MPS224 Scientific Computing

Dr. Sam Dolan

School of Mathematical and Physical Sciences,
University of Sheffield

Spring 2026

8. Data analysis and curve-fitting

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

Today's lecture: Working with data

- Scientific computing packages:
  - `numpy`
  - `matplotlib`
  - `pandas`
- Example data sets:
  - British earthquakes
  - Popular names in 1904 and 2019
  - Hills in the Peak District
- Fitting a model to data
  - Linear models (theory)
  - Non-linear models

pandas

- pandas is a **data analysis tool** that is built on numpy.
  https://pandas.pydata.org/
- The key data structure is the DataFrame: a collection of Series.

	Name	Height	Drop	Latitude	Longitude	County
0	Kinder Scout	636.3	496.6	53.384808	-1.873910	Derbyshire
1	Bleaklow Head	633.0	128.0	53.461267	-1.858959	Derbyshire
2	Higher Shelf Stones	621.8	19.8	53.449816	-1.867610	Derbyshire
3	Black Hill	582.0	165.0	53.538013	-1.885306	Derbyshire/Kirklees
4	Shining Tor	559.0	235.7	53.260729	-2.009472	Cheshire East/Derbyshire
...	...	...	...	...	...	...

- The R language also uses DataFrames
- Tutorials:
  https://pandas.pydata.org/docs/getting_started
  https://www.learnpython.org/en/Pandas_Basics

`DataFrame` and `Series`

- In essence, a `DataFrame` is a table of data.

	Date	Time (UTC)	Lat	Lon	Depth (km)	Mag
0	2019/09/02	05:33:07.3	53.785	-2.964	2	-1.0
1	2019/09/02	03:12:21.3	53.786	-2.966	2	0.0
2	2019/09/02	03:12:37.3	53.786	-2.963	2	0.2
3	2019/09/01	18:42:07.7	53.785	-2.965	2	-0.9
4	2019/08/29	07:00:05.6	53.784	-2.964	1	-1.0

- A `Series` is a named **column** of values.
- A `DataFrame` consists of multiple `Series`.
  Each `Series` has a 1D array with a `dtype`.
- Tables are commonly stored as `.csv` files
  (CSV = Comma-Separated Values).

1. British earthquakes

Background: In August 2019, hydraulic fracturing (fracking) operations were undertaken at the Preston New Road site in Lancashire. Hydraulic fracturing involves the high-pressure injection of fluid at a depth of $\sim 1.8\,\text{km}$, and typically generates *seismic events*. The majority of such seismic events are rather weak, with a Local Magnitude (ML) below 0.5 ML.

