

MAS212 Scientific Computing and Simulation

Dr. Sam Dolan

School of Mathematics and Statistics,
University of Sheffield

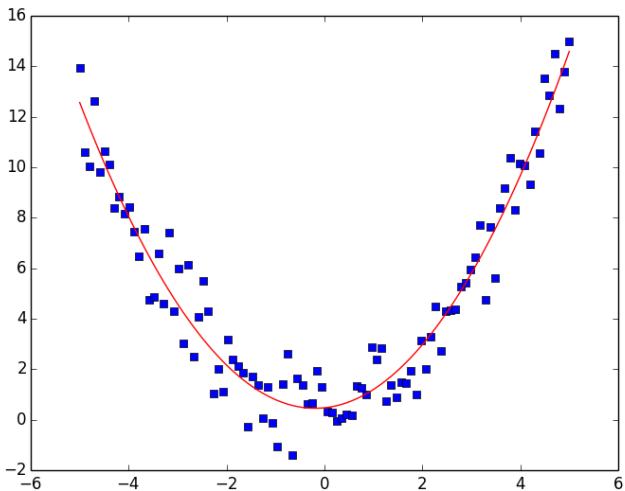
Autumn 2017

<http://sam-dolan.staff.shef.ac.uk/mas212/>

G18 Hicks Building
s.dolan@sheffield.ac.uk

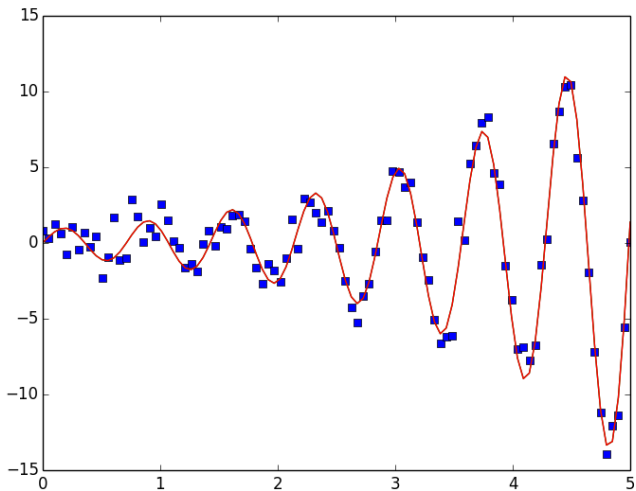
Today's lecture

- **Fitting data to models**
- Theory:
 - The line of best fit
 - The method of least squares
 - Linear models
 - Non-linear models
- Using Python:
 - `scipy.optimize.curve_fit()`



Example #1: Least-squares fit to quadratic model

$$f(x, \beta_i) = \beta_0 + \beta_1 x + \beta_2 x^2.$$



Example #2: Fit to non-linear model

$$f(x, \beta_i) = \beta_0 \exp(\beta_1 x) \sin(10\beta_2 x) + \beta_3.$$

Example: Fitting a straight line

- Suppose I have a data set (x_i, y_i) for $i = 0 \dots N - 1$
- How do I find the **line of best fit**?
- That is, how do I fit a two-parameter model to the data?

$$f(x; \beta_0, \beta_1) = \beta_0 + \beta_1 x \quad \text{where } \beta_0, \beta_1 \text{ are model parameters}$$

Example: Fitting a straight line

- The textbook formulae for **linear regression** are

$$\beta_1 = \frac{\text{covar}(x, y)}{\text{var}(x)}, \quad \beta_0 = \bar{y} - \beta_1 \bar{x},$$

$$\text{var}(x) \equiv \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2,$$

$$\text{covar}(x, y) \equiv \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y}).$$

- Here an over-bar denotes the **mean**:

$$\bar{x} \equiv \frac{1}{N} \sum_{i=1}^N x_i, \quad \bar{y} \equiv \frac{1}{N} \sum_{i=1}^N y_i$$

- Where do these formulae derive from?

