ENGG1003 - Monday Week 11

Fitting curves to data: beyond straight-line fit

Steve Weller

University of Newcastle

17 May 2021

Last compiled: May 15, 2021 4:18pm +10:00

Lecture overview

1. some key ideas for assignment 2 (???)

2. recap: fitting straight line to data

3. least squares fit
   - explain concept of *least squares*
   - closed-form expression for best straight-line fit
   - Python code to fit a straight line to data (closed-form)

4. beyond straight-line fit
   - Python code to fit a polynomial (eg: third order)
   - Python code to fit a decaying exponential

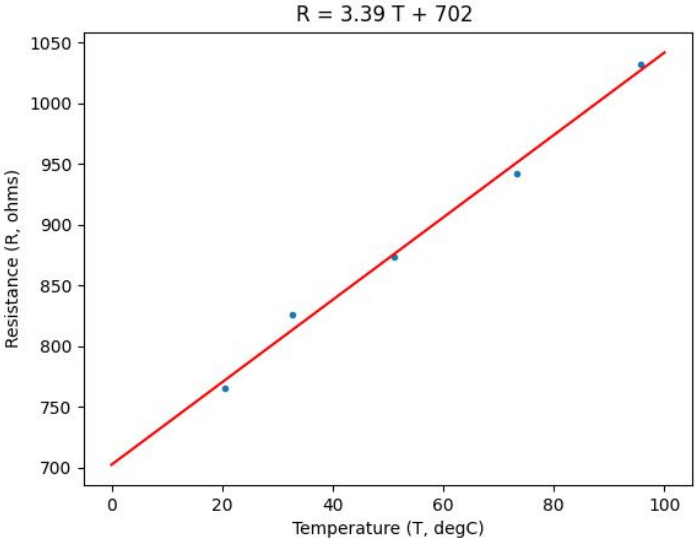
1) Recap: fitting straight line to data

- recap from Monday last week
- Monday week 10, pp 24–25

Example

- effect of temperature on electrical resistance
- Gerald & Wheatley, page 531
- only 5 data point pairs, ideal setup for extended example illustrating least squares and closed-form expression for straight-line fit
- data, see p 531 G&W
- want to find constants m and b in equation relating resistance R and temperature T:

$$R = mT + b$$


R = 3.39 T + 702
1050
1000
950
900
850
800
750
700
Resistance (R, ohms)
0
20
40
60
80
100
Temperature (T, degC)

Python code

resistancetemp.py

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

def line(x, m, b):
    return m * x + b

T = np.array([20.5, 32.7, 51.0, 73.2, 95.7])    # temp (degC)
R = np.array([765, 826, 873, 942, 1032]) # resistance (ohms)
plt.plot(T, R, '.')

popt, pcov = curve_fit(line, T, R)
m = popt[0]
b = popt[1]

