ENGG1003 - Monday Week 11

Fitting curves to data: beyond straight-line fit

Steve Weller

University of Newcastle

17 May 2021

Last compiled: May 16, 2021 9:32pm +10:00

Lecture overview

1. recap: fitting straight line to data

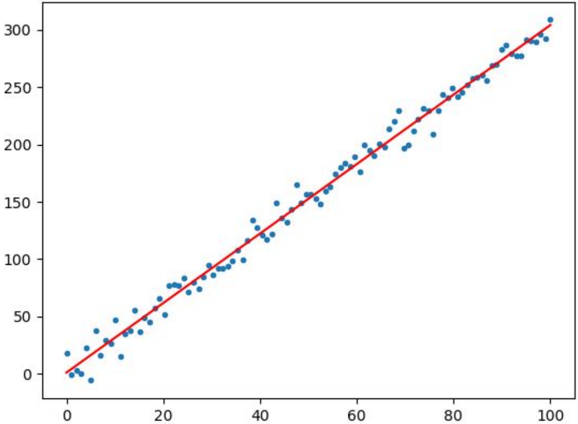
2. least squares fit
   - describe concept of *least squares*
   - "do it yourself" best straight-line fit
   - Python code to fit a straight line to data (DIY)

3. beyond straight-line fit
   - Python code to fit a polynomial (eg: parabola, cubic)

4. preliminary discussion of the final exam

1) Recap: fitting straight line to data

- recap from Monday week 10, pp. 24–25
- output generated by `linefitdemo.py`
- blue dots: given data red line: line-of-best-fit



Recap: line-fitting in Python

- input data consists of (x, y) data pairs
- goal is to calculate gradient m and y-intercept b of line-of-best-fit

$$y = mx + b$$

- in Python, we use `curve_fit()` function in `scipy.optimize` library to find m and b
  - may need `pip install scipy` in terminal

```
popt, pcov = curve_fit(line, x, y)
m = popt[0]
b = popt[1]
```

- ignore `pcov` returned by `curve_fit`

Two questions

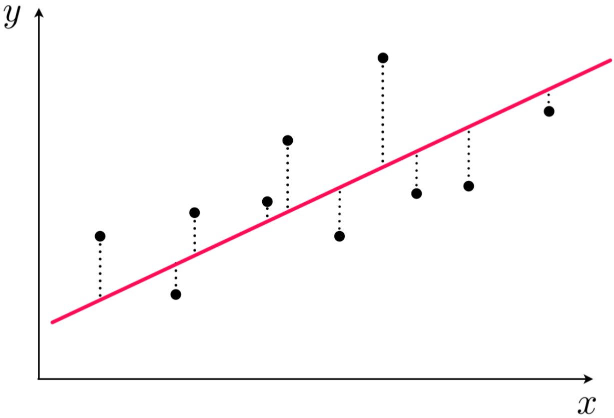
1. how do we define "best fit"?

2. how is equation of line-of-best-fit calculated?
   - how does `curve_fit()` function in `scipy.optimize` library actually work?
   - how are gradient m and y-intercept b actually calculated?

2) Least squares fit

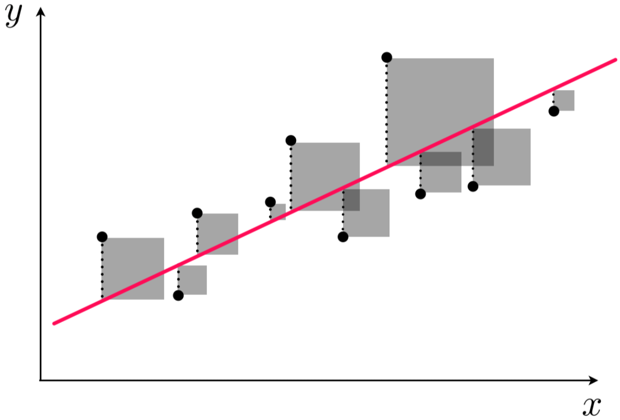
- "best" straight-line minimises size of "error" between the line and the data points
  - definition of "best" and "error" are somewhat arbitrary...
- BUT method of least squares is standard approach
  - overwhelmingly the most commonly used in Engineering
  - also the basis for more advanced methods

Residuals



- for any choice of straight line, residuals are shown as dotted lines (different lines $\Rightarrow$ different residuals)
- goal is to choose the line with the smallest residuals

Method of least squares



- method of least squares calculates the line which makes *total area of grey squares* as small as possible
- ie: minimises sum of squares of residuals

Method of least squares

- for every choice of m and b, can compute total area of grey squares

- to find minimum (least value), does that mean we have to search over all possible choices of m and b?

