Lecture 4: Data

- Data files and types (munging data)

- Least squares and χ^2 fitting

Data

Tabbed or simply delimited data can be read(or written) easily with np.loadtxt (or np.savetxt)

Multiple data sets from different types of experiments or simulations

Metadata: data about your data.

    simulation ~ compiler flags or libraries, input parameters, dates, etc.

    experiment ~ background readings of other equipment, instrumental settings, who took data, etc.

Pandas Library can read a large assortment of datafile types

Lots of different formats for data:

    structured: .csv, excel (.xlsx), .sql

    unstructured (markup for metadata): .xml or .json

    (pseudo)file-system : HDF5, zip (with metadata in .xml or .json)

?Stata, Clipboard, Pickle, Feather, SAS, …?

Structured vs. unstructured

Structured: each entry has the same fields (csv, sql)

entry from SQL database, country demographics

```
id|code|name|area|area_land|area_water|population|population_growth|birth_rate|death_rate|migration_rate|created_a
t|updated_at1|af|Afghanistan|652230|652230|0|32564342|2.32|38.57|13.89|1.51|2015-11-01 13:19:49.461734|2015-11-01
13:19:49.4617342|al|Albania|28748|27398|1350|3029278|0.3|12.92|6.58|3.3|2015-11-01 13:19:54.431082|2015-11-01
13:19:54.4310823|ag|Algeria|2381741|2381741|0|39542166|1.84|23.67|4.31|0.92|2015-11-01 13:19:59.961286|2015-11-01
13:19:59.961286
```

.xml file (extensible markup language)

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>

  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>

  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>

  <book category="web">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
```

.json (javascript object notation) file, events from visitors to a websi

```
{'event_type': 'started-mission', 'keen': {'created_at': '2015-06-12T23:09:03.966Z
'557b668fd2eaaa2e7c5e916b', 'timestamp': '2015-06-12T23:09:07.971Z'}, 'sequence':
{'event_type': 'started-screen', 'keen': {'created_at': '2015-06-12T23:09:03.979Z'
'557b668f90e4bd26c10b6ed6', 'timestamp': '2015-06-12T23:09:07.987Z'}, 'mission': 1
'sequence': 4, 'type': 'code'} {'event_type': 'started-screen', 'keen': {'created_
06-12T23:09:22.517Z', 'id': '557b66a246f9a7239038b1e0', 'timestamp': '2015-06-
12T23:09:24.246Z'}, 'mission': 1, 'sequence': 3, 'type': 'code'},
```

Pandas

Import data as a “DataFrame”

- Collection of series (similar to ndarray, but not fixed length)
  - Can be created from dictionary of arrays, lists, or series
- Gracefully handles missing or corrupt data
- Many numpy operations also work on dataframes (slicing, whole set manipulation/operations, Boolean logic, etc.)

```
import pandas as pd
data=pd.read_excel("data.xlsx", sheet_name="day1")
df=pd.DataFrame(data,columns=['time', 'x'])
```

Wrangling

```python
data=pd.read_excel("/Users/damien/Downloads/data.xlsx", sheet_name="day1")
df=pd.DataFrame(data,columns=['time','x'])
```

```python
print(df)
```

```
    time         x
0    0.0 -0.040263
1    1.0 -0.159246
2    2.0 -0.112235
3    3.0  0.451919
4    4.0  0.464793
5    5.0  0.472553
6    6.0  0.471418
7    7.0  0.604470
8    8.0  0.781627
9    9.0  0.977531
10  10.0  0.991702
11  11.0  1.117166
12  12.0  1.052343
13  13.0  1.154983
14  14.0  1.923066
15  15.0  1.225923
16  16.0  1.406605
17  17.0  1.756546
18  18.0  1.791405
19  19.0  1.824762
20   NaN       NaN
21   0.0 -0.143025
22   1.0 -0.172238
23   2.0 -0.024825
24   3.0  0.618444
```

Boolean arrays:

```python
dat=np.arange(10)
print(dat)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```python
dat<5
```

```
array([ True,  True,  True,  True,  True, False, False, False, False,
       False])
```

```python
print(dat[dat<5])
```

```
[0 1 2 3 4]
```

a Boolean array as an argument to an ndarray (or df) selects elements which are true

```python
df['x'].notnull()
```

```
0      True
1      True
2      True
3      True
4      True
5      True
6      True
7      True
8      True
9      True
10     True
11     True
12     True
13     True
14     True
15     True
16     True
17     True
18     True
19     True
20    False
21     True
```

