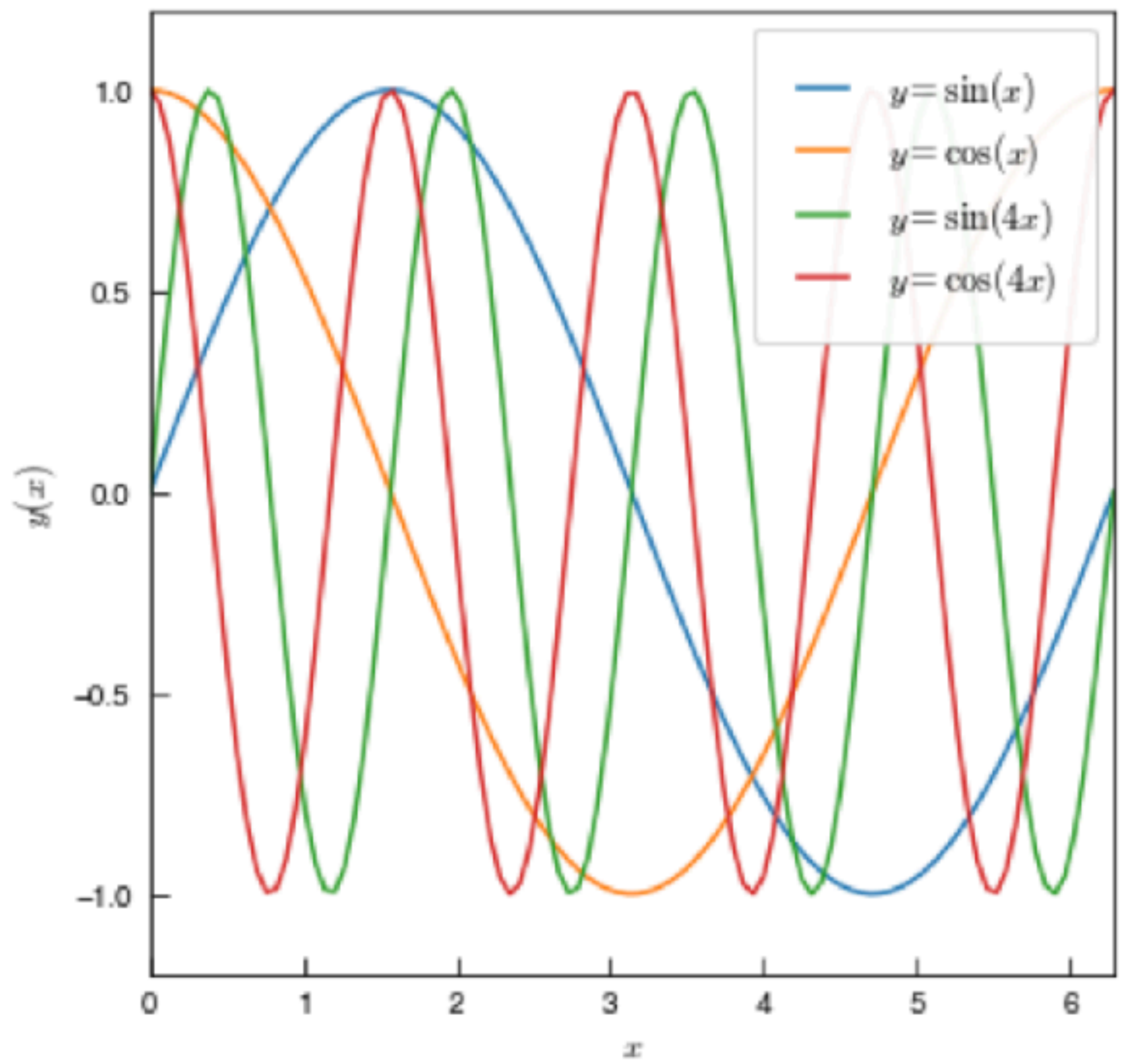
Legends are an easy way to notate a complicated figure.