- **NO!** — there are equations for m and b which minimise total area of grey squares

- we'll present those equations in a few slides: great opportunity to write some Python code
  - compare results with `curve_fit()` function in `scipy.optimize`

Sigma notation: $\sum$

- equations for best (least squares) choice of m and b use sigma notation
- $\sum$ here denotes "summing up"

$$\sum_{k=0}^{N-1} x_k = x_0 + x_1 + \cdots + x_{N-1}$$

- in Python:
  - data in length-N array `x[0], x[1], ..., x[N-1]`
  - use a loop to calculate sum: `for k in range(0,N)`

Least squares straight-line fit

- input data for straight-line fit problem is N pairs

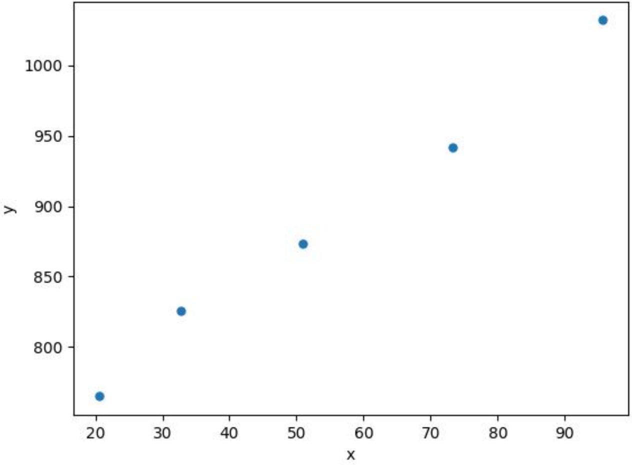
$$(x_0, y_0), (x_1, y_1), \ldots (x_{N-1}, y_{N-1})$$

Example: effect of temperature T on resistance R

T (°C)	R (ohms)
20.5	765
32.7	826
51.0	873
73.2	942
95.7	1032

- we'll use $x = T$ and $y = R$

output of `LSlinefitData.py`




```
x = np.array([20.5, 32.7, 51.0, 73.2, 95.7])
y = np.array([765, 826, 873, 942, 1032])
```

Least squares straight-line fit

Aim: find $\mathbf{m}$ and $\mathbf{b}$ in least squares straight-line fit

$$\boxed{y = \mathbf{m}x + \mathbf{b}}$$

- define

$$\bar{\boldsymbol{x}} = \frac{\sum_{k=0}^{N-1} x_k}{N} \qquad \bar{\boldsymbol{y}} = \frac{\sum_{k=0}^{N-1} y_k}{N}$$

  - $\bar{x}$ and $\bar{y}$ are averages (means) of x and y arrays

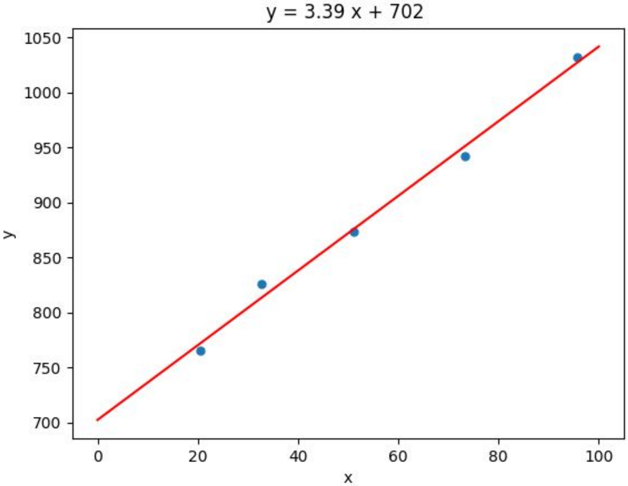
Equations for best straight-line fit

$$\mathbf{m} = \frac{\sum_{k=0}^{N-1}(x_k - \bar{\boldsymbol{x}})(y_k - \bar{\boldsymbol{y}})}{\sum_{k=0}^{N-1}(x_k - \bar{\boldsymbol{x}})^2}$$