```python
print(df[df['x'].notnull()])
```

```
    time         x
0    0.0 -0.040263
1    1.0 -0.159246
2    2.0 -0.112235
3    3.0  0.451919
4    4.0  0.464793
5    5.0  0.472553
6    6.0  0.471418
7    7.0  0.604470
8    8.0  0.781627
9    9.0  0.977531
10  10.0  0.991702
11  11.0  1.117166
12  12.0  1.052343
13  13.0  1.154983
14  14.0  1.923066
15  15.0  1.225923
16  16.0  1.406605
17  17.0  1.756546
18  18.0  1.791405
19  19.0  1.824762
21   0.0 -0.143025
22   1.0 -0.172238
```

```python
print(df[df['x']<1])
```

```
    time         x
0    0.0 -0.040263
1    1.0 -0.159246
2    2.0 -0.112235
3    3.0  0.451919
4    4.0  0.464793
5    5.0  0.472553
6    6.0  0.471418
7    7.0  0.604470
8    8.0  0.781627
9    9.0  0.977531
10  10.0  0.991702
21   0.0 -0.143025
22   1.0 -0.172238
23   2.0 -0.024825
24   3.0  0.618444
25   4.0  0.584245
26   5.0  0.517715
```

Ingredients of a data fitting problem

- A set of data $\{x_i, y_i\}$

  Usually, x_i are measured with sufficient certainty, while y_i are subject to some uncertainty (i.e., error bars).

- A model function $f(x; \{a_i\})$

  The function depends on a set of adjustable parameters $\{a_i\}$
  For linear fitting, the function is simply $f(x; \{a_1, a_2\}) = a_1 + a_2 x$
  Or in a more common form $f(x) = a + bx$