Tfine = np.linspace(0., 100., 100)
plt.plot(Tfine, line(Tfine, m, b), 'r')
plt.title('R = {:.2f} T + {:.0f} '.format(m, b))
plt.xlabel('Temperature (T, degC)')
plt.ylabel('Resistance (R, ohms)')
plt.show()
```

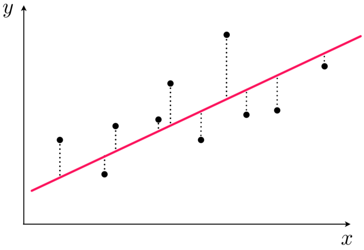
Code commentary

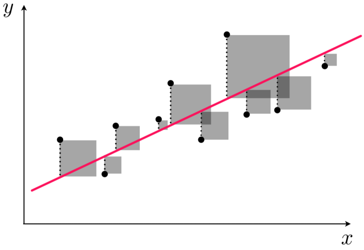
- xxx

2) Least squares fit

What makes the "best fit" line?

- residual: difference between an observed value, and the fitted value provided by a model
  - image on next page: residuals are dotted lines
- best fit in the least-squares sense minimizes the sum of squared residuals





- line fit to the data using the Least Squares framework
- aims at recovering the line that minimizes the total squared length of the dashed error bars
- Least Squares error can be thought of as the total area of the gray squares, having dashed error bars as sides (right panel)

Closed-form expression

Data is N pairs

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

Need to calculate constants $\mathbf{m}$ and $\mathbf{b}$ in best-fit straight line

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

Define

$$\bar{x} = \frac{x_0 + x_1 + \cdots + x_{N-1}}{N}$$

$$\bar{y} = \frac{y_0 + y_1 + \cdots + y_{N-1}}{N}$$

Closed-form expression (ctd.)

$$\mathbf{m} = \frac{(x_0 - \bar{x})(y_0 - \bar{y}) + (x_1 - \bar{x})(y_1 - \bar{y}) + \cdots + (x_{N-1} - \bar{x})(y_{N-1} - \bar{y})}{(x_0 - \bar{x})^2 + (x_1 - \bar{x})^2 + \cdots + (x_{N-1} - \bar{x})^2}$$

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

- back to resistance data, repeated here
- Python code for closed-form linear fit
- display $\mathbf{m}$ and $\mathbf{b}$ and show identical results from `curve_fit`

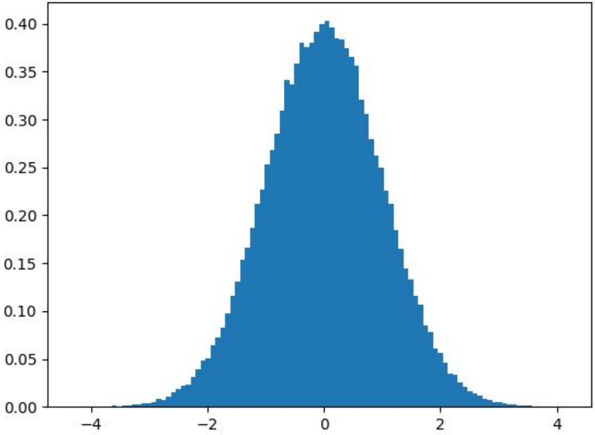
Recap

- given N data pairs, can calculate straight line of best fit using `curve_fit` or in closed-form
- give same results, ie: for best straight-line fit, closed-form expression exists
- this approach minimises squares of residuals
  - haven't proved this (beyond scope)
  - don't need to explicitly calculate residuals

3) Beyond straight-line fit

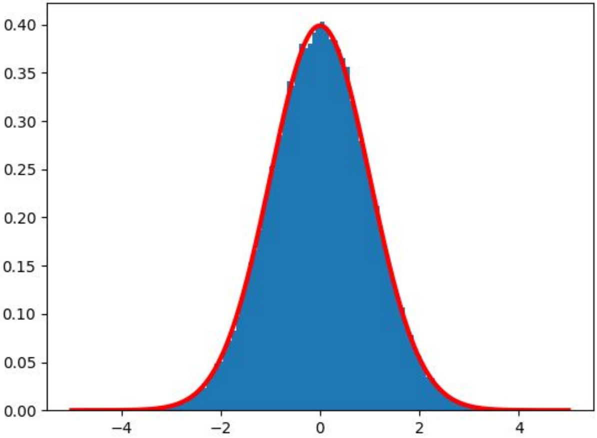
- xxx

Normalized histogram (area 1), 100 bins



- same histogram, except total area of rectangles is normalized to be 1

Normalized histogram with PDF

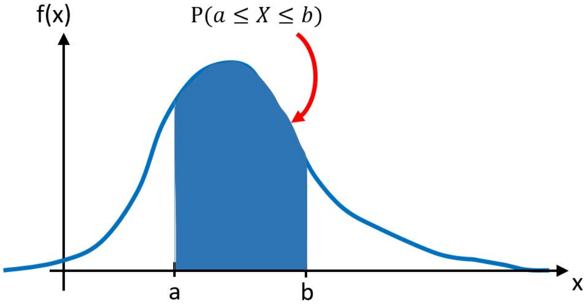


red curve is *probability density function (PDF)*

Probability density functions

If X is a random number drawn from a distribution with PDF $f(x)$, probability X takes a value in interval $[a, b]$ is

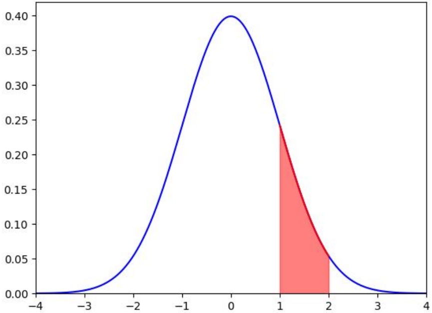
$$\boxed{\mathrm{P}(a \leq X \leq b) = \int_a^b f(x)dx}$$



Example

Use trapezoidal method to approximate $\mathrm{P}(1 \leq X \leq 2)$ when X is drawn from standard normal distribution

$$\mathrm{P}(1 \leq X \leq 2) = \frac{1}{\sqrt{2\pi}} \int_1^2 e^{-x^2/2} dx \approx \mathbf{0.1359}$$



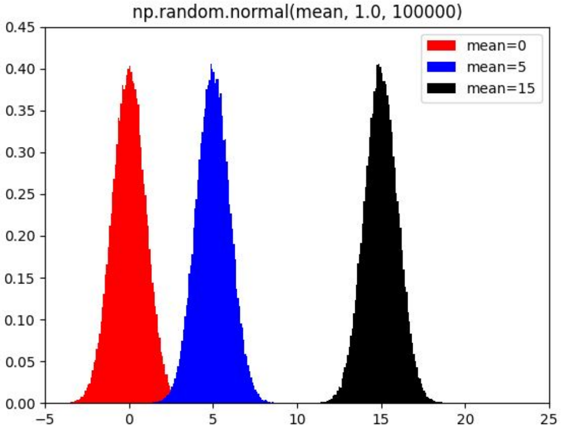
Extending the standard normal distribution

- standard normal is useful, but inflexible
- now **experimentally observe** impact of first two parameters in `normal()` function call

