

ENGG1003 - Monday Week 10

Normal distribution: extension and application
Fitting straight line to data

Steve Weller

University of Newcastle

10 May 2021

Last compiled: May 9, 2021 8:47pm +10:00

Lecture overview

1 Normal distribution

- ▶ extension of *standard* normal distribution (previous lecture)
- ▶ application

2 Fitting straight line to data

- ▶ using Python to fit a straight line to data
- ▶ application

1) Normal distributions

- **quick recap** of standard normal PDF: equation, interpretation, how to generate & plot histogram
- reiterate importance of normal distribution in applications
- but standard normal is inflexible

Recap: standard normal distribution

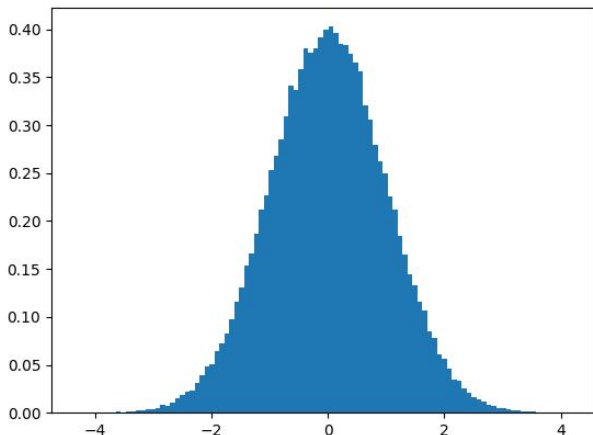
Standard normal probability density function (PDF):

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

- *standard* normal distribution is a special case of normal (Gaussian) distribution
- corresponds to parameters **0.0** and **1.0** in:

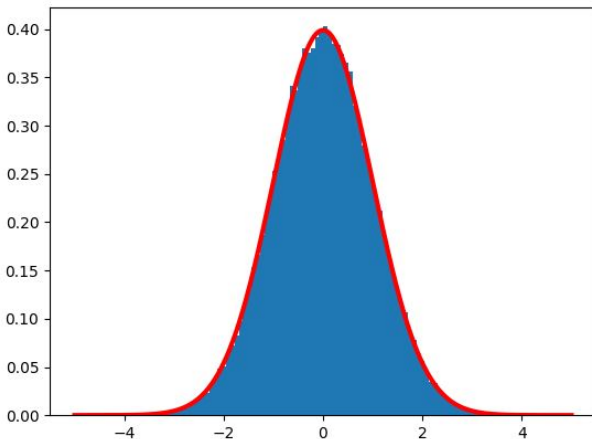
```
x = np.random.normal(0.0, 1.0, size=100000)
```

Normalized histogram (area 1), 100 bins



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

Normalized histogram with PDF

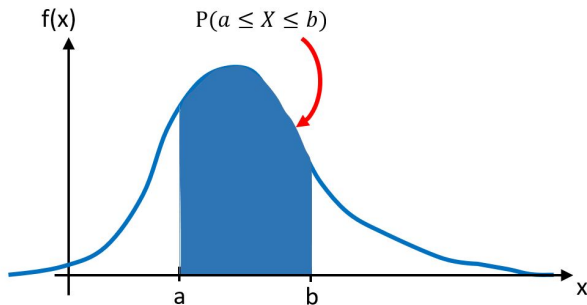


red curve is *probability density function (PDF)*

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

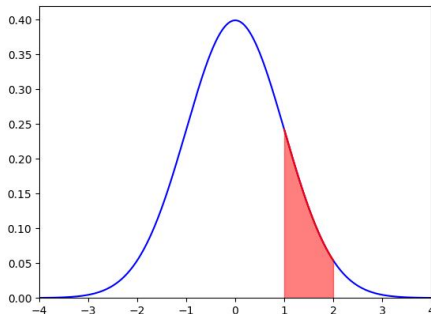
$$P(a \leq X \leq b) = \int_a^b f(x)dx$$