- A figure-of-merit to be optimized

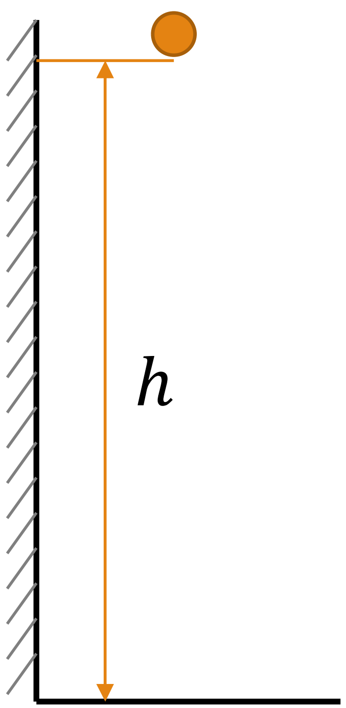
  Least squares: $$S = \sum_{i=1}^{N} [y_i - f(x_i)]^2$$

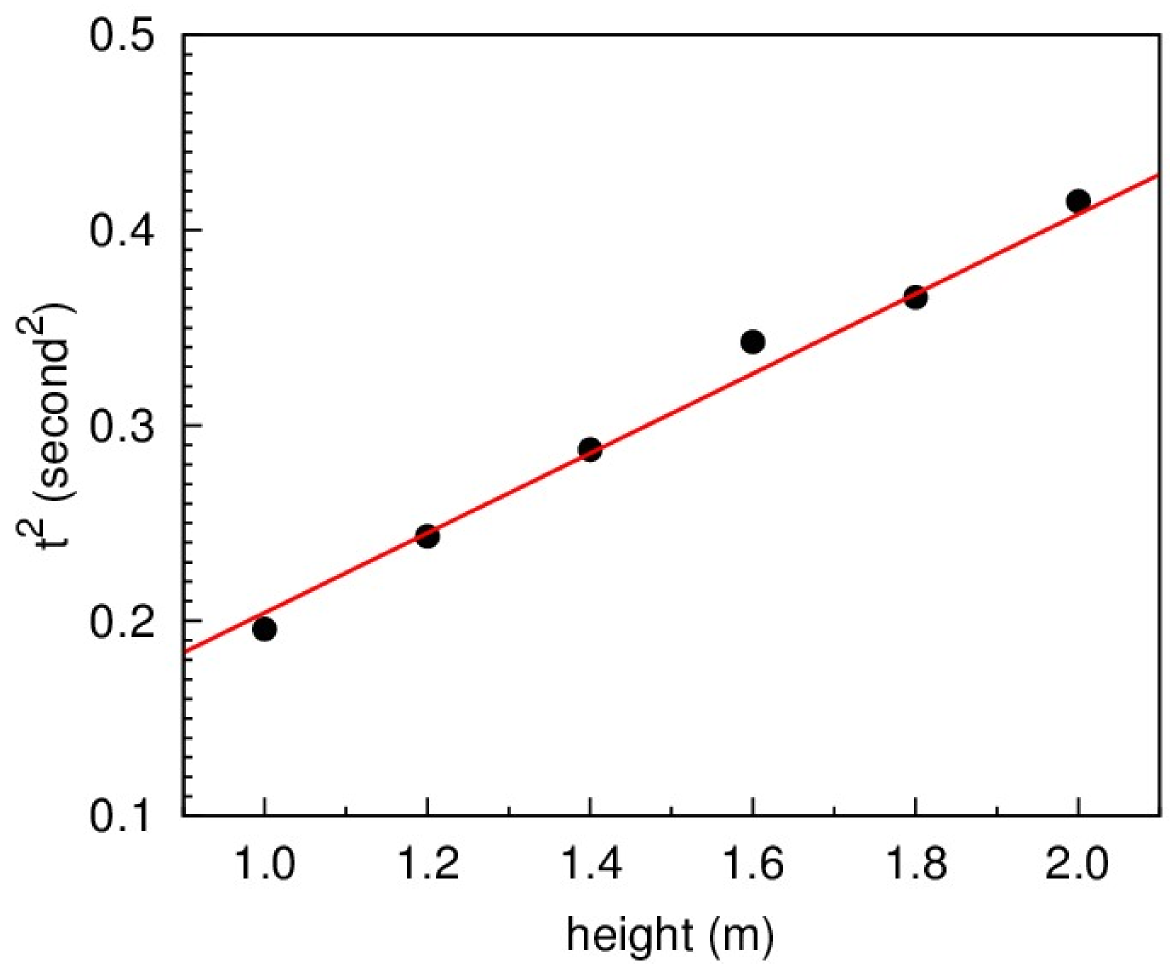
  Chi-square: $$\chi^2 = \sum_{i}^{N} \left[\frac{y_i - f(x_i)}{\sigma_i}\right]^2$$

Least squares are special case of chi-square with all σ_i equal. On the other hand, chi-square can be considered as weighted least squares.

Measuring the acceleration of gravity, ***g***, using free fall

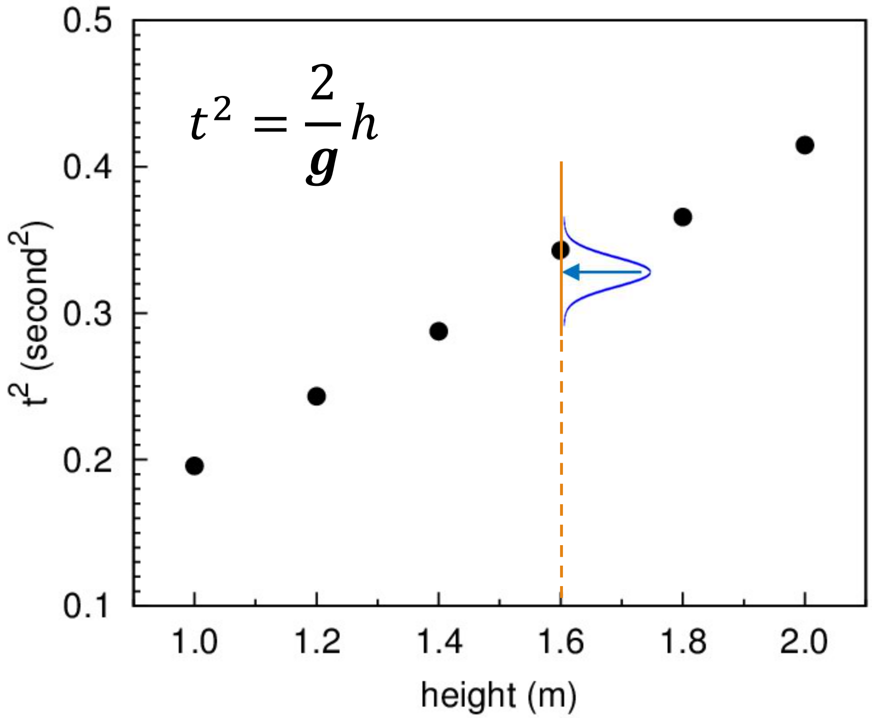
$$h = \frac{1}{2}\boldsymbol{g}t^2$$





Why Least squares and chi-square?

$x_i\ (h_i)$	1.0	1.2	1.4	1.6	1.8	2.0
$y_i\ (t_i^2)$	0.20	0.24	0.29	0.34	0.37	0.41



Why do the measured points, y_i, not fall on exactly a straight line?

Because any experimental measurement has an **error bar**.

$$P(y_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{[y_i - f(x_i)]^2}{2\sigma^2}\right)$$

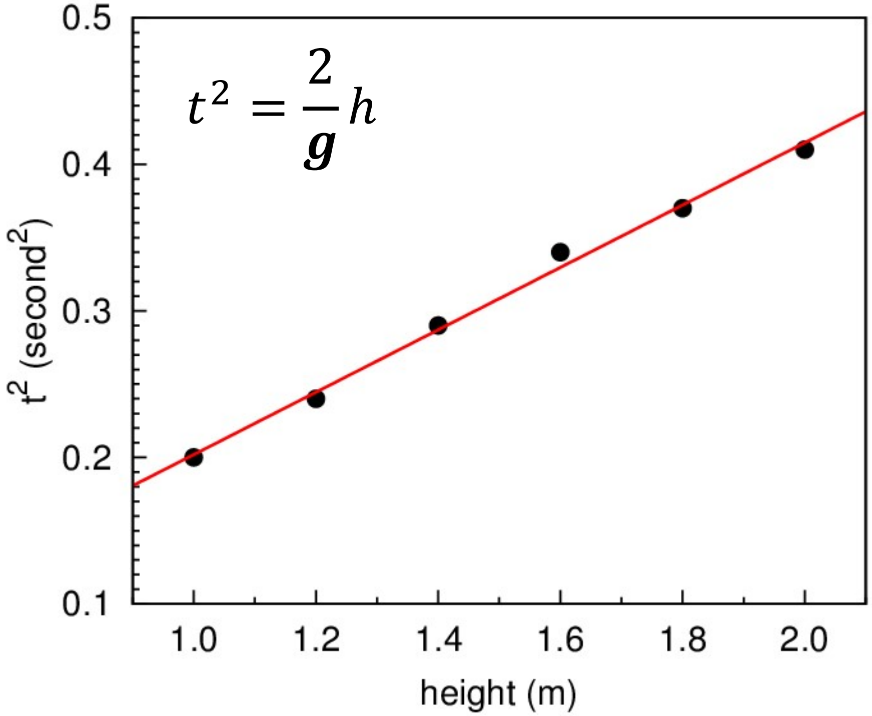
Probability of finding y_i in the vicinity of its actual (or expected) value with $\boldsymbol{\sigma}$ being the standard deviation (or error bar).

- The goal of fitting is to find the set of $\{a_i\}$ that maximizes the probability for ALL measured points, i.e., $\prod_i P(y_i)$.
- This is equivalent to the minimization of least squares or chi-square

