ENGG1003 - Monday Week 10

Normal distribution: extensions and applications
Fitting straight line to data

Steve Weller

University of Newcastle

10 May 2021

Last compiled: May 8, 2021 7:19pm +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 distribtion in applications
- but standard normal is inflexible

Recap: standard normal distribution

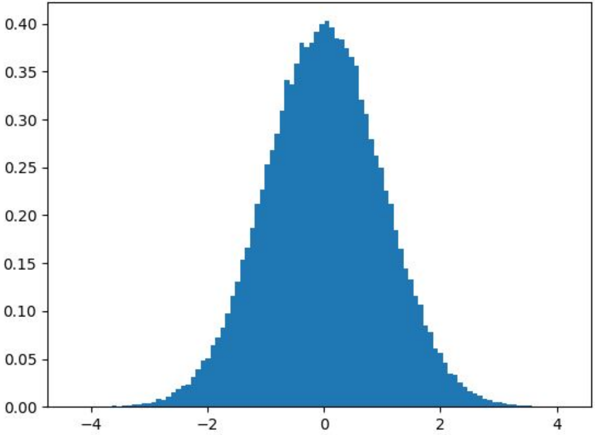
Standard normal probability density function:

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

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

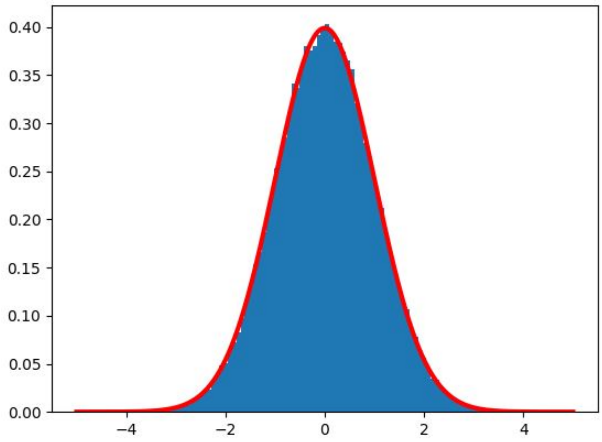
```
x = np.random.normal(0.0, 1.0, size=100000)
```

Normalized histogram (area 1), 100 bins



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

Normalized histogram with PDF

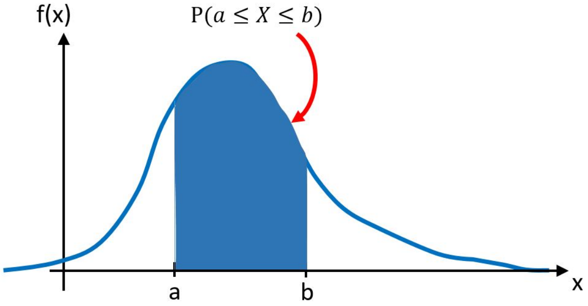


red curve is *probability density function (PDF)*

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

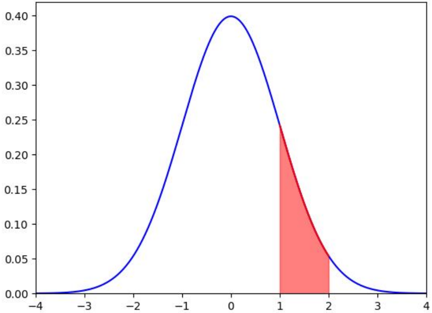
$$\boxed{\mathrm{P}(a \leq X \leq b) = \int_a^b f(x)dx}$$



Example

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

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

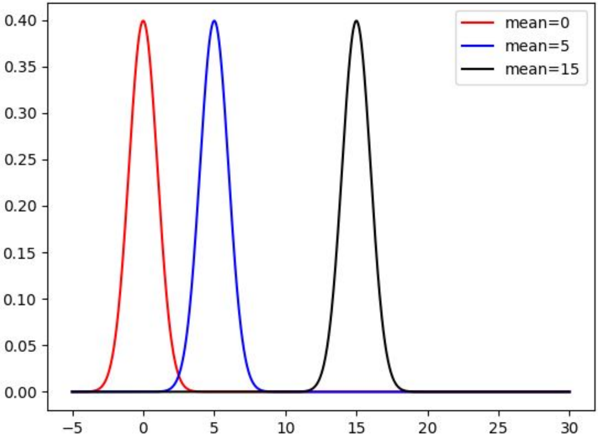


- reiterate importance of normal/Gaussian in applications
- but standard normal is inflexible
- now **experimentally observe** impact of first two parameters in `normal()` function call