$$\mathbf{b} = \bar{\boldsymbol{y}} - \mathbf{m}\bar{\boldsymbol{x}}$$

```
N = len(x)
xbar = np.mean(x)
ybar = np.mean(y)
mnum = 0    # numerator of m
mden = 0    # denominator of m
for k in range(0,N):
    mnum += (x[k]-xbar)*(y[k]-ybar)
    mden += (x[k]-xbar)**2
m = mnum/mden
b = ybar - m*xbar
```

output of `LSlinefit.py`



- line of best-fit using least squares equations:

$$y = 3.39x + 702$$

Python code

LSlinefit.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 
4 def line(x, m, b):
5     return m * x + b
6 
7 x = np.array([20.5, 32.7, 51.0, 73.2, 95.7])    # temp (degC)
8 y = np.array([765, 826, 873, 942, 1032])  # resistance (ohms)
9 plt.plot(x, y, '.', markersize=10)
```

- lines 4–5: prepare to plot straight line obtained by least squares fit
- lines 7–9: plot the (x, y) data as blue dots

Python code

LSlinefit.py—continued

```
1  N = len(x)
2  xbar = np.mean(x)
3  ybar = np.mean(y)
4  mnum = 0     # numerator of m
5  mden = 0     # denominator of m
6  for k in range(0,N):
7      mnum += (x[k]-xbar)*(y[k]-ybar)
8      mden += (x[k]-xbar)**2
9  m = mnum/mden
10 b = ybar - m*xbar
11
12 xfine = np.linspace(0., 100., 100)
13 plt.plot(xfine, line(xfine, m, b), 'r')
14 plt.title('y = {:.2f} x + {:.0f} '.format(m, b))
15 plt.xlabel('x')
16 plt.ylabel('y')
17 plt.show()
```

- lines 1–10: equations for best straight-line fit
- lines 12–13: plot straight-line fit (red line)

Straight-line fit using `curve_fit()`

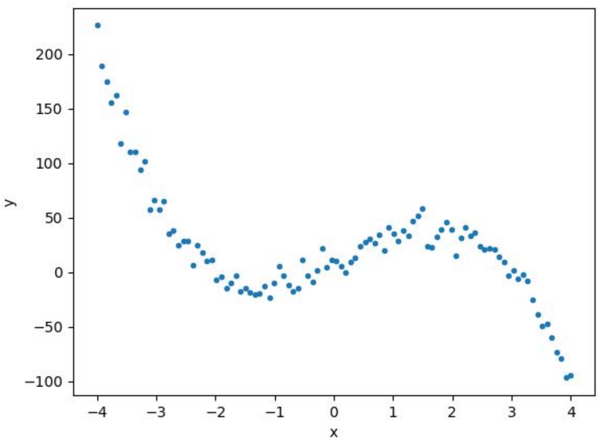
- obtain identical results using `curve_fit()` function in `scipy.optimize`
- simply replace lines 1–10 on previous slide with:

```
popt, pcov = curve_fit(line, x, y)
m = popt[0]
b = popt[1]
```

- code for `curve_fit()` version in `resistancetemp.py`
  - posted in BB and #lecturecode

3) Beyond straight-line fit

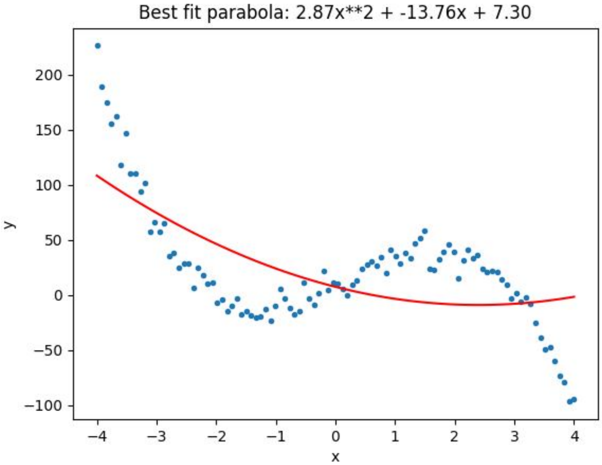
- what if data not well described by a straight line?



Curve-fitting with polynomials

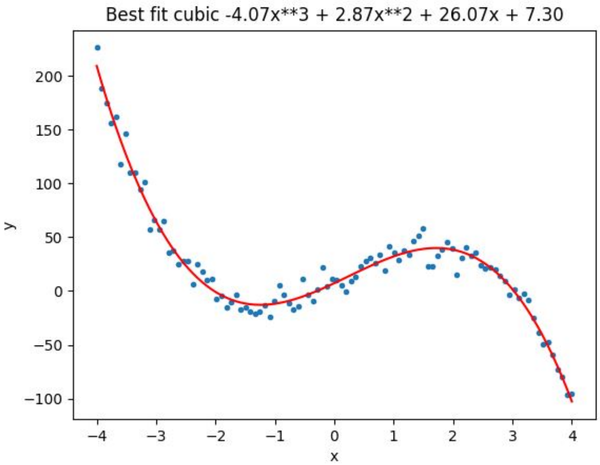
- `curve_fit()` function can also be used to fit other curves, eg:
  - parabolas: $ax^2 + bx + c$
  - cubic polynomials: $ax^3 + bx^2 + cx + d$
  - higher-order polynomials (order-4, -5 etc)
  - other "nonlinear" functions, eg: $e^{Bx}, \sin(Cx), 1 - e^{-Dx^2}, \ldots$ and combinations of these
- `curve_fit()` uses the method of least squares to fit these curves, too
- sometimes physics / creativity / guesswork needed on the "right" model to fit

Fit parabola to data



`curve_fit()` finds best $a, b, c: \quad ax^2 + bx + c$

Fit cubic to data



`curve_fit()` finds best $a, b, c, d:\ ax^3 + bx^2 + cx + d$

Python code: nonlinearguess1.py

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

# guess data is a parabola
def parabola(x, a, b, c):
    return a*x**2 + b*x + c

np.random.seed(1)   # replicate results by fixing seed
# data is actually a cubic + noise
x = np.linspace(-4, 4, 100)
y = -4*x**3 + 3*x**2 + 25*x + 6 + np.random.normal(0., 10, len(x))
plt.plot(x, y, '.')

popt, pcov = curve_fit(parabola, x, y)
a = popt[0]; b = popt[1]; c = popt[2]

plt.plot(x, parabola(x, a, b, c), 'r')
plt.title('Best fit parabola: {:.2f}x**2 + {:.2f}x + {:.2f}'.format(a,b,c))
plt.xlabel('x'); plt.ylabel('y')
plt.show()
```

Code commentary

- lines 6–7: fit a parabola to the data: $ax^2 + bx + c$
- lines 11–12: data is cubic polynomial + noise
  - but `curve_fit()` doesn't know this!
- lines 15–16 call the `curve_fit()` function and ask it to find best-fit parabola

Python code: nonlinearguess2.py

- fitting a *cubic* polynomial $ax^3 + bx^2 + cx + d$ to the data requires only a few changes to code in `nonlinearguess1.py`
- see BB for full code listing of `nonlinearguess2.py`
- define cubic function to fit to data:

```
def cubic(x, a, b, c, d):
    return a*x**3 + b*x**2 + c*x + d
```

- call `curve_fit()` to find best values of a, b, c, d

```
popt, pcov = curve_fit(cubic, x, y)
a = popt[0]; b = popt[1]; c = popt[2]; d = popt[3]
```

Lecture summary

- least squares fit
  - basic concept of least squares
  - "do it yourself" straight-line fit
- beyond straight-line fit
  - fitting polynomials to data
- preliminary discussion of the final exam