$$S = \sum_{i=1}^{N} [y_i - f(x_i)]^2 \qquad \chi^2 = \sum_{i}^{N} \left[\frac{y_i - f(x_i)}{\sigma_i}\right]^2$$

Linear fitting (or regression)

x_i (h_i)	1.0	1.2	1.4	1.6	1.8	2.0
y_i (t_i^2)	0.20	0.24	0.29	0.34	0.37	0.41



$$y = f(x) = ax + b$$

$$S = \sum_{i=1}^{N} [y_i - f(x_i)]^2$$

$$\begin{aligned} S = {} & [0.20 - (a * 1.0 + b)]^2 \\ & +[0.24 - (a * 1.2 + b)]^2 \\ & +[0.29 - (a * 1.4 + b)]^2 \\ & +[0.34 - (a * 1.6 + b)]^2 \\ & +[0.37 - (a * 1.8 + b)]^2 \\ & +[0.41 - (a * 2.0 + b)]^2 \end{aligned}$$

$$S = 0.6023 - 5.848a + 14.2a^2 - 3.7b + 18ab + 6b^2$$

$$\left.\begin{aligned} \frac{\partial S}{\partial a} &= -5.848 + 28.4a + 18b = 0 \\ \frac{\partial S}{\partial b} &= -3.7 + 18a + 12b = 0 \end{aligned}\right\} \quad \begin{aligned} a &= 0.213 \\ b &= -0.011 \end{aligned}$$

Linear fitting – General formulation

$$\chi^2(a,b) = \sum_{i=1}^{N}\left(\frac{y_i - ax_i - b}{\sigma_i}\right)^2 = \sum_{i=1}^{N}\frac{y_i^2}{\sigma_i^2} - \frac{2x_iy_i}{\sigma_i^2}a - \frac{2y_i}{\sigma_i^2}b + \frac{x_i^2}{\sigma_i^2}a^2 + \frac{2x_i}{\sigma_i^2}ab + \frac{1}{\sigma_i^2}b^2$$