Alright, let's keep the square figure, merge them into one, remove the titles and add legends

In [ ]:
```
#adjust the size of the figure
fig = plt.figure(figsize=(6,6))


plt.plot(x, y, label=r'$y = \sin(x)$') #add a label to the line
plt.plot(x, z, label=r'$y = \cos(x)$') #add a label to the second line
plt.plot(x, w, label=r'$y = \sin(4x)$') #add a label to the third line
plt.plot(x, v, label=r'$y = \cos(4x)$') #add a label to the fourth line

plt.xlabel(r'$x$')                      #note set_xlabel vs. xlabel
plt.ylabel(r'$y(x)$')                   #note set_ylabel vs. ylabel
plt.xlim([0,2*np.pi])                   #note set_xlim vs. xlim
plt.ylim([-1.2,1.2])                    #note set_ylim vs. ylim
plt.legend(loc=1,framealpha=0.95)       #add a legend with a semi-transparent frame in the upper RH corner

#fix the axis ratio
plt.gca().set_aspect(np.pi/1.2)  #use "gca" to get current axis()
```

Legends

Legends are an easy way to notate a complicated figure.



Line Fitting

Using python, we can quickly perform least squares line fitting

Example of performing linear least squares fitting

First we import numpy and matplotlib as usual.

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

Now, let's generate some random data about a trend line.

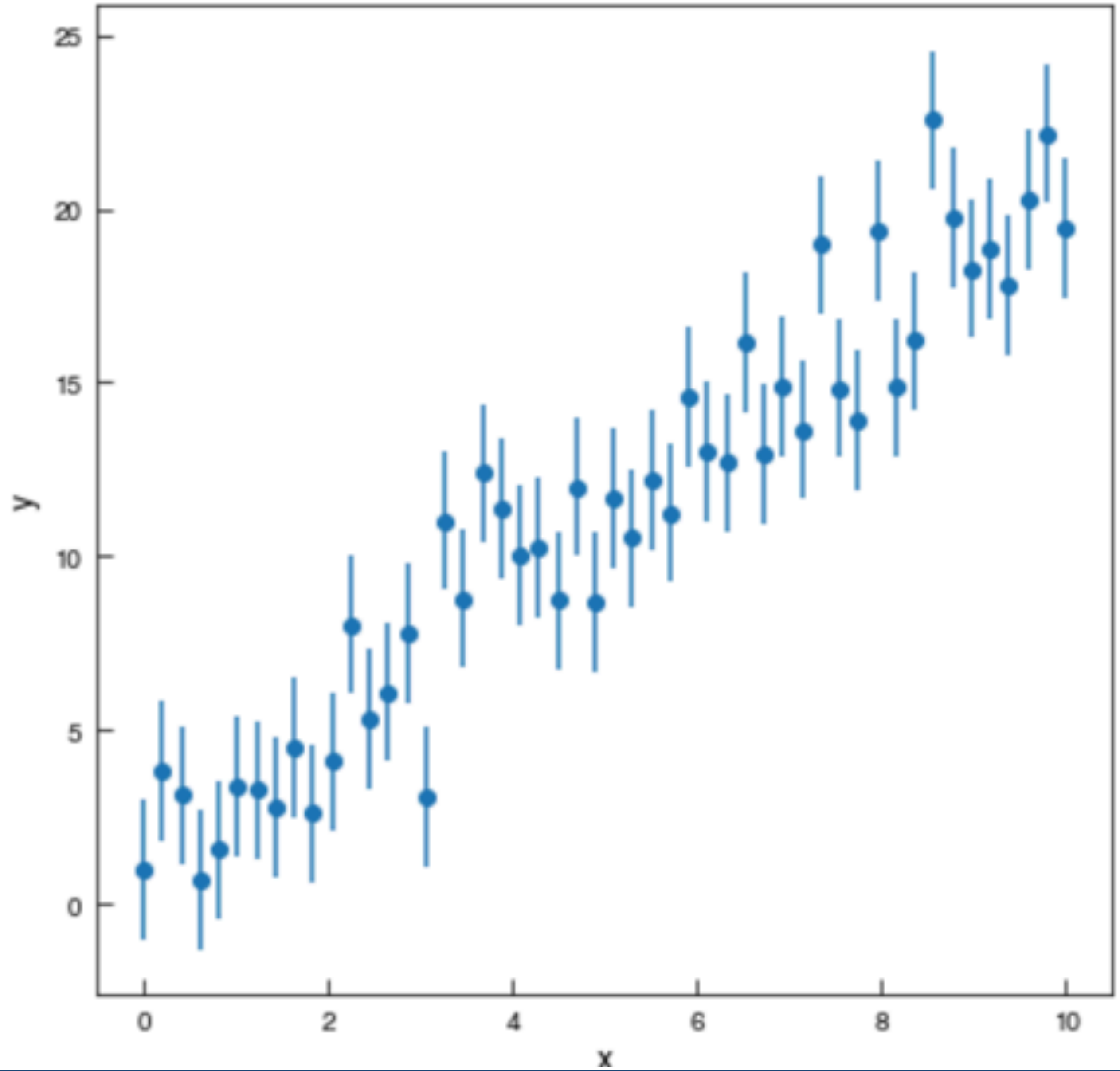
In [20]:
```
#set a random number seed
np.random.seed(119)

#set number of data points
npoints = 50

#set x
x = np.linspace(0,10.,npoints)

#set slope, intercept, and scatter rms
m = 2.0
b = 1.0
sigma = 2.0

#generate y points
y = m*x + b + np.random.normal(scale=sigma,size=npoints)
y_err = np.full(npoints,sigma)
```