Example

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

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



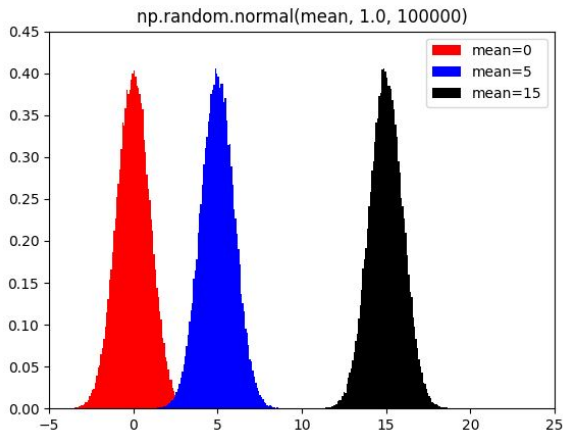
Extending the standard normal distribution

- 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

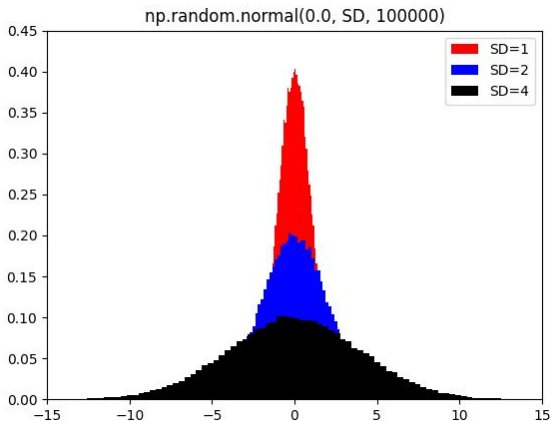
Python code: impact of mean

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:

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

- μ = mean
- σ = 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

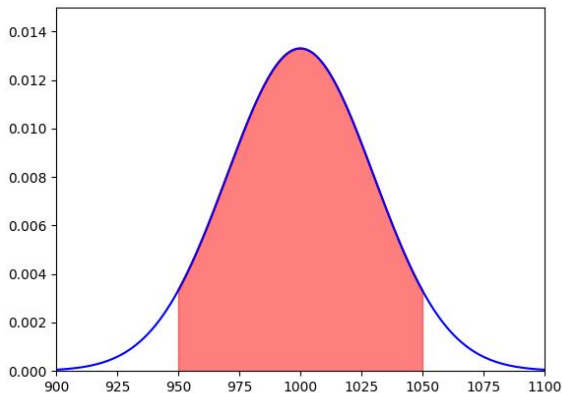
Application: resistor values

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

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(x):
5     mu = 1000
6     sigma = 30
7     return 1/(sigma * np.sqrt(2 * np.pi)) * np.exp(-(x - mu)**2 /
8         (2 * sigma**2 ))
9
10 def trapezoidal(f, a, b, n):
11     h = (b - a) / n
12     f_sum = 0
13     for i in range(1, n, 1):
14         x = a + i * h
15         f_sum = f_sum + f(x)
16     return h * (0.5 * f(a) + f_sum + 0.5 * f(b))
```

Python code: resistor values

resistors.py—continued

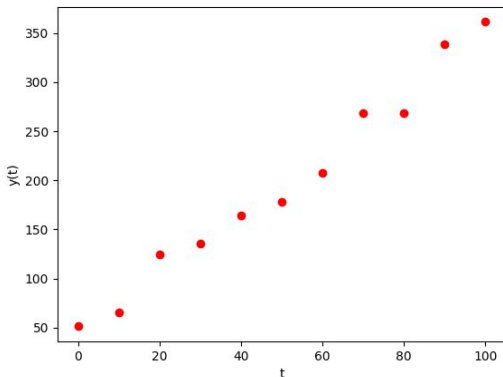
```
1 a = 950
2 b = 1050
3 prob_ab = trapezoidal(f, a, b, 100)
4 print('Probability resistance in range [{},{}] is: {:.2f} percent
      '.format(a, b, 100*prob_ab))
5
6 x = np.linspace(900, 1100, 1000)
7 xab = np.linspace(a, b, 100)
8
9 plt.plot(xab, f(xab), 'r')
10 plt.plot(x, f(x), 'b') # standard normal pdf
11 plt.fill_between(xab, f(xab), color='r', alpha=0.5) # alpha=
    transparency
12 plt.axis([900, 1100, 0, 0.015])
13 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