$$\frac{\partial\chi^2}{\partial a} = \sum_{i=1}^{N} -\frac{2x_iy_i}{\sigma_i^2} + \frac{2x_i^2}{\sigma_i^2}a + \frac{2x_i}{\sigma_i^2}b = 0 \qquad \overbrace{\sum_{i=1}^{N}\frac{x_i^2}{\sigma_i^2}}^{C_{xx}}a + \overbrace{\sum_{i=1}^{N}\frac{x_i}{\sigma_i^2}}^{C_x}b = \overbrace{\sum_{i=1}^{N}\frac{x_iy_i}{\sigma_i^2}}^{C_{xy}} \qquad C_{xx}a + C_xb = C_{xy}$$

$$\frac{\partial\chi^2}{\partial b} = \sum_{i=1}^{N} -\frac{2y_i}{\sigma_i^2} + \frac{2x_i}{\sigma_i^2}a + \frac{2}{\sigma_i^2}b = 0 \qquad \underbrace{\sum_{i=1}^{N}\frac{x_i}{\sigma_i^2}}_{C_x}a + \underbrace{\sum_{i=1}^{N}\frac{1}{\sigma_i^2}}_{C}b = \underbrace{\sum_{i=1}^{N}\frac{y_i}{\sigma_i^2}}_{C_y} \qquad C_xa + Cb = C_y$$

$$\boxed{a = \frac{CC_{xy} - C_xC_y}{CC_{xx} - C_x^2}} \qquad \boxed{b = \frac{C_{xx}C_y - C_xC_{xy}}{CC_{xx} - C_x^2}}$$

Can we estimate the error bars in a and b given the error bars of y_i (i.e, σ_i)?

Errors (or uncertainties) in *a* and *b*

variance is the square of standard deviation

For a function with independent variables $f(x_1, x_2, \ldots x_N)$, the variance in f, σ_f^2, is given by the variance of the variables, $\sigma_{x_i}^2$, through the error propagation formula:

$$a(y_i) = \frac{CC_{xy} - C_x C_y}{CC_{xx} - C_x^2}$$

$$b(y_i) = \frac{C_{xx} C_y - C_x C_{xy}}{CC_{xx} - C_x^2}$$

$$\boxed{\sigma_f^2 = \sum_{i=1}^{N} \left(\frac{\partial f}{\partial x_i}\right)^2 \sigma_{x_i}^2}$$

σ_i^2 is the short form for $\sigma_{y_i}^2$