Line Fitting

Let's just plot the data first

In [14]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,sigma,fmt='o')
plt.xlabel('x')
plt.ylabel('y')
```

Out[14]: `Text(0,0.5,'y')`



Line Fitting

Method #1, polyfit()

In [28]:

```
m_fit, b_fit = np.poly1d(np.polyfit(x, y, 1, w=1./y_err))  #weight with uncertainties
print(m_fit, b_fit)

y_fit = m_fit * x + b_fit
```

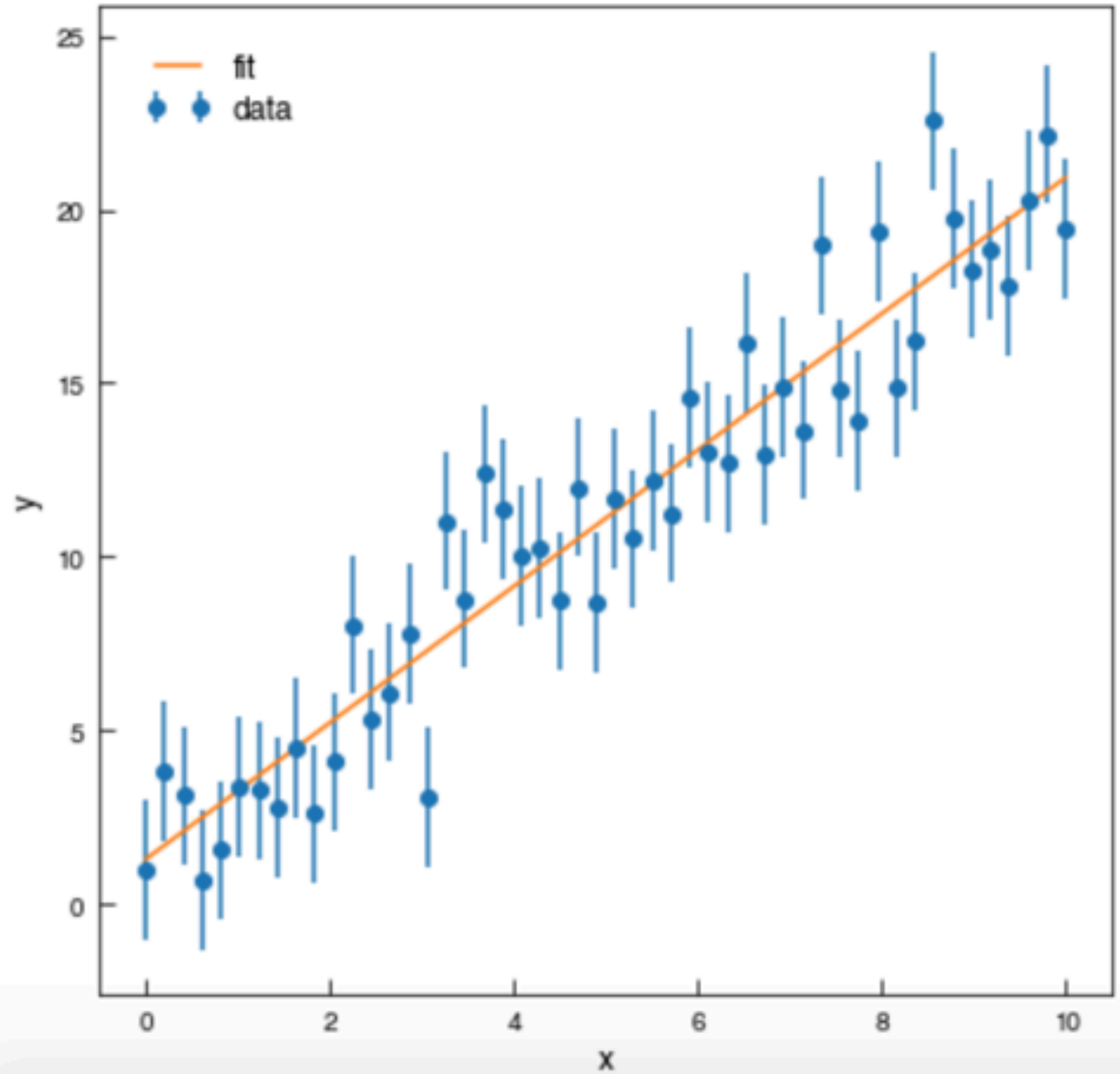
```
1.96340434704 1.2830106813
```

Line Fitting

Plot result

In [29]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o',label='data')
plt.plot(x,y_fit,label='fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=2,frameon=False)
```