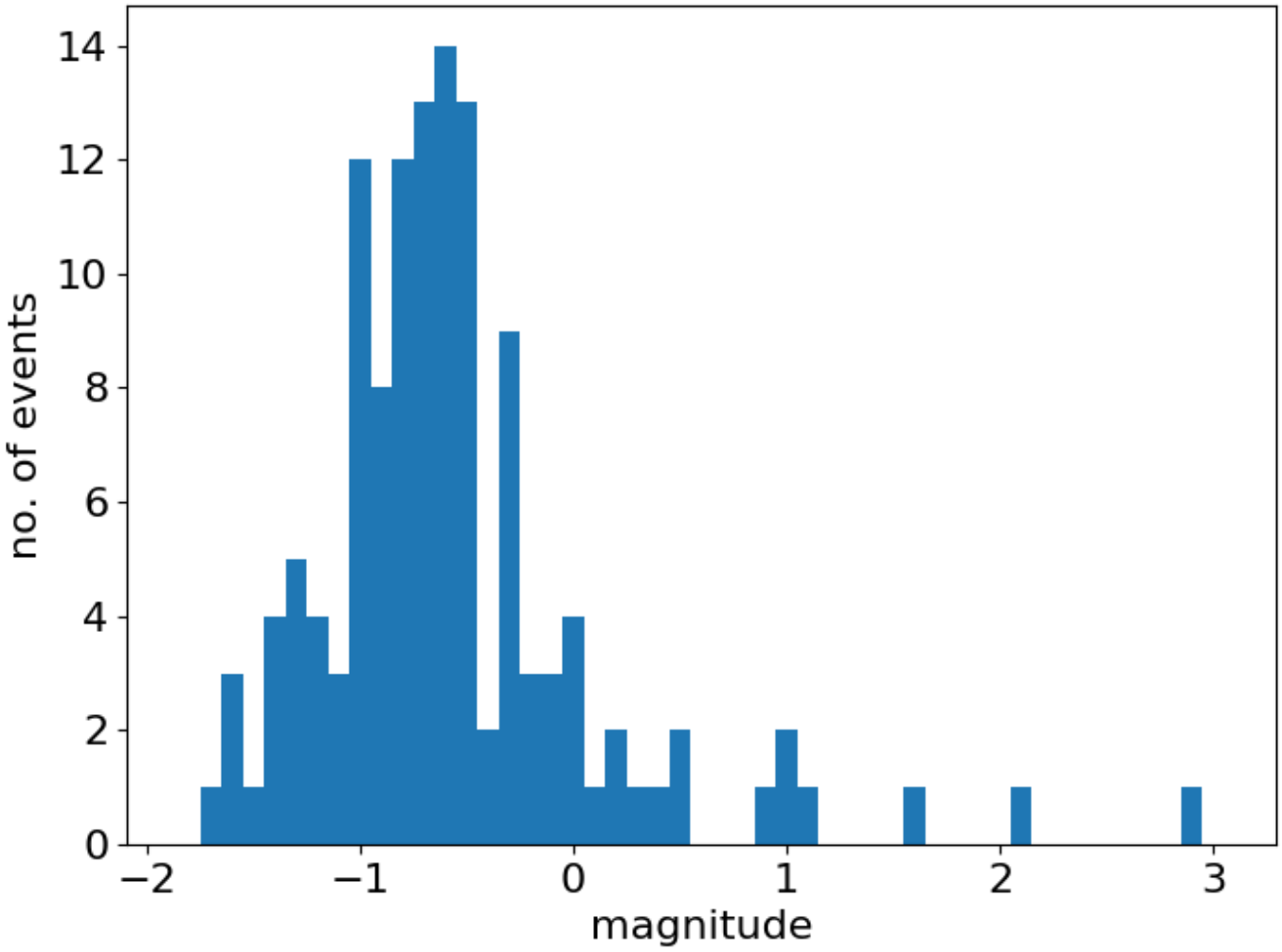
At 08:30am on 26th August, a 2.9 ML tremor was recorded by the British Geological Survey (BGS). This event was reportedly felt by some residents of Blackpool, 4 miles from the site, as a loud tremor lasting for 10–15 seconds. Fracking was subsequently suspended at the site.

Task (hypothetical): The Oil and Gas Authority have commissioned you to study seismic data from the fracking operations. Your analysis will inform their decision on whether fracking should resume.

The British Geological Society keep track of live seismic activity in and around the British Isles, here:
`http://www.earthquakes.bgs.ac.uk/earthquakes/recent_uk_events.html`

1. Earthquakes and fracking: task

- From the data file `seismic_events.csv`, make a histogram of the magnitude (ML) of the seismic events.



Convention:

Import `pandas` as follows:

```
>>> import pandas as pd
```

Basic code

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>> df = pd.read_csv('seismic_events.csv') # read from file
>>> print(df.shape) # How many rows and columns?
>>> df.head()  # show the first few rows
>>> df.hist('Mag')  # draw a histogram of the event magnitudes
```

- `read_csv()` : makes a new `DataFrame` from a Comma-Separated Values file.

Objects

- DataFrame: `df = pd.read_csv('seismic_events.csv')`
- Series : `df['Mag']`
- 1D array : `df['Mag'].values`
- the data-type: `df['Mag'].values.dtype`

Indexing

- `df['Mag']` : the column (series) named 'Mag'
- `df.loc[:, 'Mag']` : same as above
- `df.loc[0]` : the first row:
- `df.loc[1, 'Mag']` : the 2nd value in series 'Mag':
- `df.loc[list(range(1,3))]` : the 2nd and 3rd rows.

Subsetting / filtering

- To return a subset of seismic events (rows) in a DataFrame `df` with a local magnitude exceeding zero:

```
>>> df[df['Mag'] > 0]
  # a view of all rows with a magnitude > zero.
>>> df2 = df[df['Mag'] > 0].copy()
  # a new DataFrame from those rows
>>> print(df2.shape[0])
  # count how many events with a magnitude > 0.
```

- More complex filters can be constructed:

```
>>> df[ (df['Mag'] > 0) & (df['Lon'] > -2.96) ]
```

2. Popular names

- What names were popular in 1904 **and** in 2019?
- One way to answer this is to **merge** two data frames using the column `Name`.
- The function `pd.merge()` will perform a database-style join.