$$\underbrace{\sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2}}_{C_{xx}} a + \underbrace{\sum_{i=1}^{N} \frac{x_i}{\sigma_i^2}}_{C_x} b = \underbrace{\sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2}}_{C_{xy}}$$

$$\underbrace{\sum_{i=1}^{N} \frac{x_i}{\sigma_i^2}}_{C_x} a + \underbrace{\sum_{i=1}^{N} \frac{1}{\sigma_i^2}}_{C} b = \underbrace{\sum_{i=1}^{N} \frac{y_i}{\sigma_i^2}}_{C_y}$$

Note: the variables here are y_i, not x_i!

$$\sigma_a^2 = \sum_{i=1}^{N} \left(\frac{\partial a}{\partial y_i}\right)^2 \sigma_i^2 = \sum_{i=1}^{N} \left(\frac{Cx_i - C_x}{CC_{xx} - C_x^2}\right)^2 \frac{1}{\sigma_i^2}$$

$$= \frac{1}{\Delta^2} \sum_{i=1}^{N} \frac{C^2 x_i^2 - 2Cx_i C_x + C_x^2}{\sigma_i^2}$$

$$= \frac{C^2 C_{xx} - 2CC_x^2 + CC_x^2}{\Delta^2} = \frac{C\Delta}{\Delta^2} = \frac{C}{\Delta}$$

$$= \frac{C}{CC_{xx} - C_x^2}$$

$$\sigma_b^2 = \sum_{i=1}^{N} \left(\frac{\partial b}{\partial y_i}\right)^2 \sigma_i^2 = \sum_{i=1}^{N} \left(\frac{C_{xx} - C_x x_i}{CC_{xx} - C_x^2}\right)^2 \frac{1}{\sigma_i^2}$$

$$= \frac{1}{\Delta^2} \sum_{i=1}^{N} \frac{C_{xx}^2 - 2C_{xx} C_x x_i + C_x^2 x_i^2}{\sigma_i^2}$$

$$= \frac{CC_{xx}^2 - 2C_{xx} C_x^2 + C_x^2 C_{xx}}{\Delta^2} = \frac{C_{xx}\Delta}{\Delta^2} = \frac{C_{xx}}{\Delta}$$