Out[29]: <matplotlib.legend.Legend at 0x10fbbcef0>



Line Fitting

Method #2, scipy + optimize

In [31]:

```
#import optimize from scipy
from scipy import optimize

#define the function to fit
def f_line(x, m, b):
    return m*x + b

#perform the fit
params, params_cov = optimize.curve_fit(f_line,x,y,sigma=y_err)

m_fit = params[0]
b_fit = params[1]
print(m_fit,b_fit)
```

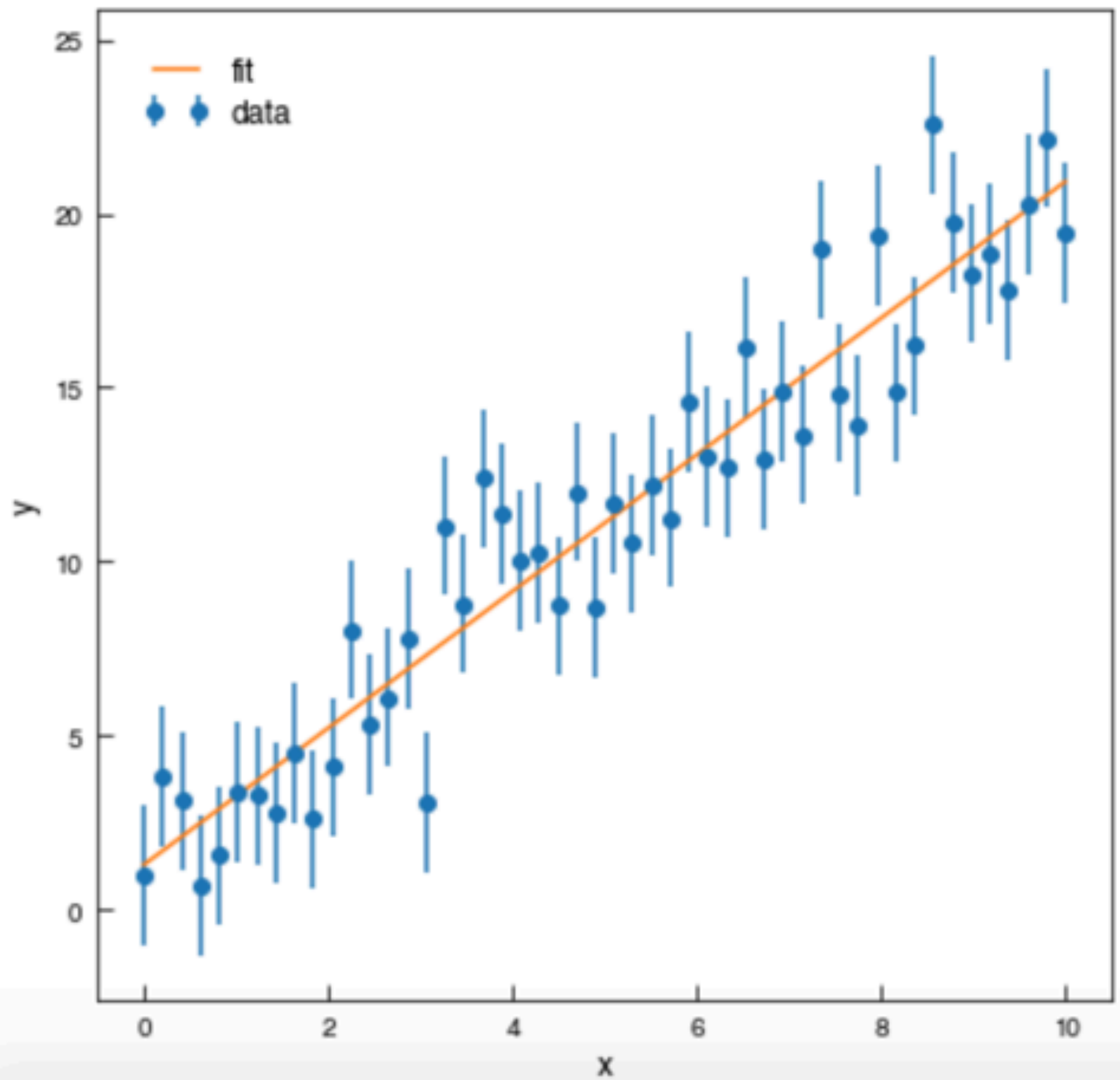
```
1.96340434575 1.28301068905
```

Line Fitting

Plot the result

In [33]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o',label='data')
plt.plot(x,y_fit,label='fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=2,frameon=False)
```

Out[33]: <matplotlib.legend.Legend at 0x112928fd0>



Using pip to install scipy

Mac OSX / Linux:

```
$ pip3 install scipy
```

Windows:

```
$ py -3  -m pip install scipy
```

Line Fitting

Method #2, scipy + optimize

In [31]:
```
#import optimize from scipy
from scipy import optimize

#define the function to fit
def f_line(x, m, b):
    return m*x + b

#perform the fit
params, params_cov = optimize.curve_fit(f_line,x,y,sigma=y_err)