Least squares method

- Suppose we have a **model** $f(x, \beta_j)$ with **parameters** β_j .
- We wish to adjust the parameters β_j of the model to achieve the best fit to a given data set (x_i, y_i) .
- In the **least squares method**, the optimal values β_j are those that **minimize** the **sum of squared residuals**:

$$S \equiv \sum_i r_i^2, \quad \text{where} \quad r_i \equiv y_i - f(x_i, \beta_j).$$

- Here r_i are **residuals**: the differences between the y -data and the model.
- To find the minimum of S we set all of its partial derivatives to zero w.r.t. the parameters β_j :

$$\frac{\partial S}{\partial \beta_j} = 0.$$

Least squares method

- Let's apply this method to **derive** the parameters of the line of best fit.

- Model:** $f(x, \beta_j) = \beta_0 + \beta_1 x$

- The **sum-of-squared-residuals** is

$$S = \sum_i r_i^2 = \sum_i (y_i - \beta_0 - \beta_1 x_i)^2$$

- S has a stationary point where $\frac{\partial S}{\partial \beta_0} = 0 = \frac{\partial S}{\partial \beta_1}$

- The partial derivative of S w.r.t. β_0 is

$$\frac{\partial S}{\partial \beta_0} = 2 \sum_i r_i \frac{\partial r_i}{\partial \beta_0} = -2 \sum_i (y_i - \beta_0 - \beta_1 x_i) = 0$$

- Divide by N to write as

$$\frac{1}{N} \sum_i (y_i - \beta_0 - \beta_1 x_i) = 0 \quad \Rightarrow \quad \boxed{\bar{y} = \beta_0 + \beta_1 \bar{x}}$$

Least squares method

- Partial derivative w.r.t. $\beta_0 \Rightarrow \boxed{\bar{y} = \beta_0 + \beta_1 \bar{x}}$
- Partial derivative w.r.t. β_1 :

$$\frac{\partial S}{\partial \beta_1} = 2 \sum_i r_i \frac{\partial r_i}{\partial \beta_1} = -2 \sum_i (y_i - \beta_0 - \beta_1 x_i) x_i = 0$$

- Divide by N and rearrange to get

$$\boxed{\overline{xy} = \beta_0 \bar{x} + \beta_1 \overline{x^2}}$$

- Solving the boxed equations for β_1 and β_0 gives

$$\beta_1 = \frac{\overline{xy} - \bar{x} \bar{y}}{\overline{x^2} - \bar{x}^2} = \frac{\text{covar}(x, y)}{\text{var}(x)}$$

and $\beta_0 = \bar{y} - \beta_1 \bar{x}$.

Linear models

- We showed that the method of least squares leads to the standard formulae for parameters β_0, β_1 in the straight-line model $f(x) = \beta_0 + \beta_1 x$.
- Next we will consider the wide class of **linear models**:

$$f(x, \beta_j) = \sum_{j=0}^{m-1} \beta_j \phi_j(x)$$

where $\phi_j(x)$ is **any** function of x .

- Note that linear models are **linear in the parameters**, but **not** necessarily linear in x .
- e.g. $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$ is a linear model,
but $f(x) = \exp(\beta_0 x)$ is not.

Linear models

- Consider a **linear model** with m parameters

$$f(x, \beta_j) = \sum_{j=0}^{m-1} \beta_j \phi_j(x)$$

and a data set with N data points, such that $N > m$.

- The best-fit parameters β_j are found by solving matrix equations known as the **normal equations**

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y}$$

- Here $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots)^T$ and $\mathbf{y} = (y_0, y_1, \dots)^T$ are vectors of length m and N , respectively, and \mathbf{X} is $N \times m$:

$$\mathbf{X} \equiv \begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \dots & \phi_{m-1}(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_{m-1}(x_1) \\ \vdots & \vdots & & \vdots \\ \phi_0(x_{N-1}) & \phi_1(x_{N-1}) & \dots & \phi_{m-1}(x_{N-1}) \end{pmatrix}$$

Linear models

- Let's **derive** the normal equations for linear model

$$f(\mathbf{x}, \beta_j) = \sum_j \beta_j \phi_j(\mathbf{x})$$

- Let X_{ij} denote the element in i th row, j th column of \mathbf{X} .

$$X_{ij} = \phi_j(\mathbf{x}_i)$$

- Consider the i th **residual**:

$$r_i = y_i - f(\mathbf{x}_i, \beta_j) = y_i - \sum_j \beta_j \phi_j(\mathbf{x}_i)$$

and its partial derivative w.r.t. β_k :

$$\frac{\partial r_i}{\partial \beta_k} = - \sum_j \frac{\partial \beta_j}{\partial \beta_k} \phi_j(\mathbf{x}_i) = - \sum_j \delta_{jk} \phi_j(\mathbf{x}_i) = -\phi_k(\mathbf{x}_i) = -X_{ik}.$$

