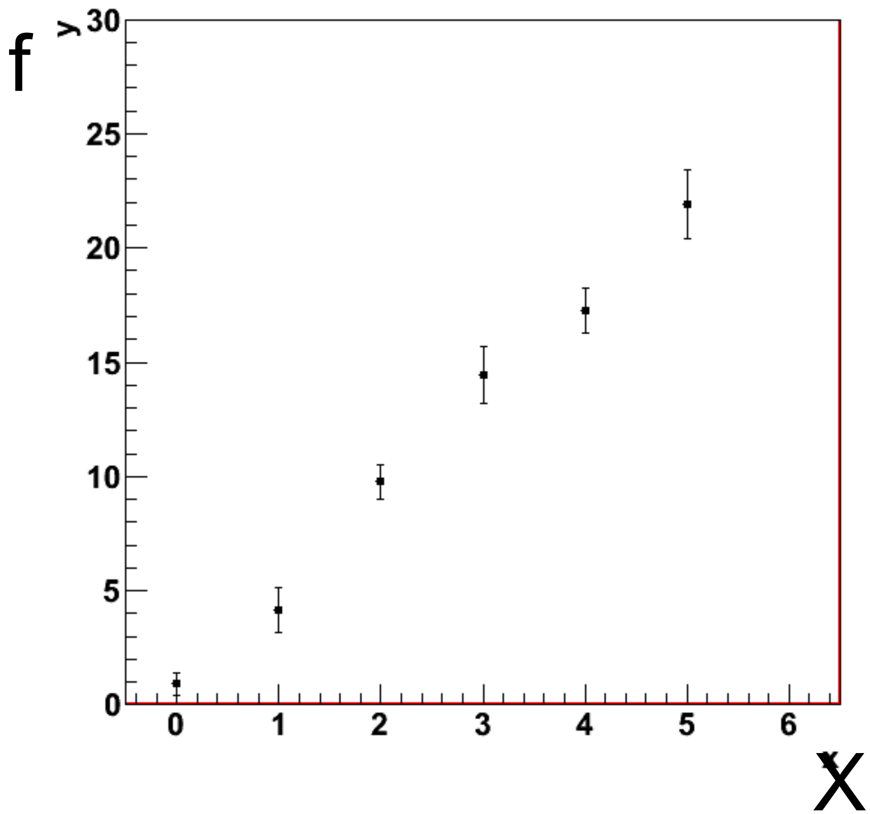
Lecture 15

Straight line fitting (con’ted)
python: numpy

Textbook: Chapter 8 (pages 181-192)

Example: Find the best straight line g(x)=ax+b through the following measured points:

x	0	1	2	3	4	5
f	0.92	4.15	9.78	14.46	17.26	21.9
σ	0.5	1.0	0.75	1.25	1.0	1.5



Write a python code to find the best fit line parametrs and their uncertainties.

Being able to do this is needed in many labs!

Result:

a=4.227 b=0.879

$(\sigma_a)^2$=0.044 $(\sigma_b)^2$=0.203

Rounding:

- 2 significant digits

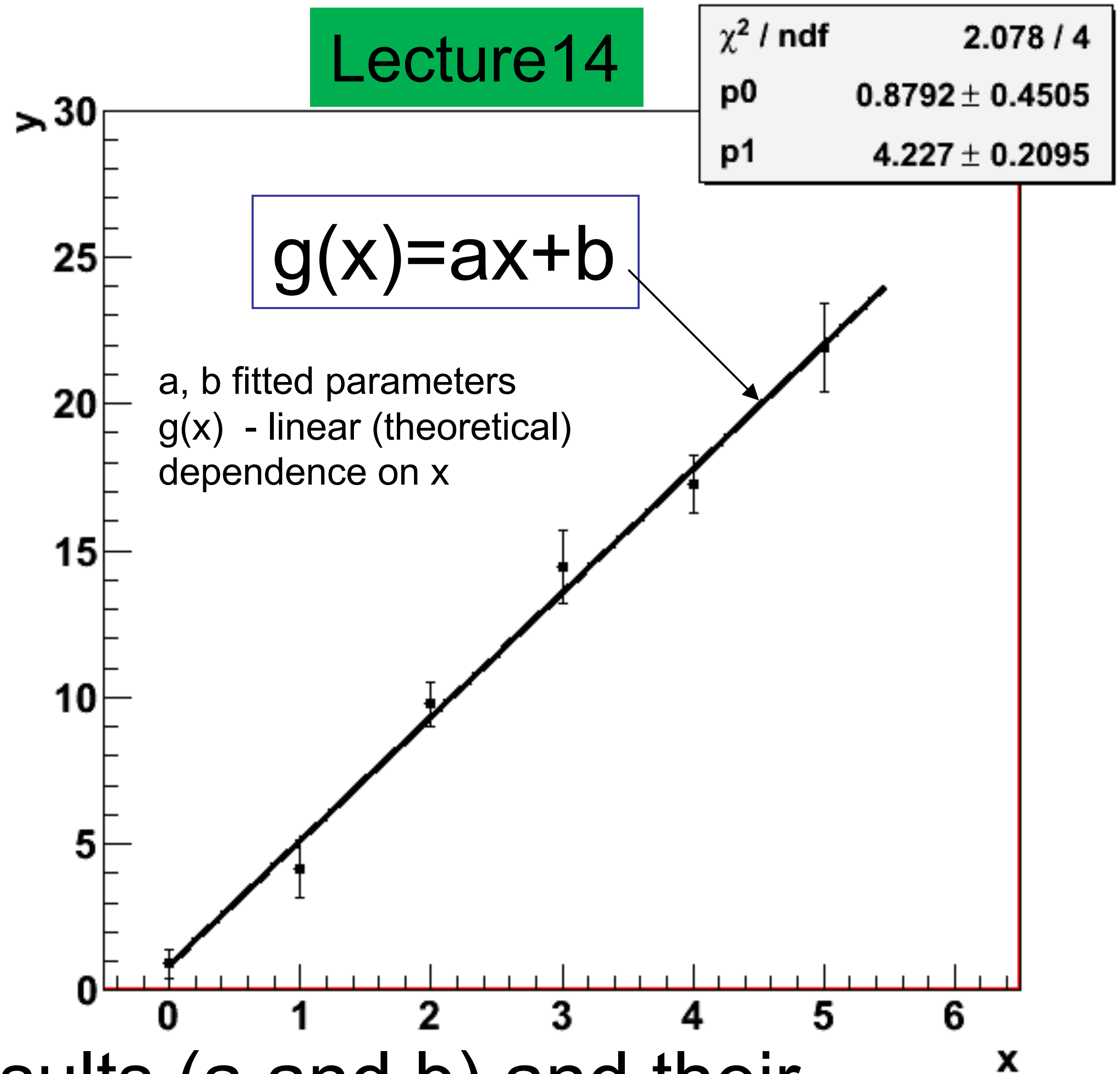
$$b = 0.88 \pm 0.45$$

$$a = 4.23 \pm 0.21$$

- 1 significant digit

$$b = 0.9 \pm 0.5$$

$$a = 4.2 \pm 0.2$$



Always round the fit results (a and b) and their uncertainties to the same significant digit

Can you reproduce those numbers using the right formulas?

S_m value for best fit parameters and b is 2.078 (check it!)

Quality of the "fit"

- For a good "fit", S_m/ndf should be close to 1 (for a large values of ndf)

where ndf (number of degree of freedom) = k= N-m,
where m=number of parameters, n= number of data points