```
x = np.random.normal(mean, SD, size=N)
```

- impact of mean
  - shifts central (average) value of random numbers
  - left-right shift of PDF
- impact of SD
  - SD is short for "standard deviation"
  - controls spread of PDF around central value

Impact of *mean* parameter



- normalized histograms of normal random numbers with mean $= 0, 5$ and 15

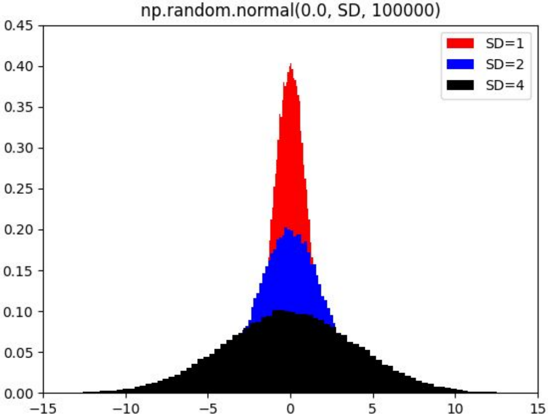
Python code: impact of mean

meandemo.py

```
1  # meandemo
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  N = 100000
6  np.random.seed(1)
7  x0 = np.random.normal(0.0, 1.0, size=N)
8  x5 = np.random.normal(5.0, 1.0, size=N)
9  x15 = np.random.normal(15.0, 1.0, size=N)
10
11 plt.hist(x0, 100, density=True, color='r', label='mean=0')
12 plt.hist(x5, 100, density=True, color='b', label='mean=5')
13 plt.hist(x15, 100, density=True, color='k', label='mean=15')
14
15 plt.title('np.random.normal(mean, 1.0, {})'.format(N))
16 plt.axis([-5, 25, 0, 0.45])
17 plt.legend()
18 plt.show()
```

- lines 7–13: generate random numbers, plot histograms

Impact of *SD* parameter



- normalized histograms of normal random numbers with SD $= 1, 2$ and 4

Python code: impact of SD

SDdemo.py