$$= \frac{C_{xx}}{CC_{xx} - C_x^2}$$

Summary of Linear Regression

Fitting function: $y = f(x) = ax + b$

$$\chi^2(a,b) = \sum_{i=1}^{N}\left(\frac{y_i - ax_i - b}{\sigma_i}\right)^2$$

Solve for a and b by minimizing χ^2:

$$\frac{\partial \chi^2}{\partial a} = 0 \quad \text{and} \quad \frac{\partial \chi^2}{\partial b} = 0$$

Solution for a and b:

$$a = \frac{CC_{xy} - C_x C_y}{CC_{xx} - C_x^2} \qquad b = \frac{C_{xx}C_y - C_x C_{xy}}{CC_{xy} - C_x^2}$$

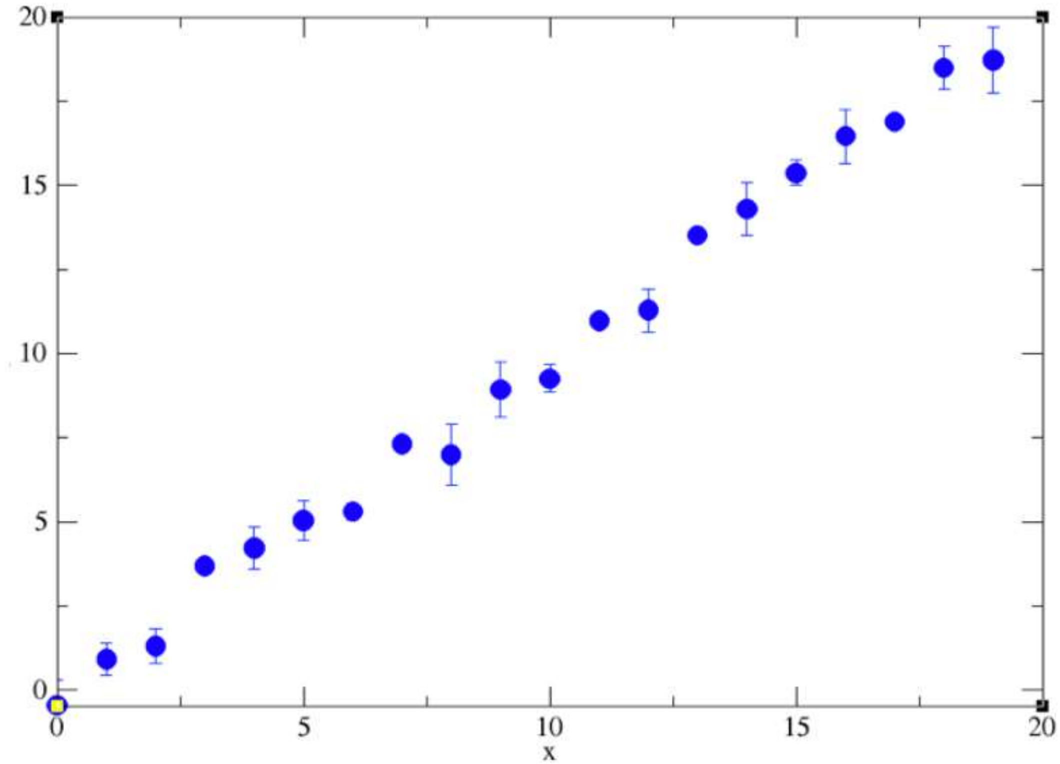
Error bars on for a and b:

$$\sigma_a^2 = \frac{C}{CC_{xx} - C_x^2} \qquad \sigma_b^2 = \frac{C_{xx}}{CC_{xx} - C_x^2}$$

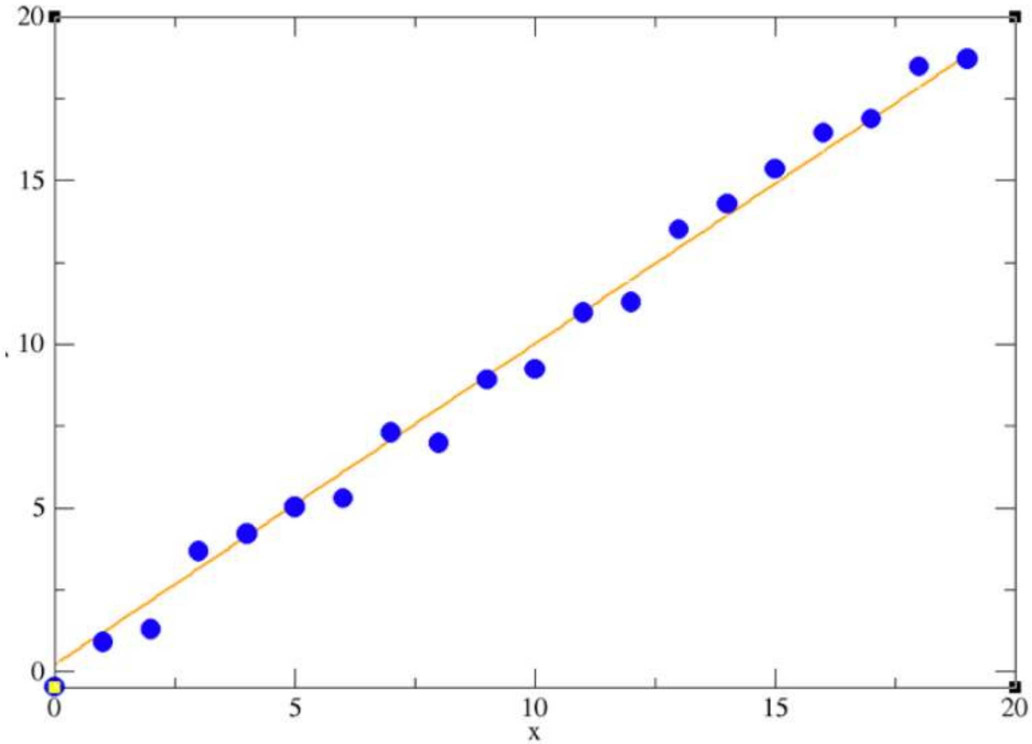
$$\overbrace{\sum_{i=1}^{N}\frac{x_i^2}{\sigma_i^2}}^{C_{xx}} a + \overbrace{\sum_{i=1}^{N}\frac{x_i}{\sigma_i^2}}^{C_x} b = \overbrace{\sum_{i=1}^{N}\frac{x_i y_i}{\sigma_i^2}}^{C_{xy}}$$

$$\underbrace{\sum_{i=1}^{N}\frac{x_i}{\sigma_i^2}}_{C_x} a + \underbrace{\sum_{i=1}^{N}\frac{1}{\sigma_i^2}}_{C} b = \underbrace{\sum_{i=1}^{N}\frac{y_i}{\sigma_i^2}}_{C_y}$$

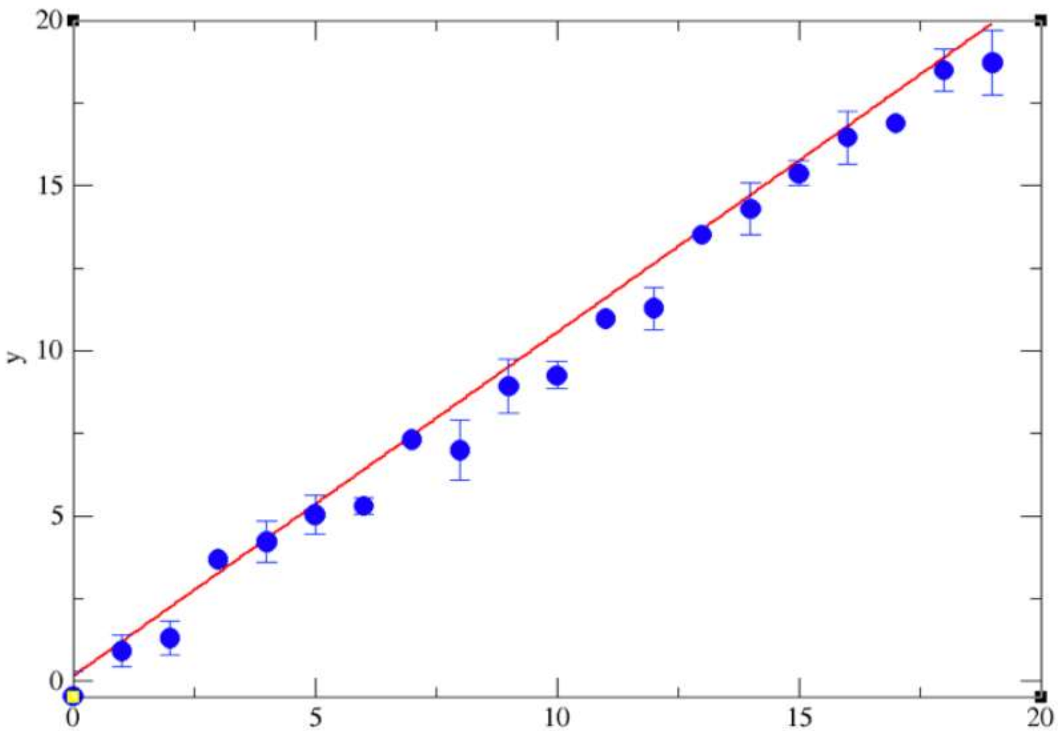
Data Set



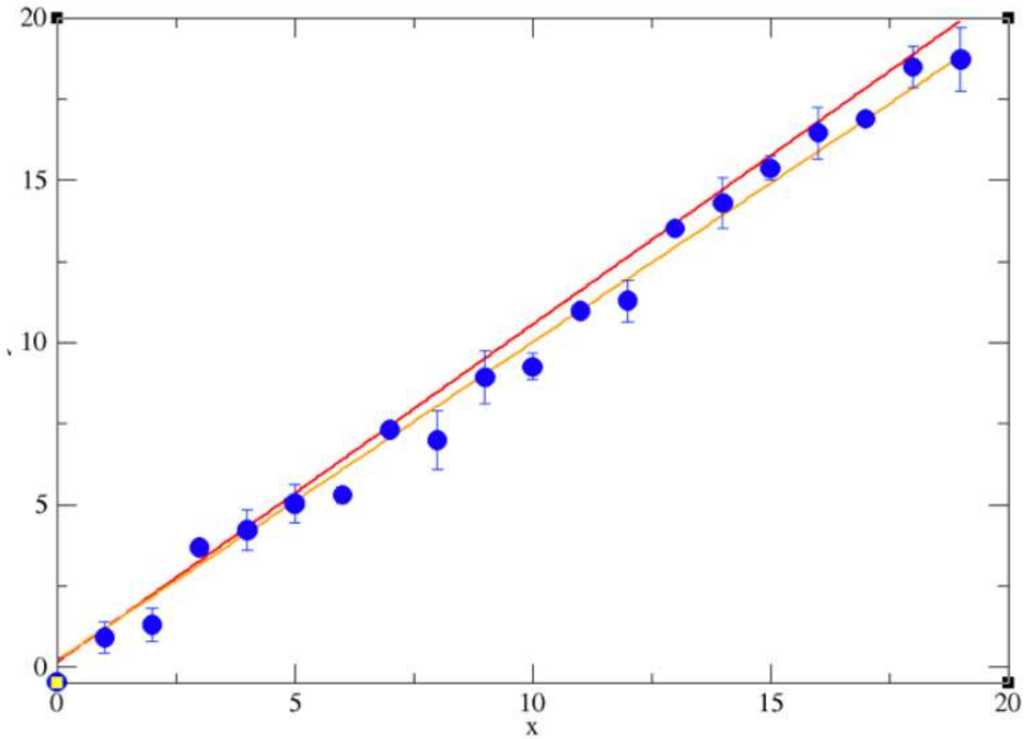
Regression



Regression with Errors



Comparison



numpy.polyfit

numpy.polyfit(*x, y, deg, rcond=None, full=False, w=None, cov=False*)

Least squares polynomial fit.