e.g. for m=2 (line fit, 2 parameters), ndf=k=N-2,
and S_m should be ~ N-2

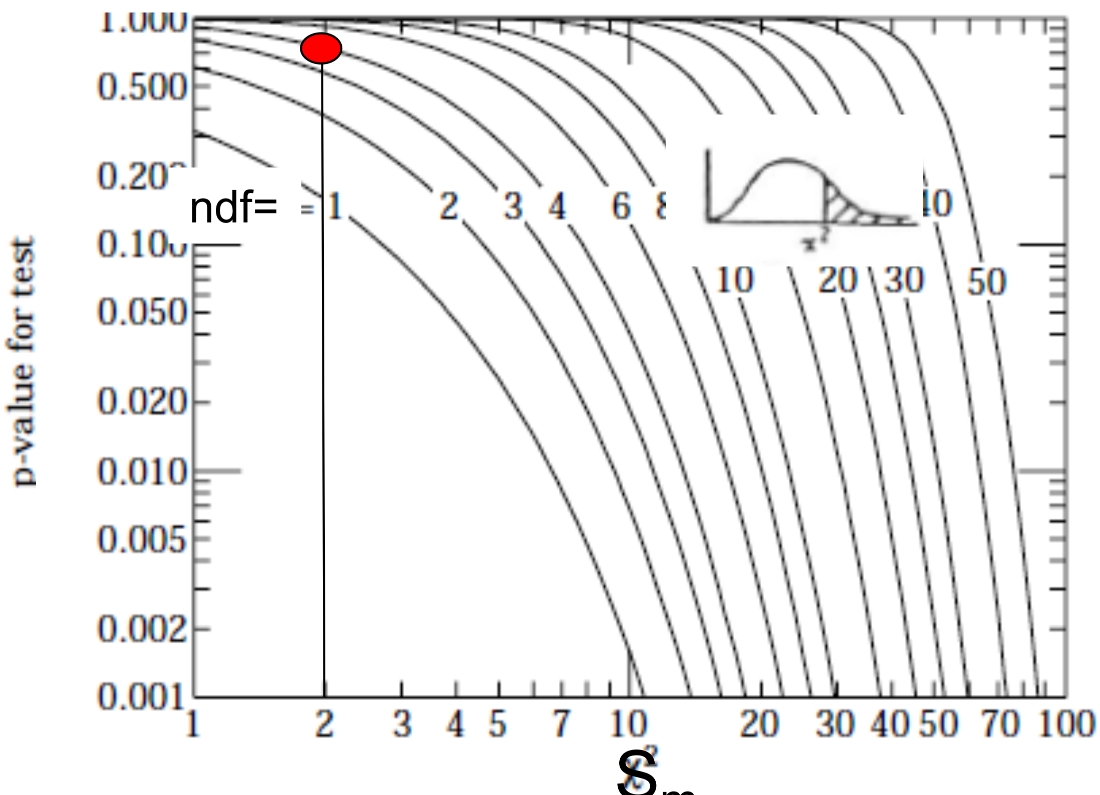
$$S_m = \sum \frac{\left(f_i - ax_i - b\right)^2}{\sigma_{i,f}^2}$$

Best fit parameters a and b (output of your program to calculate a and b) are used to calculate S_m (aka Chi2)

χ^2 Test: p-Value

Example L14:

S_m = 2.1
Ndf= 6 (nr of data points) – 2 (number of fit parameters i.e. a and b)
Large p-value (good fit)



Here you can check what is the probability for obtaining S_m for a given number of degree of freedom ndf

What does p mean?

If our experiment is repeated many times, and assuming that our hypothesis is correct, then because of fluctuations we expect a large value of of S_m (estimate of χ^2) then the particular one we are considering in a fraction p experiments.

Linear Fits. The straight line.

$\boxed{g(x)=ax+b}$

Lecture14: programmed functions to get A, B, …, F

$$a = \frac{EB - CA}{DB - A^2} \qquad \sigma_a^2 = \frac{B}{DB - A^2}$$

$$b = \frac{DC - EA}{DB - A^2} \qquad \sigma_b^2 = \frac{D}{DB - A^2}$$

Numbers A through F are determined from the measured N data points (x_i, f_i +/- $\sigma_{i,f}$), where i=1, …, N :

$$A \equiv \sum \frac{x_i}{\sigma_{i,f}^2} \qquad C \equiv \sum \frac{f_i}{\sigma_{i,f}^2} \qquad D \equiv \sum \frac{x_i^2}{\sigma_{i,f}^2}$$

$$B \equiv \sum \frac{1}{\sigma_{i,f}^2} \qquad E \equiv \sum \frac{x_i f_i}{\sigma_{i,f}^2} \qquad F \equiv \sum \frac{f_i^2}{\sigma_{i,f}^2}$$