```
1 # SDdemo
2 import numpy as np
3 import matplotlib.pyplot as plt
4 
5 N = 100000
6 np.random.seed(1)
7 x0 = np.random.normal(0.0, 1.0, size=N)
8 x5 = np.random.normal(0.0, 2.0, size=N)
9 x15 = np.random.normal(0.0, 4.0, size=N)
10 
11 plt.hist(x0, 100, density=True, color='r', label='SD=1')
12 plt.hist(x5, 100, density=True, color='b', label='SD=2')
13 plt.hist(x15, 100, density=True, color='k', label='SD=4')
14 
15 plt.title('np.random.normal(0.0, SD, {})'.format(N))
16 plt.axis([-15, 15, 0, 0.45])
17 plt.legend()
18 plt.show()
```

- lines 7–13: generate random numbers, plot histograms

Normal PDF

Mathematical expression for normal PDF:

$$\boxed{f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}}$$

- $\mu =$ mean
- $\sigma =$ SD (standard deviation)
- what are you expected to do with the normally distributed random numbers in ENGG1003?
  1. call `np.random.normal()` to generate random numbers for specified μ and σ
  2. compute probability normally distributed random number X falls in range $[a, b]$ using numerical integration (eg: trapezoidal method)

Standard normal as special case

Important special case: $\mu = 0$ and $\sigma = 1$

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Key point: standard normal distribution has a mean of 0 and a standard deviation of 1

```
x = np.random.normal(0.0, 1.0, size=N)
```

- we saw this special case in Thursday week 9 lecture

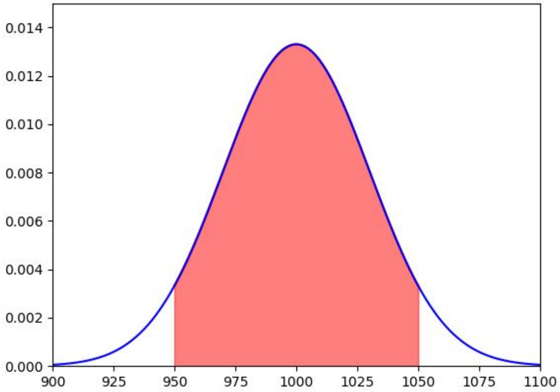
Application: resistor values

The distribution of resistor values (measured in ohms (Ω)) is observed to follow a normal distribution with mean $\mu = 1000$ and SD $\sigma = 30$

Write a Python script which:

1. plots the PDF of resistance values
2. uses numerical integration to show that $\approx 90\%$ of the resistor values fall in the range $[950, 1050]\ \Omega$

Distribution of resistance values



- blue curve is PDF: mean $= 1000$ and SD $= 30$
- red shaded area $(= 0.9)$ shows range $[950, 1050]$

Python code: resistor values

resistors.py

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    mu = 1000
    sigma = 30
    return 1/(sigma * np.sqrt(2 * np.pi)) * np.exp(-(x - mu)**2 / (2 * sigma**2 ))

def trapezoidal(f, a, b, n):
    h = (b - a) / n
    f_sum = 0
    for i in range(1, n, 1):
        x = a + i * h
        f_sum = f_sum + f(x)
    return h * (0.5 * f(a) + f_sum + 0.5 * f(b))
```