```
df = pd.merge(df2019, df1999, on="Name")
print(df.head())
```

	Rank_x	Name	Count_x	Year_x	Rank_y	Count_y	Year_y
0	1	Olivia	4082	2019	4	5250	1999
1	2	Amelia	3712	2019	37	1511	1999
2	5	Mia	2500	2019	54	980	1999

2. Popular names

- **Q.** Which names were most popular overall?
- **A.** Add a series with the cumulative count (1904 + 2019) and sort:

```
df["CumulCount"] = df["Count_x"] + df["Count_y"]
df = df.sort_values("CumulCount", ascending=False)
print(df.head())
```

	Name	CumulCount
21	Chloe	10221
0	Olivia	9332
8	Emily	9022
12	Charlotte	6979

3. Hills in the Peak District

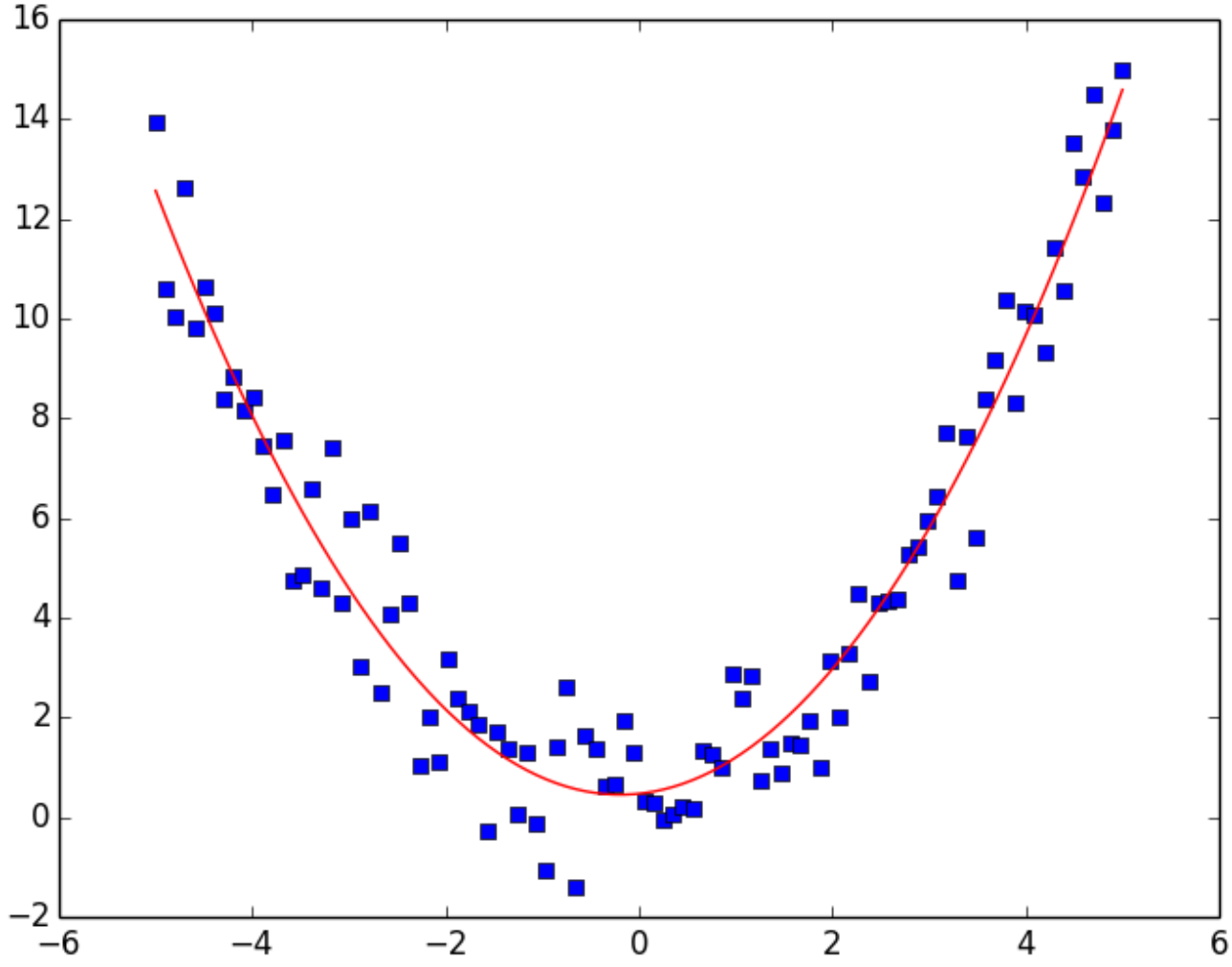
	Name	Height	Drop	Latitude	Longitude	County
0	Kinder Scout	636.3	496.6	53.384808	-1.873910	Derbyshire
1	Bleaklow Head	633.0	128.0	53.461267	-1.858959	Derbyshire
2	Higher Shelf Stones	621.8	19.8	53.449816	-1.867610	Derbyshire
3	Black Hill	582.0	165.0	53.538013	-1.885306	Derbyshire/Kirklees
4	Shining Tor	559.0	235.7	53.260729	-2.009472	Cheshire East/Derbyshire
...	...	...	...	...	...	...

Typical tasks:

1. **Read:** read the data frame file
2. **Sort:** sort the hills by height
3. **Filter:** find all hills higher than 500m.
4. **Query:** is Mam Tor the highest hill in the Peaks?
5. **Plot**: show a histogram of hills by height.

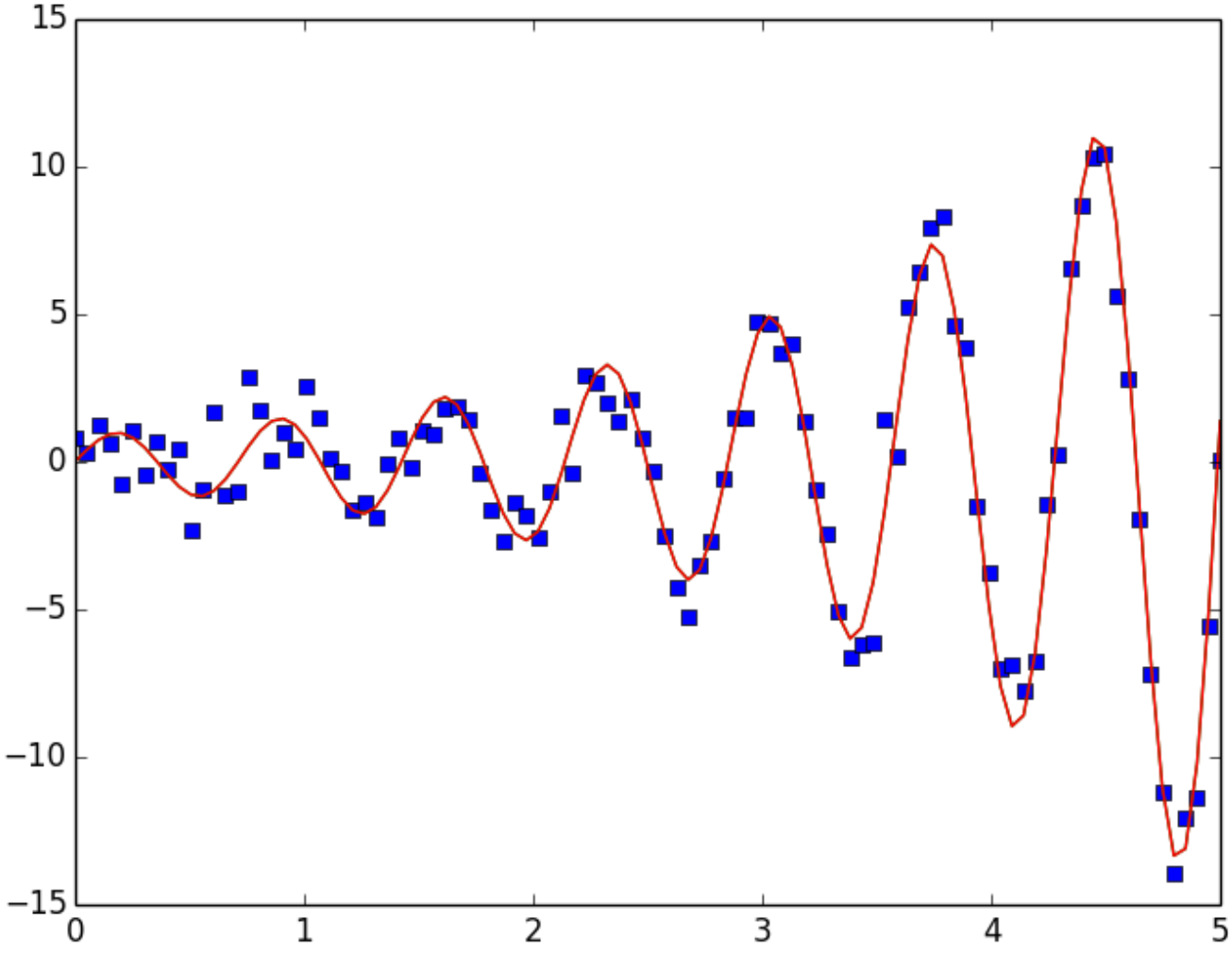
Models and curve fitting

- Fitting models to data
- Theory:
  - Finding the line of best fit (linear regression)