Calculation of 2-parameter straight line fit coefficients in a code

<u>Example code</u> (Amanda Macinnis, this class former student)

import numpy as np

```
# note: the arguments f, sigma, and x need to be numpy arrays, not lists
def A(sigma, x):
    return np.sum(x / sigma**2)
def B(sigma):
    return np.sum(1.0 / sigma**2)
def C(f, sigma):
    return np.sum(f / sigma**2)
def D(sigma, x):
    return np.sum((x/sigma)**2)
def E(f, sigma, x):
    return np.sum(x*f / sigma**2)
def denominator(sigma, x):
    return D(sigma, x)*B(sigma) - A(sigma, x)**2
def a(f, sigma, x):
    return (E(f, sigma, x)*B(sigma) - C(f, sigma)*A(sigma, x))/denominator(sigma, x)
def sigma_a(f, sigma, x):
    return np.sqrt(B(sigma)/denominator(sigma, x))
def b(f, sigma, x):
    return (D(sigma, x)*C(f, sigma) - E(f, sigma, x)*A(sigma, x))/denominator(sigma, x)
def sigma_b(f, sigma, x):
    return np.sqrt(D(sigma, x)/denominator(sigma, x))
```

```
xdata=np.array([0,1,2,3,4,5])
ydata=np.array([0.92, 4.15, 9.78, 14.46, 17.26, 21.9])
yerror=np.array([0.5, 1.0, 0.75, 1.25, 1.0,1.5])

a1 = a(ydata, yerror, xdata)
sigma_a1 = sigma_a(ydata, yerror, xdata)
b1 = b(ydata, yerror, xdata)
sigma_b1 = sigma_b(ydata, yerror, xdata)

print(a1, sigma_a1)
print(b1, sigma_b1)
```

Lecture 14 example

L15 Example1: Length versus Mass for a Spring balance (textbook, ch 8, page 184)

- Data

m	2	4	6	8	10
l	42.0	48.4	51.3	56.3	58.6
σ	0.1	0.1	0.1	0.1	0.1

- Functional form

$$l = A + Bm$$

- Find:
  - A and B (fit parameters and uncertainties)
  - S_m , ndf , S_m / ndf and p-value.
  - Give interpretation of a p-value
- Calculate k and its uncertainty (use error propagation), knowing:

A=unloaded length of the spring
B=k/g (k=spring constant)

L15Example2: Measurement of absolute zero (textbook, ch 8, sec. 8.5, page 190)

- Data

P [mm Hg]	65	75	85	95	105
T[^{o}C]	-20	17	42	94	127
σ	1	1	1	1	1

- Functional form

  $T = A + BP$

- Find:
  - A and B (fit parameters and uncertainties)
  - S_m , ndf , S_m / ndf and p-value.
  - Give interpretation of a p-value
- Compare the known value for the absolute zero temperature (-273.15 ^{o}C) with the fitted parameter. How good is the agreement (or disagreement) Quantify your answer by giving the corresponding p-value and its interpretation

http://matplotlib.org/users/pyplot_tutorial.html#pyplot-tutorial

matplotlib

Pyplot tutorial

`matplotlib.pyplot` is a collection of command style functions that make matplotlib work like MATLAB. Each `pyplot` function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In `matplotlib.pyplot` various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the *axes* part of a figure and not the strict mathematical term for more than one axis).