Impact of mean

- shifts average (mean) value
- left-right shift of PDF
- image here: overlay PDFs for $\mu = 0, 5, 20$

Mean demo



- blah

Python code

meandemo.py

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

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

x = np.linspace(-5, 30, 1000)

plt.plot(x, f(x, 0, 1), color='r', label='mean=0')
plt.plot(x, f(x, 5, 1), 'b', label='mean=5')
plt.plot(x, f(x, 15, 1), 'k', label='mean=15')
plt.legend()
plt.show()
```

- blah

Impact of standard deviation σ

- shifts spread of PDF
- interpretation of “standard deviation”
- image here
- “most” of PDF within plus/minus 3 sigma of mean

“In statistics, the standard deviation is a measure of the amount of variation or dispersion of a set of values. A low standard deviation indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high standard deviation indicates that the values are spread out over a wider range. Standard deviation may be abbreviated SD, and is most commonly represented in mathematical texts and

Impact of stedev

- XXX

Stddev demo

- blah

Normal PDF

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

- mean μ
- standard deviation σ
- what are you expected to do with this PDF?
  1. call np.random.standard() to generate random numbers for specified μ and σ
  2. compute prob X in range $[a, b]$ using numerical integration (trapezoidal)

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

Application

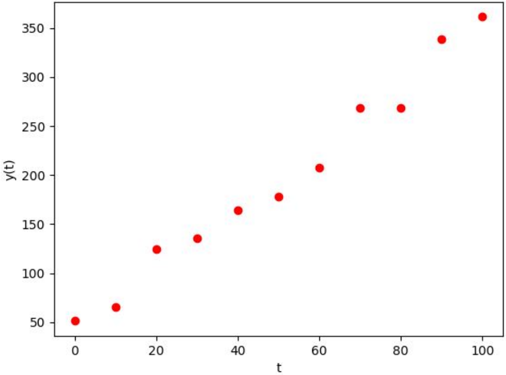
- xxx

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

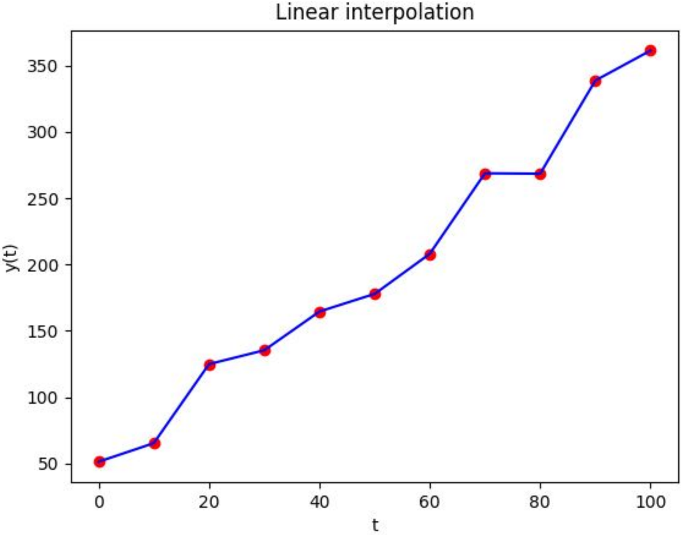
Curve-fitting dataset

`Week6Monday.py`

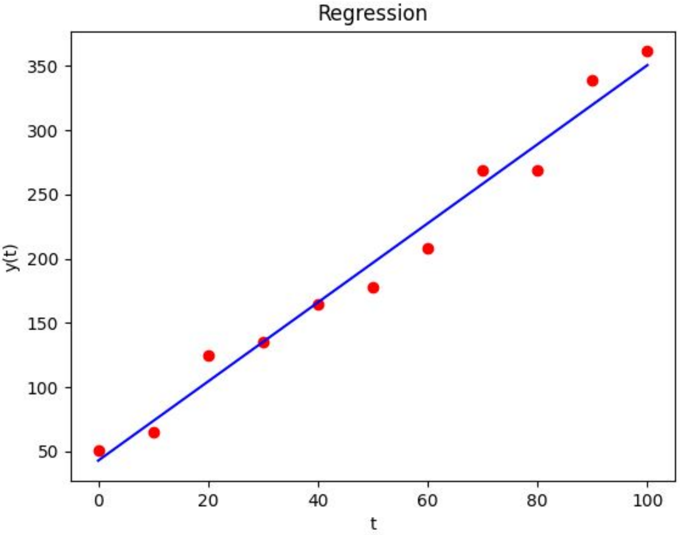


- 11 pairs of data points $(t_i, y_i), i = 0, 1, 2, \ldots, 10$
  $(0, 51.29), (10, 65.24), (20, 124.89), \ldots, (100, 361.32)$

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

$$y = mx + b$$

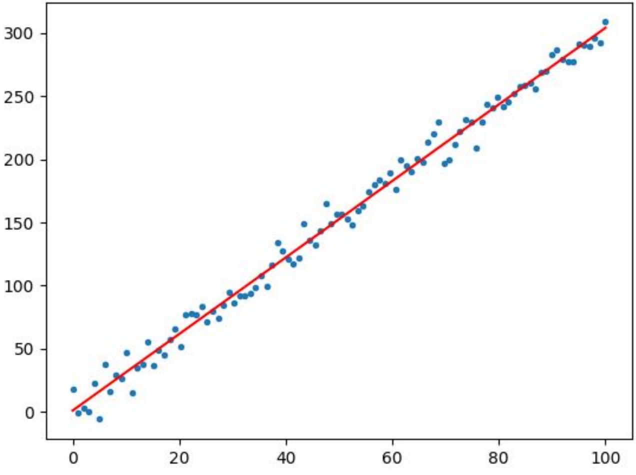
- in Python, we use `curve_fit()` function in `scipy.optimize` library to find m and b
  - may need `pip install scipy` in terminal