  - 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 $i = 0 \ldots N - 1$
- How do I find the **line of best fit**?
- That is, how do I fit a two-parameter model to the data?

  $f(x; \beta_0, \beta_1) = \beta_0 + \beta_1 x$ where β_0, β_1 are model parameters

- Here, the two parameters are the **gradient** (β_1) and the **intercept** (β_0).
- (Typically we use m for the gradient and c for the intercept).

Example: Fitting a straight line

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

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

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

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

- The **variance** and **covariance** are:

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

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

- 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**:

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

- Here r_i are **residuals**: the differences between the y-data and the model.

Least squares method

- The **sum of squared residuals** S is a function of the parameters:

$$S = S(\beta_j)$$

- To find the minimum of S we set all of its partial derivatives to zero w.r.t. the parameters β_j:

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

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

Least squares method

- **Model:** $f(x, \beta_j) = \beta_0 + \beta_1 x$
- The ith **residual** is $r_i \equiv y_i - f(x_i, \beta_j) = y_i - \beta_0 - \beta_1 x_i$.
- The **sum-of-squared-residuals** is

$$S = \sum_i r_i^2$$

- 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

$$\frac{\partial S}{\partial \beta_0} = 2 \sum_i r_i \frac{\partial r_i}{\partial \beta_0} = \ = -2 \sum_i r_i = 0$$

- Divide by N to write as

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

Least squares method

- Now take 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,

$$\frac{1}{N}\sum_i x_i y_i = \beta_0 \frac{1}{N}\sum_i x_i + \beta_1 \frac{1}{N}\sum_i x_i^2$$

- Written using the overbar notation,

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

- Solving the boxed equations for β_1 and β_0 gives

$$\begin{aligned} \beta_1 &= \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2} = \frac{\mathrm{covar}(x,y)}{\mathrm{var}(x)} \\ \beta_0 &= \overline{y} - \beta_1 \overline{x}. \end{aligned}$$

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 (x_i, y_i) 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**:

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

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

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

Linear models

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

$$f(x, \beta_j) = \sum_j \beta_j \, \phi_j(x)$$

- Let X_{ij} denote the element in ith row, jth column of $\mathbf{X}$.

$$X_{ij} = \phi_j(x_i)$$