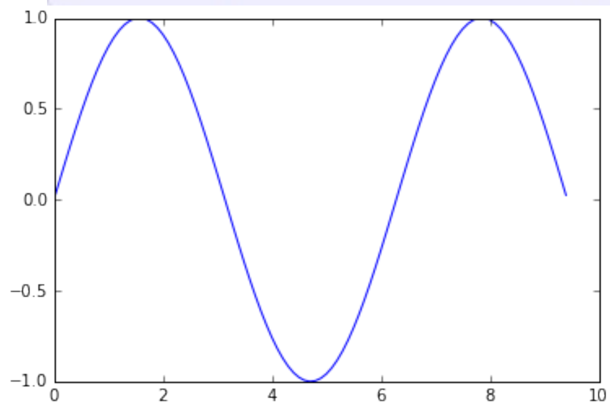
```
import matplotlib.pyplot as plt
```

Matplotlib.pyplot (very brief here)

Matplotlib - a plotting library.
`matplotlib.pyplot` - module, which provides a plotting system
The most important function in matplotlib is plot

Example:

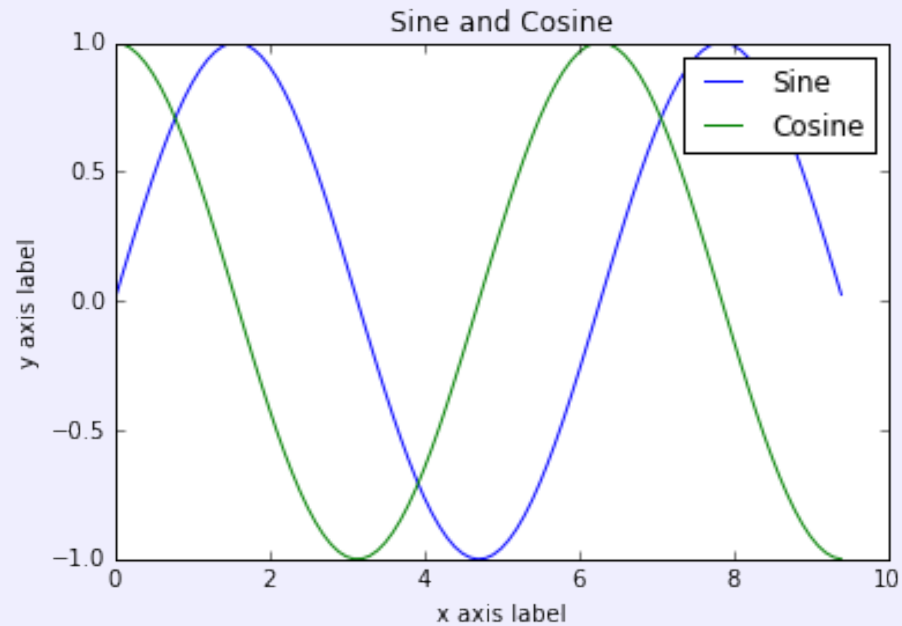
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```



Exercise:
a) re-do the above example for step=1, 0.5, 0.2 and 0.01 values
b) re-do the above example by using linspace function instead of arange

Example:

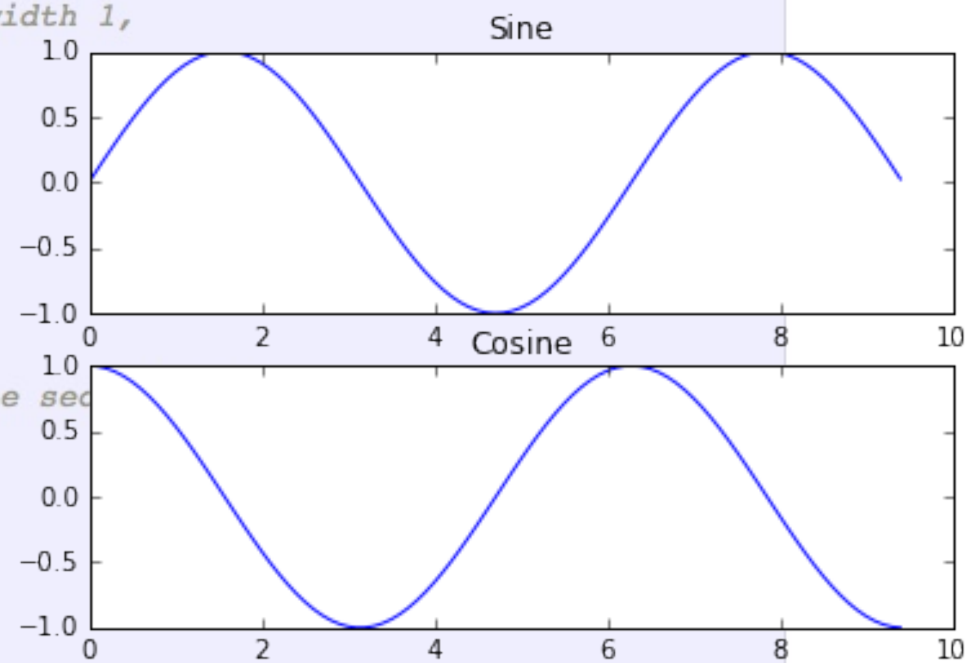
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