Python code: resistor values

resistors.py—continued

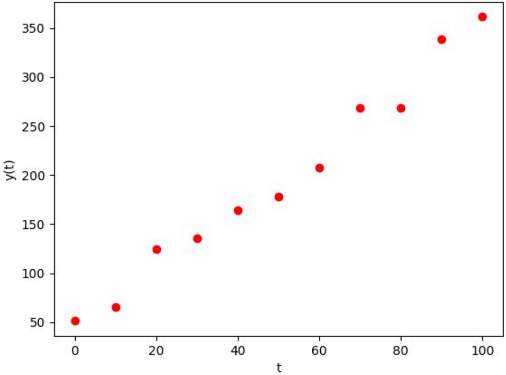
```
a = 950
b = 1050
prob_ab = trapezoidal(f, a, b, 100)
print('Probability resistance in range [{},{}] is: {:.2f} percent
    '.format(a, b, 100*prob_ab))

x = np.linspace(900, 1100, 1000)
xab = np.linspace(a, b, 100)

plt.plot(xab, f(xab), 'r')
plt.plot(x, f(x), 'b')  # standard normal pdf
plt.fill_between(xab, f(xab), color='r', alpha=0.5)  # alpha=
    transparency
plt.axis([900, 1100, 0, 0.015])
plt.show()
```

2) Fitting straight line to data

- **Aim:** construct a function that best fits a series of data points
  - simplest function is a *straight line*
- two common forms of *curve-fitting* in Engineering applications:
  1. *interpolation*
     - week 6, Monday lecture
  2. *regression*
     - today's lecture
- we now demonstrate both curve-fitting methods applied to the same dataset

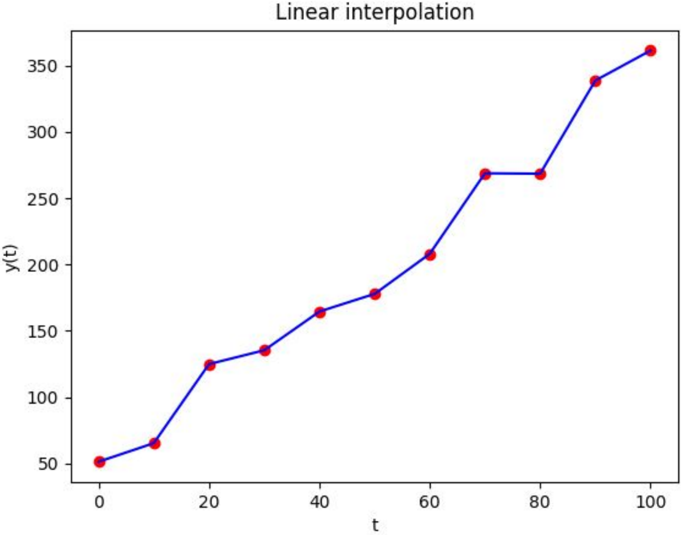
Curve-fitting dataset

`Week6Monday.py`

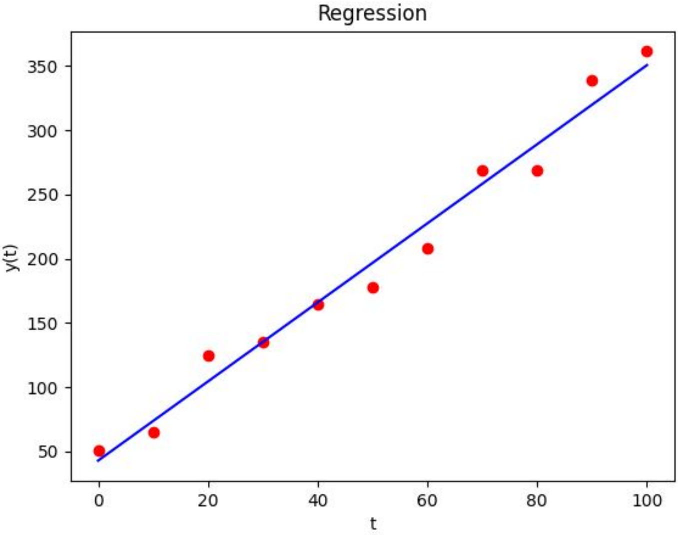


- 11 pairs of data points $(t_i, y_i), i = 0, 1, 2, \ldots, 10$
  $(0, 51.29), (10, 65.24), (20, 124.89), \ldots, (100, 361.32)$

