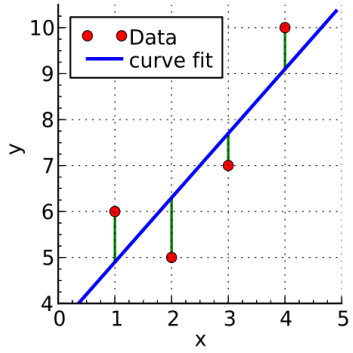
I. GETTING INFORMATION ABOUT VECTORS

There are a bunch of routines built into Python which give information about vectors as shown in the IDLE conversation below. The conversation also shows how to generate random numbers using either a uniform distribution **np.random.uniform(low value, high value ,number of points)** or the random distribution about the mean **np.random.normal(central value, spread ,number of points)**

```
>>> import numpy as np
>>> x=np.linspace(0,5,6)
>>> print x
[ 0.  1.  2.  3.  4.  5.]
>>> mean=np.mean(x)
>>> print mean
2.5
>>> standard_deviation=np.std(x)
>>> print standard_deviation
1.70782512766
>>> sum_of_elements=np.sum(x)
>>> print sum_of_elements
15.0
>>> r=np.random.uniform(-1,1,5)
>>> print r
[-0.03559537  0.06025269  0.03055397  0.72838517  0.59171467]
>>> r=np.random.uniform(-1,1,5)
>>> print r
[ 0.29430366  0.74615866  0.96678452 -0.27776108 -0.08117502]
>>> r=np.random.normal(0,1,100)
>>> print np.mean(r)
-0.0230497638522
>>> print np.sum(r)
-2.30497638522
>>> print np.std(r)
1.03412792852
>>> r=np.random.normal(10,1,100)
>>> print np.mean(r)
10.082592169
>>> print np.std(r)
1.02812176346
>>>
```

II. LEAST SQUARES FITTING TO A STRAIGHT LINE

A computer can find the best-fit parameters of a data set of N points (x_i, y_i) in which y is assumed to depend linearly on x using the method of least squares. How does it do this? Consider the figure below (source: Wikipedia). There is a discrepancy (green line) between the measured data (red dots) and the model (blue curve):



The best-fit parameters (m, b) will minimize sum the of the squares of the difference between the measured value y_i and the linear model value $y_i^{th} = mx_i + b$:

$$S = \sum_i^N [y_i - (mx_i + b)]^2$$

$S(m, b)$, a function of two variables is minimized when both partial derivative are zero:

$$\frac{\partial S}{\partial m} = \sum_i^N 2[y_i - (mx_i + b)](-x_i) = 0$$

$$\frac{\partial S}{\partial b} = \sum_i^N 2[y_i - (mx_i + b)](-1) = 0$$

which can be rewritten as:

$$m \sum_i^N x_i x_i + b \sum_i^N x_i = \sum_i^N x_i y_i$$

$$m \sum_i^N x_i + b \sum_i^N 1 = \sum_i^N y_i$$

Writing the sums as $\sum_i^N x_i y_i = [XY], \sum_i^N x_i = [X]$ etc. and where N is the number of data points we have two linear equations:

$$m[XX] + b[X] = [XY]$$

$$m[X] + bN = [Y]$$

which are easily solved for the best fit parameters m and b

III. EXERCISES

1. Write a program that will perform the following tasks

   (a) Generate a data simulated dataset of one hundred (x_i, y_i) pairs where $-10 \leq x(i) \leq 10$ and $y(i) = 4 * x(i) + 2 + r(i)$ where $r(i)$ is a random number between approximately -5.0 and +5.0. Search the **numpy** reference guide (scipy.org) for guidance in generating random numbers. Hint: **np.random.uniform() ; np.random.normal()**

   (b) Write a routine that uses that will determine the slope and y-intercept of the line of best fit using the method of least squares. (Hint: You do not need for loop, **np.sum()** is a kind of built-in for loop). You know that the answers have to be $m \approx 4$ and $b \approx 2$

   (c) Plot the y vs x data as well a best fit line to the data. Label the axes and give the graph a title.

   (d) Calculate S as a function of m (i) S using best fit b and the best fit m as well as $0.5m, 0.6m, ..., 1.4m, 1.5m$.

   (e) Plot S as a function of m with axes labels and title and save the graph to disk.