- Consider the ith **residual**:

$$r_i = y_i - f(x_i, \beta_j) = y_i - \sum_j \beta_j \phi_j(x_i)$$

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

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

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 $r_i = y_i - \sum_j X_{ij}\beta_j$

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

- This is the kth 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

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

- How should we solve the normal equations to find best-fit parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots)^T$?
- 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.
- Methods for solving linear systems $\mathbf{Ax} = \mathbf{b}$ are well established.

Linear models

$$(\mathbf{X}^T\mathbf{X})\,\beta = \mathbf{X}^T\mathbf{y}.$$

- Here's an implementation, to see the method working in practice.
- First, simulate some data based on 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

$$\boxed{(\mathbf{X}^T\mathbf{X})\,\beta = \mathbf{X}^T\mathbf{y}.}$$

- Now we wish to fit a three-parameter model:

$$\begin{aligned} f(x;\beta_j) &= \beta_0 + \beta_1 x + \beta_2 x^2 \\ &= \sum_{j=0}^{2} \beta_j\,\phi_j(x). \end{aligned}$$

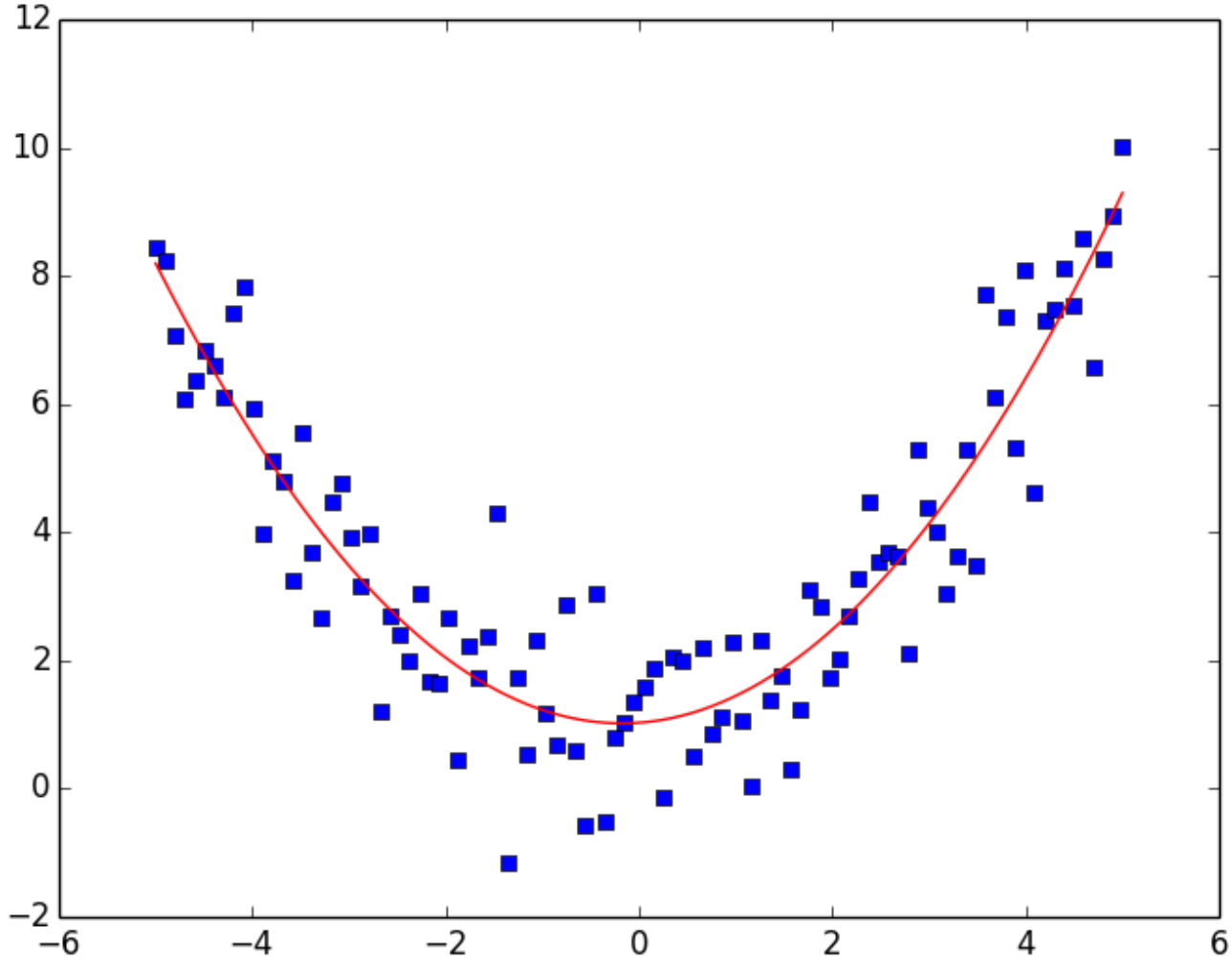
- That is, $\phi_0(x) = 1$, $\phi_1(x) = x$ and $\phi_2(x) = x^2$.

Linear models

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