```
fit = np.polyfit(x,y,1)
fit_fn = np.poly1d(fit)
print(fit)
```

(linear fit)

```
[0.08802552 0.08670555]
```

Returns:	p : *ndarray, shape (deg + 1,) or (deg + 1, K)* Polynomial coefficients, highest power first. If *y* was 2-D, the coefficients for *k*-th data set are in `p[:,k]`. **residuals, rank, singular_values, rcond** Present only if full = True. Residuals of the least-squares fit, the effective rank of the scaled Vandermonde coefficient matrix, its singular values, and the specified value of *rcond*. For more details, see linalg.lstsq.

```
fit,res,_,_,_ = np.polyfit(x,y,1,full=True)
print("variance =", res/len(y))
```

```
variance = [0.12962381]
```

Goodness of fit, R^2 value

$R^2 = 1 - \frac{S}{S_{tot}}$,

with $S_{tot} = \Sigma_i (y_i - y_{mean})^2$

General least-squares problem

$$f(x) = \sum_{k=1}^{M} a_k X_k(x)$$

(X_k are basis functions)

(note, fitting function is still only linear in the parameters, $a_0, a_1, a_2, \ldots$)

Linear regression:

- $f(x) = a_0 + a_1 x \quad (f_0 = 1, f_1 = x)$

Higher order polynomial (still considered linear regression):

- $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots \quad (f_0 = 1, f_1 = x, f_2 = x^2, f_3 = x^3, \ldots)$

Nonlinear functions:

- $f(x) = a_0 + a_1 \sin(x) + a_2 e^x \quad (f_0 = 1, f_1 = \sin(x), f_2 = e^x,)$

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2$$

Minimize χ^2 to find $a_0, a_1, a_2, \ldots$

will be a set of linear equations in a's

χ^2 and nonlinear routines

Non-linear routines may converge to "local" minima, not the true optimum

scipy.optimize.curve_fit

scipy.optimize.curve_fit(*f, xdata, ydata, p0=None, sigma=None, absolute_sigma=False, check_finite=True, bounds=(-inf, inf), method=None, jac=None, **kwargs*) [source]

Use non-linear least squares to fit a function, f, to data.

Returns:

popt : *array*

Optimal values for the parameters so that the sum of the squared residuals of `f(xdata, *popt) - ydata` is minimized

pcov : *2d array*

The estimated covariance of popt. The diagonals provide the variance of the parameter estimate. To compute one standard deviation errors on the parameters use `perr = np.sqrt(np.diag(pcov))`.