```
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

```python
# 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()
```

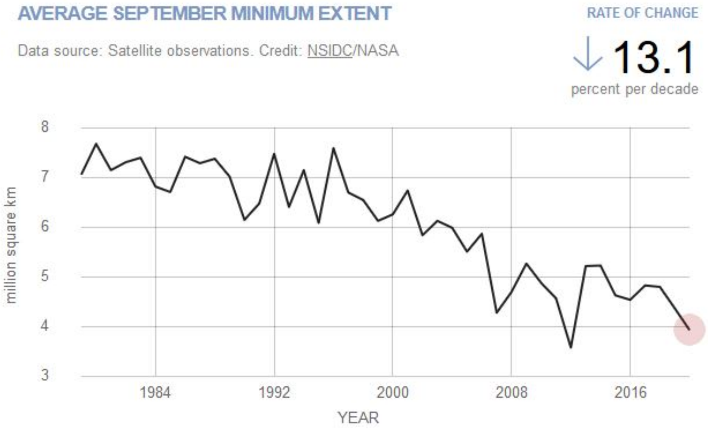
Python code: commentary

- 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

How does line-fitting work?

- least-squares
- maybe an image?
- extension to higher-order polynomials fitting

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

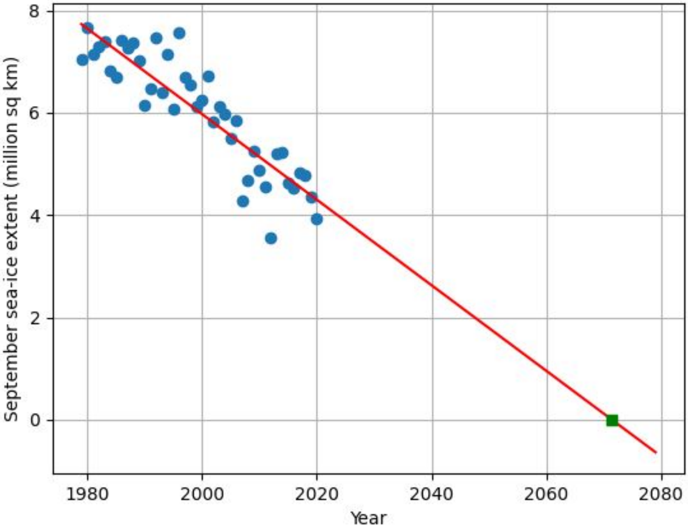
Python code: sea-ice extent

`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