m_fit = params[0]
b_fit = params[1]
print(m_fit,b_fit)
```

```
1.96340434575 1.28301068905
```

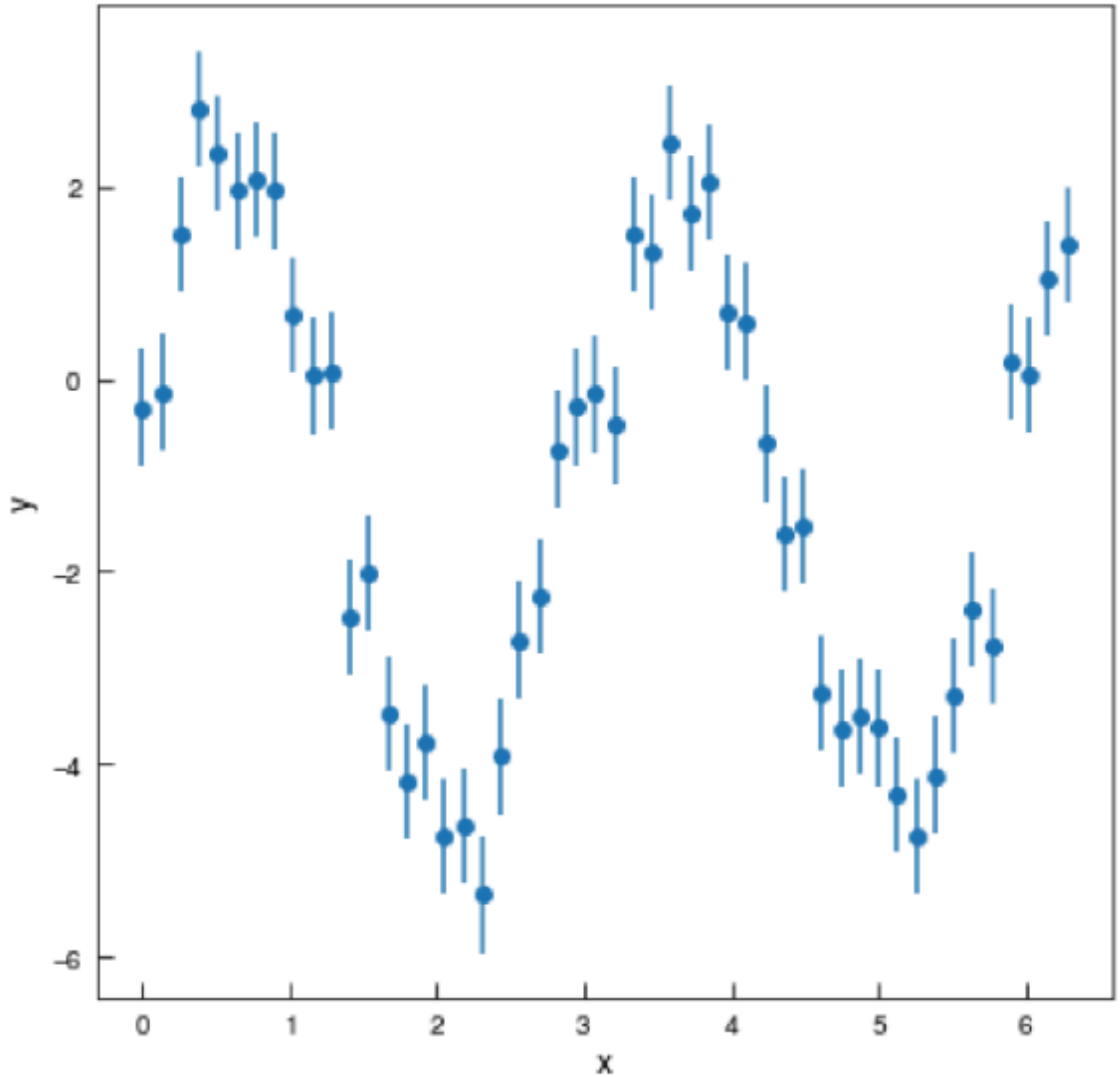
Line Fitting

We can perform much more complicated fits....

In [38]:
```
#redefine x and y
npoints = 50
x = np.linspace(0.,2*np.pi,npoints)

#make y a complicated function
a = 3.4
b = 2.1
c = 0.27
d = -1.3
sig = 0.6

y = a * np.sin( b*x + c) + d + np.random.normal(scale=sig,size=npoints)
y_err = np.full(npoints,sig)

f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o')
plt.xlabel('x')
plt.ylabel('y')
```

Line Fitting

```
Out[38]: Text(0,0.5,'y')
```



Line Fitting

Perform a fit using scipy.optimize.curve_fit()

In [45]:

```
#import optimize from scipy
from scipy import optimize

#define the function to fit
def f_line(x, a, b, c, d):
    return a * np.sin( b*x + c) + d

#perform the fit
params, params_cov = optimize.curve_fit(f_line,x,y,sigma=y_err,p0=[1,2.,0.1,-0.1])

a_fit = params[0]
b_fit = params[1]
c_fit = params[2]
d_fit = params[3]

print(a_fit,b_fit,c_fit,d_fit)