Linear models

- Now minimize the sum-of-square-residuals S :

$$\frac{\partial S}{\partial \beta_k} = \frac{\partial}{\partial \beta_k} \left(\sum_i r_i^2 \right) = 2 \sum_i r_i \frac{\partial r_i}{\partial \beta_k} = 0$$

- Inserting $\frac{\partial r_i}{\partial \beta_k} = -X_{ik} = -(X^T)_{ki}$ and $r_i = y_i - \sum_j X_{ij}\beta_j$

$$\Rightarrow - \sum_i (X^T)_{ki} \left(y_i - \sum_j X_{ij}\beta_j \right) = 0$$

- This is the j th row of a vector in the matrix equation,

$$\mathbf{X}^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{0},$$

or, rearranging,

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y}.$$

Linear models

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y}.$$

- How should we solve the matrix equations to find best-fit parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots)^T$?
- Naive method: find the inverse of the $m \times m$ square matrix $\mathbf{X}^T \mathbf{X}$ and apply to both sides.
- Better method:
 - Check that equation is **well-conditioned**.
 - Apply Gaussian elimination or other efficient method.
- (We will consider linear algebra and methods for solving $\mathbf{Ax} = \mathbf{b}$ in the next lecture)

Linear models

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y}.$$

- Here's a crude implementation, to see the method working in practice. Let's consider a random quadratic with noise.

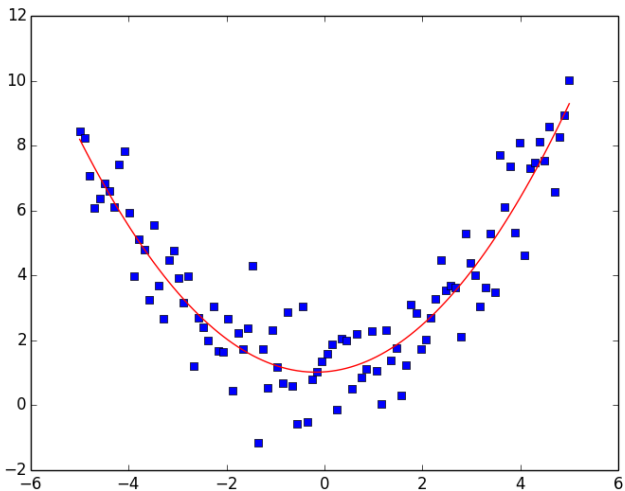
```
import numpy as np
import matplotlib.pyplot as plt
### Make an example data set: random quadratic with noise
N = 100
coef = np.random.random(3)
x = np.linspace(-5, 5, N)
y0 = coef[0]*x**2 + coef[1]*x + coef[2]
sigma = 1.0
y = y0 + sigma*np.random.normal(size=N)
```

Linear models

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y}.$$

```
### Fit a quadratic
m = 3 # number of parameters
X = np.zeros((N, m)) # an N x m matrix
for i in range(N):
    X[i,:] = 1.0, x[i], x[i]**2
A = np.dot(np.transpose(X), X)
b = np.dot(np.transpose(X), y)
beta = np.linalg.solve(A, b) # best-fit parameters
```

```
### Plot
y_est = beta[0] + beta[1]*x + beta[2]*x**2
plt.plot(x, y, 's'); plt.plot(x, y_est, 'r-')
plt.show()
```

Example: Least-squares fit to quadratic

$$f(x, \beta_i) = \beta_0 + \beta_1 x + \beta_2 x^2.$$

Non-linear models

- What about **non-linear models**? e.g.