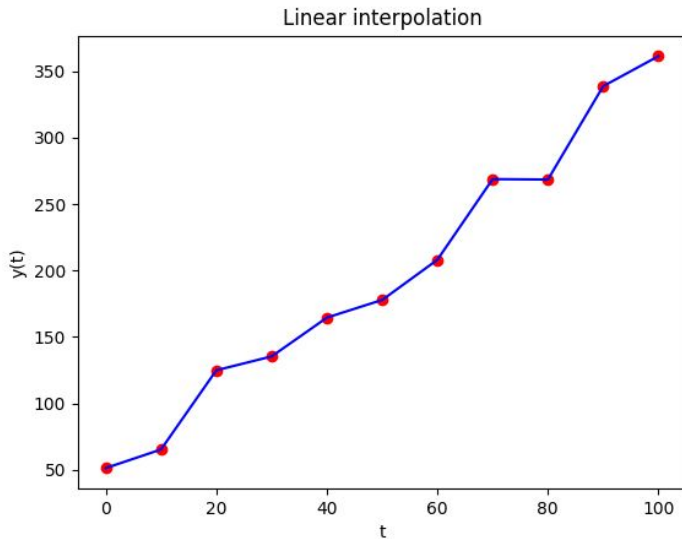
Curve-fitting dataset

Week6Monday.py

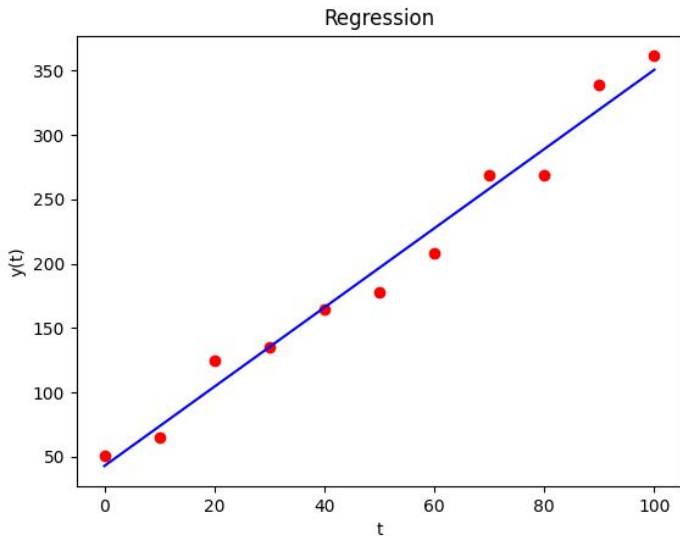


- 11 pairs of data points $(t_i, y_i), i = 0, 1, 2, \dots, 10$
 $(0, 51.29), (10, 65.24), (20, 124.89), \dots, (100, 361.32)$

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

$$y = mx + b$$

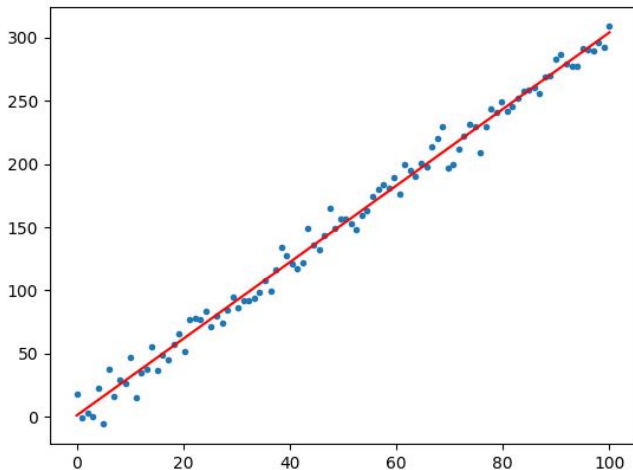
- in Python, we use `curve_fit()` function in `scipy.optimize` library to find m and b
 - ▶ may need `pip install scipy` in terminal