Interpolation



Regression: straight-line fit



Interpolation vs. regression

- **interpolation:** joining the dots
  - obtain value of y at some intermediate point
  - week 6, Monday lecture
  - linear interpolation, cubic spline interpolation

- **regression:** fitting a straight line
  - when there's "too much data", simplify
  - here, simplifying to a straight line
  - today's lecture

- both interpolation & regression involve creating a function (blue line) from data (red dots)

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

$$y = mx + b$$

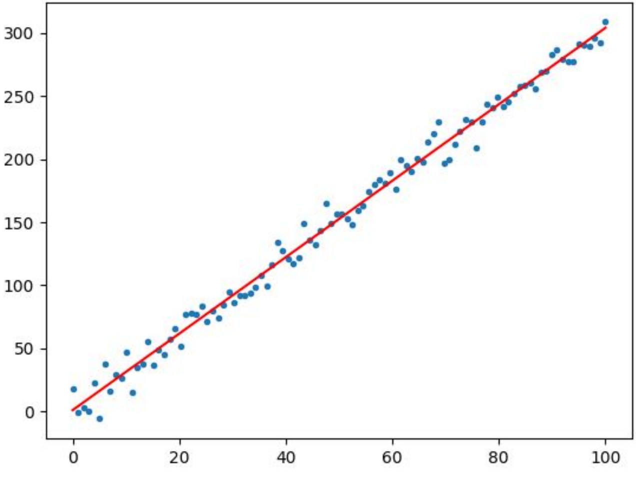
- in Python, we use `curve_fit()` function in `scipy.optimize` library to find m and b
  - may need `pip install scipy` in terminal

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

- ignore `pcov` returned by `curve_fit`

Line-fitting example

Output generated by `linefitdemo.py`



Python code: line-fitting

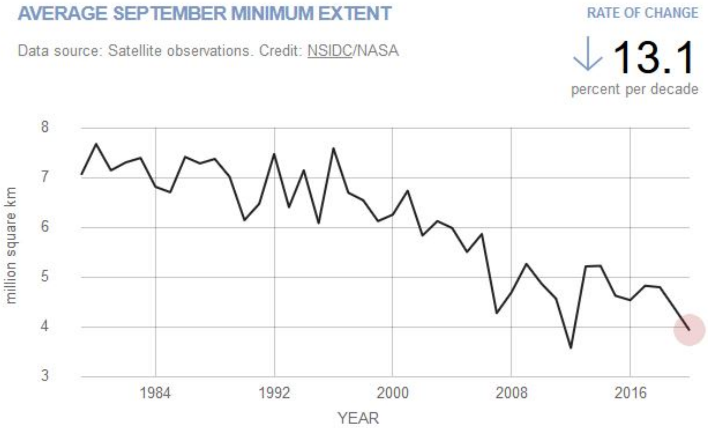
linefitdemo.py

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

def line(x, m, b):
    return m * x + b

np.random.seed(1)   # replicate results by fixing seed
x = np.linspace(0, 100, 100)
y = 3. * x + 2. + np.random.normal(0., 10., 100)
plt.plot(x, y, '.')

popt, pcov = curve_fit(line, x, y)
m = popt[0]
b = popt[1]
print('Straight-line gradient m = {:.2f}'.format(m))
print('Straight-line intercept b = {:.2f}'.format(b))