Exercise: plot a straight line
y=ax+b for x from 0 to 20, a=1, b=3

Example:

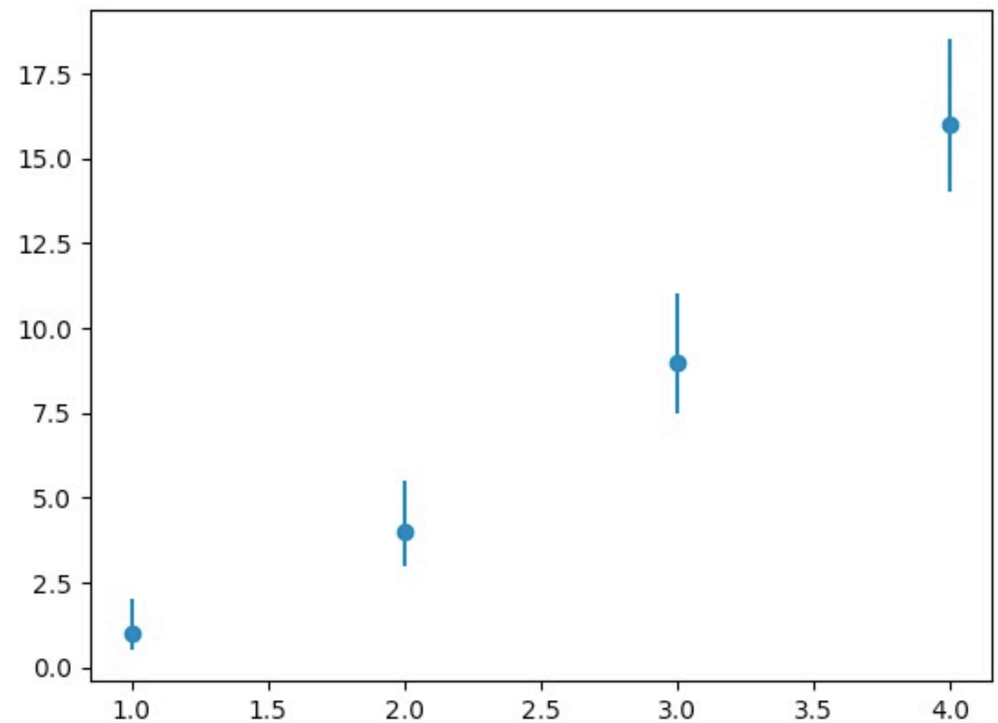
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the sec
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```




```
fig_file='fig1.png'
plt.savefig(fig_file)
plt.clf()
plt.close()
```

Plotting data points with errors

```
import matplotlib.pyplot as plt
x= np.array([1,2,3,4])
y=np.array([1,4,9,16])
ey=np.array([0.5,1.,1.5,2.])
le=np.array([0.5,1.,1.5,2.])
ue=np.array([1.0,1.5,2.0,2.5])

#plt.errorbar(x,y,ey,fmt='o')
plt.errorbar(x,y,yerr=[le,ue],fmt='o')

plt.show()
```



https://pythonhealthcare.org/2018/04/13/51-matplotlib-adding-error-bars-to-charts/

Exercise: make plots (data points with errors and resulting straight lines) for L15Example1 and L15Example2