$$\beta_1 x^{\beta_0}, \quad \text{or} \quad \beta_0 \sin(\beta_1 x + \beta_2)$$

- There is **no closed-form solution** for β_i
- Start with a guess $\beta^{[0]}$ and **iterate** to get $\beta^{[k]} \dots$
- Test for convergence:

$$\|\beta^{[k+1]} - \beta^{[k]}\| \leq \epsilon$$

Non-linear models

The Gauss-Newton method

- Choose a starting guess for parameters $\beta^{[0]}$
- Apply

$$\beta^{[k+1]} = \beta^{[k]} + \Delta\beta$$

where $\Delta\beta$ is the solution to

$$(\mathbf{J}^T \mathbf{J}) \Delta\beta = \mathbf{J}^T \Delta\mathbf{y}$$

- Here $\Delta\mathbf{y} = [\Delta y_i]$ where

$$\Delta y_i = y_i - f(x_i, \beta_j^{[k]}).$$

- The **Jacobian** $\mathbf{J} = [J_{ij}]$ is

$$J_{ij} = \frac{\partial f}{\partial \beta_j}(x_i, \beta_j^{[k]})$$

Non-linear models

- Iterative methods (such as Gauss-Newton) require an **initial guess**.
- **Convergence is not guaranteed**. Convergence depends on choice of initial guess (cf. Newton-Raphson method).
- There may be **multiple minima!**
- If the model is **linear**, the iterative approach will find the solution in one step.

Limitations / Extensions / Questions

- We have assumed that there is no error in independent variable x . What if this is not the case? (see Errors-in-variables models)
- How do we estimate the uncertainties in the best-fit parameters? (see 'boot-strap' method).
- What if there are several dependent or independent variables?
- What if we are not sure of the form of the underlying model?
- Adding more parameters always leads to a better fit . . . but how do we determine whether extra parameters are really necessary?
- What is the connection between the least-squares method and the normal distribution?

Fitting data with Python

- The module `scipy.optimize` provides several useful functions:
 - `minimize` : find minimum of a function
 - `leastsq` : minimize the sum-of-squares of a set of equations
 - `curve_fit` : fit a non-linear model to data using least-squares method
 - `fsolve` : find the roots of a function
- Let's try using `curve_fit` with a non-linear model:

$$f(x) = \beta_0 \exp(\beta_1 x) \sin(10\beta_2 x) + \beta_3$$

Fitting data with Python

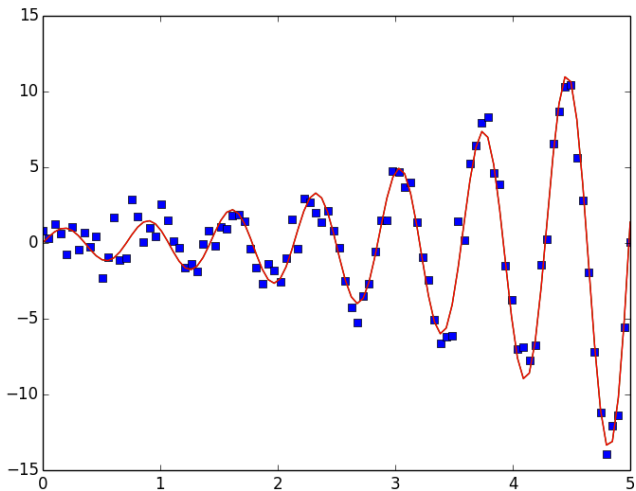
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

def func(x, a, b, c, d):
    """A function which is non-linear in its parameters a,b,c,d."""
    return a*np.exp(b*x)*np.sin(10*c*x) + d

# Make a sample data set.
N = 100
m = 4
coef = np.random.random(m)
x = np.linspace(0, 5, N)
y0 = func(x, coef[0], coef[1], coef[2], coef[3])
sigma = 1.0
y = y0 + sigma*np.random.normal(size=N)
```

Fitting data with Python

```
params0 = [0.5,0.5,coef[2],0.5] # Try changing this!  
# I find that the results are not good unless  
# the starting guess for the frequency is accurate!  
  
ps, pcov = curve_fit(func, x, y0, p0=params0)  
# 'ps' is array of best-fit parameters.  
# 'pcov' is the covariance matrix.  
y_true = func(x, coef[0], coef[1], coef[2], coef[3])  
y_est = func(x, ps[0], ps[1], ps[2], ps[3])  
plt.plot(x, y, 's')  
plt.plot(x, y_true, '-')  
plt.plot(x, y_est, 'r-')
```

Example: Fit to non-linear model

$$f(x, \beta_i) = \beta_0 \exp(\beta_1 x) \sin(10\beta_2 x) + \beta_3.$$