y_fit = a_fit * np.sin(b_fit * x + c_fit) + d_fit
```

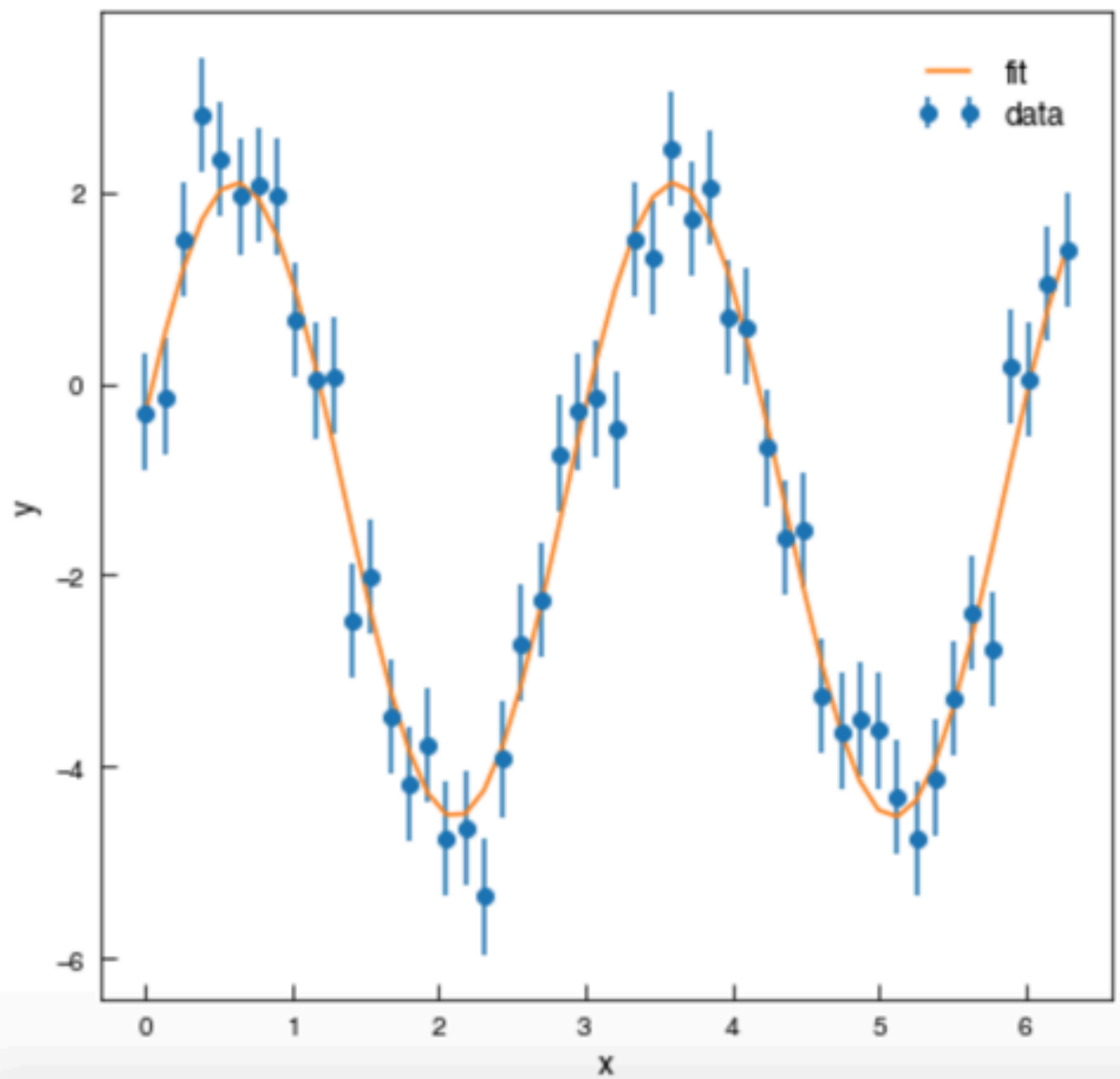
```
3.31470667373 2.10036419339 0.278528774808 -1.21522166095
```

Line Fitting

Plot the fit

In [48]:
```
f = plt.figure(figsize=(7,7))
plt.errorbar(x,y,yerr=y_err,fmt='o',label='data')
plt.plot(x,y_fit,label='fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc=0,frameon=False)
```

Out[48]:
```
<matplotlib.legend.Legend at 0x11346b198>
```



Save Your Work

Make a GitHub project “astr-119-session-6”, and commit the programs my_first_jupyter_notebook.ipynb and test_matplotlib.ipynb you made today.