```
### 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.

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

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

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

The Gauss-Newton method

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

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

where $\Delta\boldsymbol{\beta}$ is the solution to

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

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

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

- 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 **just one step**.

Fitting data with Python

- The module `scipy.optimize` provides several useful functions:
  - `minimize` : find minimum of a function
  - `fsolve` : find the roots 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
- Let's try using `curve_fit` with a non-linear model:

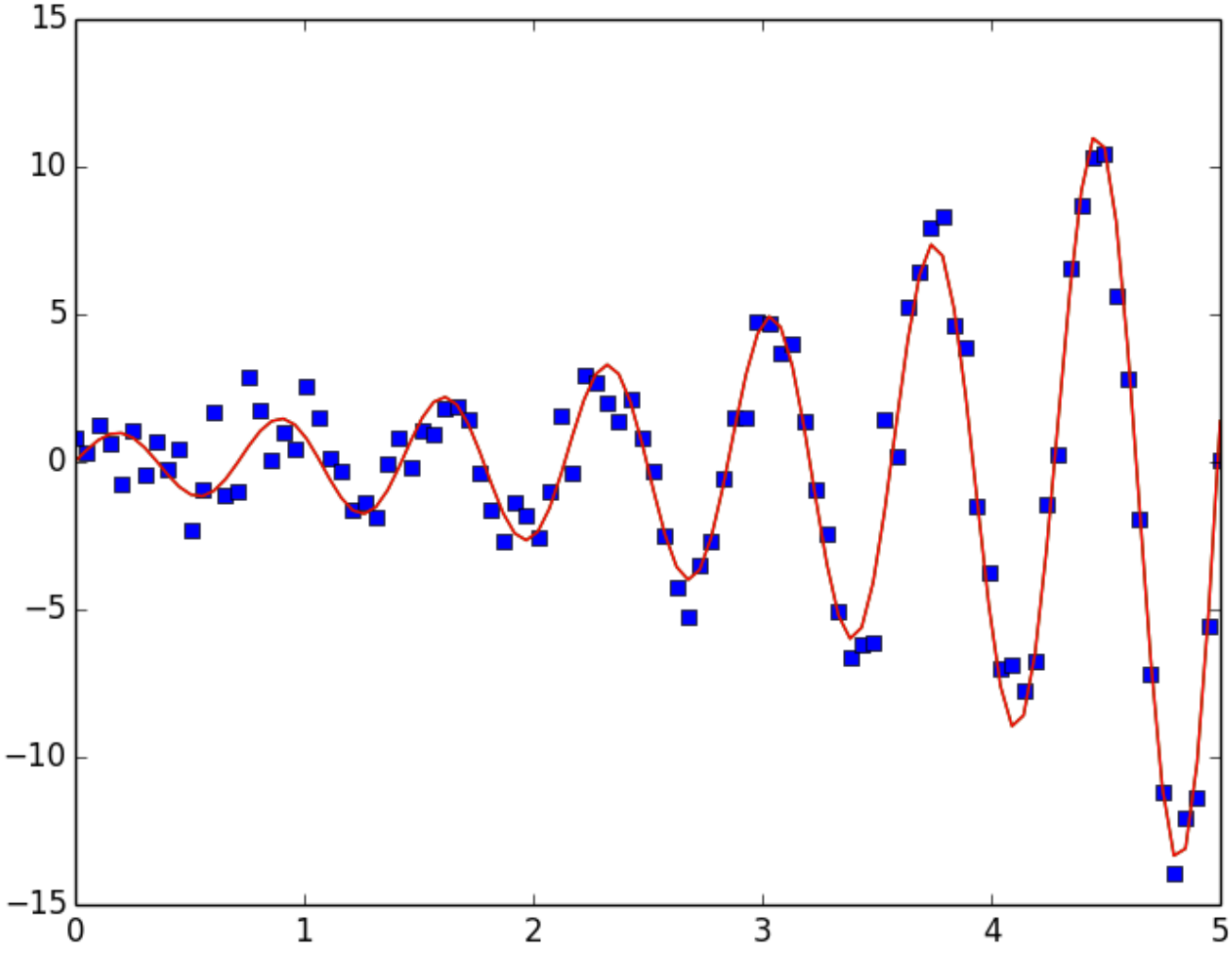
$$f(x) = \beta_0 \exp(\beta_1 x) \sin(10\beta_2 x) + \beta_3$$

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

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