xfine = np.linspace(0., 100., 100)
plt.plot(xfine, line(xfine, m, b), 'r')
plt.show()
```

Python code: commentary

- lines 6–7: prepare to fit a straight line to (x, y) data
  - line equation $y = mx + b$
- lines 9–12: create and plot (x, y) data pairs
  - straight line (gradient 3 and y-intercept b) $+$ Gaussian noise ($\mu = 0, \sigma = 10$)
- lines 14–16: where the action happens!
  - `curve_fit()` function calculates m and b which provide best fit to (x, y) data
- lines 20–21: plot best-fit straight line

Application of line-fitting: sea-ice extent



https://climate.nasa.gov/vital-signs/arctic-sea-ice/

Fitting a straight line to sea-ice data

- graph shows average monthly Arctic sea ice extent each September since 1979, derived from satellite observations

  Aim: use straight-line fit to data to estimate when Arctic will be free of sea-ice

  - ie: when sea-ice extent is zero

- Key steps in solution
  1. fit straight line to data, using `scipy.optimize.curve_fit`
  2. line $y = mx + b$ intersects x-axis ($y = 0$) when $x = -b/m$

Python code: sea-ice extent

seaice.py

```
1  # seaiceextent
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.optimize import curve_fit
5
6  def line(x, m, b):
7      return m * x + b
8
9  # dataset: September sea-ice extent 1979-2020
10 # https://climate.nasa.gov/vital-signs/arctic-sea-ice/
11 year = np.arange(1979, 2021)
12 extent = np.array([7.05,7.67,7.14,7.3,7.39,6.81,6.7,7.41,
13        7.28,7.37,7.01,6.14, 6.47,7.47,6.4,7.14,6.08,
14        7.58,6.69,6.54,6.12,6.25,6.73,5.83,6.12,
15        5.98,5.5,5.86,4.27,4.69,5.26,4.87,4.56,
16        3.57,5.21,5.22,4.62,4.53,4.82,4.79,4.36,3.92])
```

- lines 6–7: prepare to fit a straight line to data
- lines 11–12: sea-ice extent dataset, 1979–2020

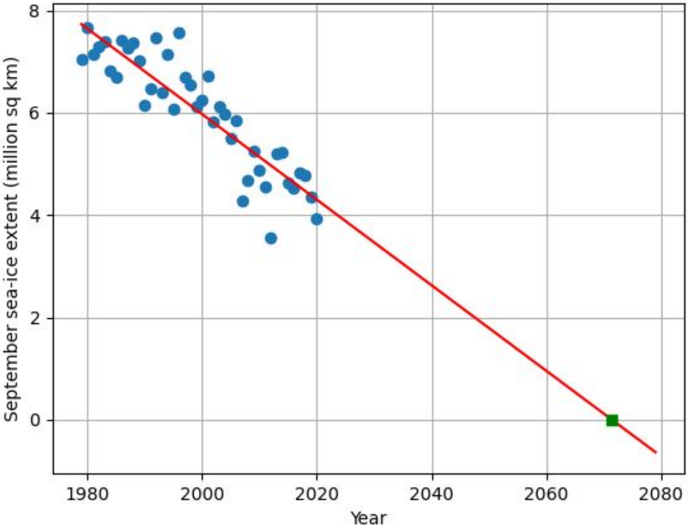
Python code: sea-ice extent

`seaice.py`—continued

```
1  popt, pcov = curve_fit(line, year, extent)
2  m = popt[0]          # gradient of best straight-line fit
3  b = popt[1]          # intercept
4
5  yearto2080 = np.arange(1979,2080)
6
7  print('extent(yr) = {:.3f}*year + {:.3f}'.format(m, b))
8  print('Estimate September sea-ice extent is 0 in year = {}'.
       format(int(-b/m)))
9  plt.plot(year, extent, 'o')
10 plt.plot(yearto2080, line(yearto2080, m, b), 'r')
11 plt.plot(-b/m,0,'gs')   # green square when ice extent is zero
12 plt.xlabel('Year')
13 plt.ylabel('September sea-ice extent (million sq km)')
14 plt.grid()
15 plt.show()
```

- lines 1–3: fit line to data: x = year, y = extent
- line 5: straight line fit over years 1979–2080
- line 8: line intersects horizontal axis at $-b/m$

Estimate Arctic sea-ice free in year 2071



Lecture summary

- Normal distribution
- Fitting straight line to data