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

- ignore `pcov` returned by `curve_fit`

Line-fitting example

Output generated by `linefitdemo.py`



Python code: line-fitting

linefitdemo.py

```
1 # linefitdemo
2 import numpy as np
3 from scipy.optimize import curve_fit
4 import matplotlib.pyplot as plt
5
6 def line(x, m, b):
7     return m * x + b
8
9 np.random.seed(1) # replicate results by fixing seed
10 x = np.linspace(0, 100, 100)
11 y = 3. * x + 2. + np.random.normal(0., 10., 100)
12 plt.plot(x, y, '.')
13
14 popt, pcov = curve_fit(line, x, y)
15 m = popt[0]
16 b = popt[1]
17 print('Straight-line gradient m = {:.2f}'.format(m))
18 print('Straight-line intercept b = {:.2f}'.format(b))
19
20 xfine = np.linspace(0., 100., 100)
21 plt.plot(xfine, line(xfine, m, b), 'r')
22 plt.show()
```

Python code: commentary

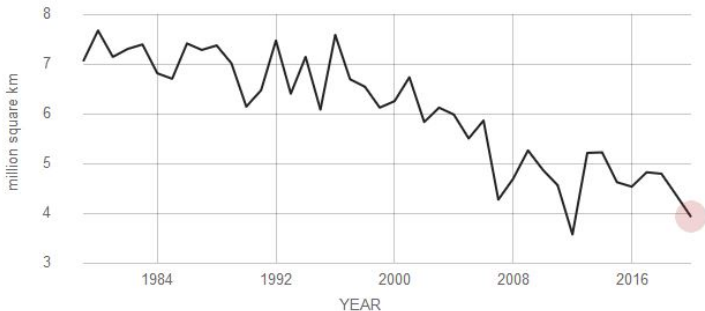
- lines 6–7: prepare to fit a straight line to (x, y) data
 - ▶ line equation $y = mx + b$
- lines 9–12: create and plot (x, y) data pairs
 - ▶ straight line (gradient 3 and y -intercept b)
+ Gaussian noise ($\mu = 0, \sigma = 10$)
- lines 14–16: where the action happens!
 - ▶ `curve_fit()` function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

AVERAGE SEPTEMBER MINIMUM EXTENT

Data source: Satellite observations. Credit: NSIDC/NASA

RATE OF CHANGE
↓ **13.1**
percent per decade



<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

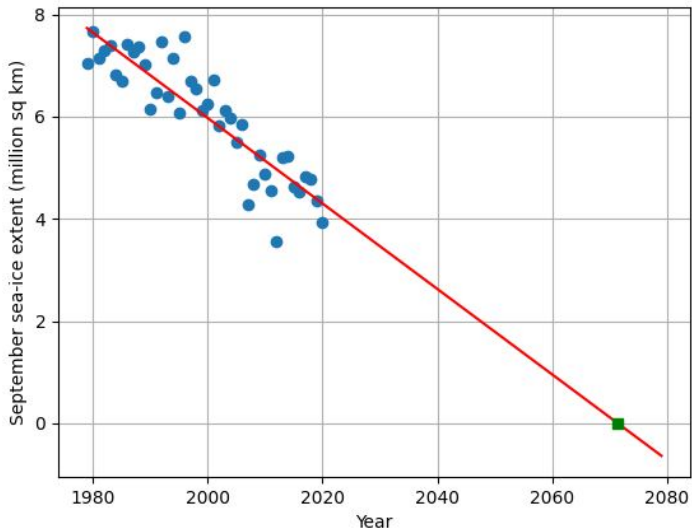
Python code: sea-ice extent

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 = {}'.
9       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: $x = \text{year}$, $y = \text{extent}$
- 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