Random Module

Thomas Schwarz, SJ

Why randomness

- Used in cryptography
  - e.g. session keys, challenges, …
- Used for simulation

Sources-of-Randomness

- True randomness is difficult
  - Hardware Random Number Generators
    - Shot Noise: a lamp shines on a photo-diode. The photons create noise in the circuit because of the uncertainty principle
    - Radioactive decay
    - Photons travelling through a semi-transparent mirror
    - Thermal noise from a resistor
    - Atmospheric noise detected by a radio receiver

Sources-of-Randomness

- Hardware Random Number Generation:
  - Translation into a given random distribution (e.g. a bit stream without correlation and 50% ones) is difficult
  - Software can be used to “extract randomness”

Sources-of-Randomness

- System data
    - Has a bad name because its randomness was overestimated in a version of Secure Socket Layer

Pseudo-Randomness

- Pseudo-random generator:
  - Produce an output stream that is statistically undistinguishable from true random data
  - Usually based on a seed
    - The same seed generates the same pseudo-random numbers
  - Use some mathematics to convert the output stream to one having any random distribution

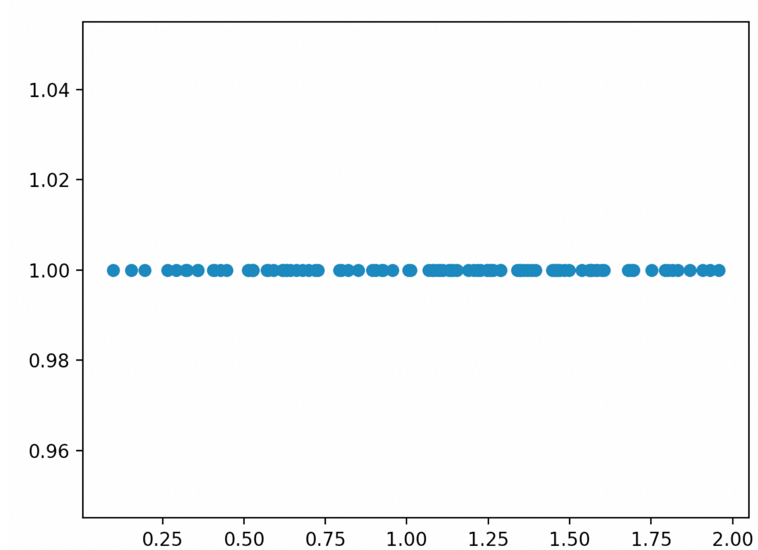
Random Module

- Imported via
  - `import random as rd`
- The abbreviation is not quite as generally used as others

Random Module

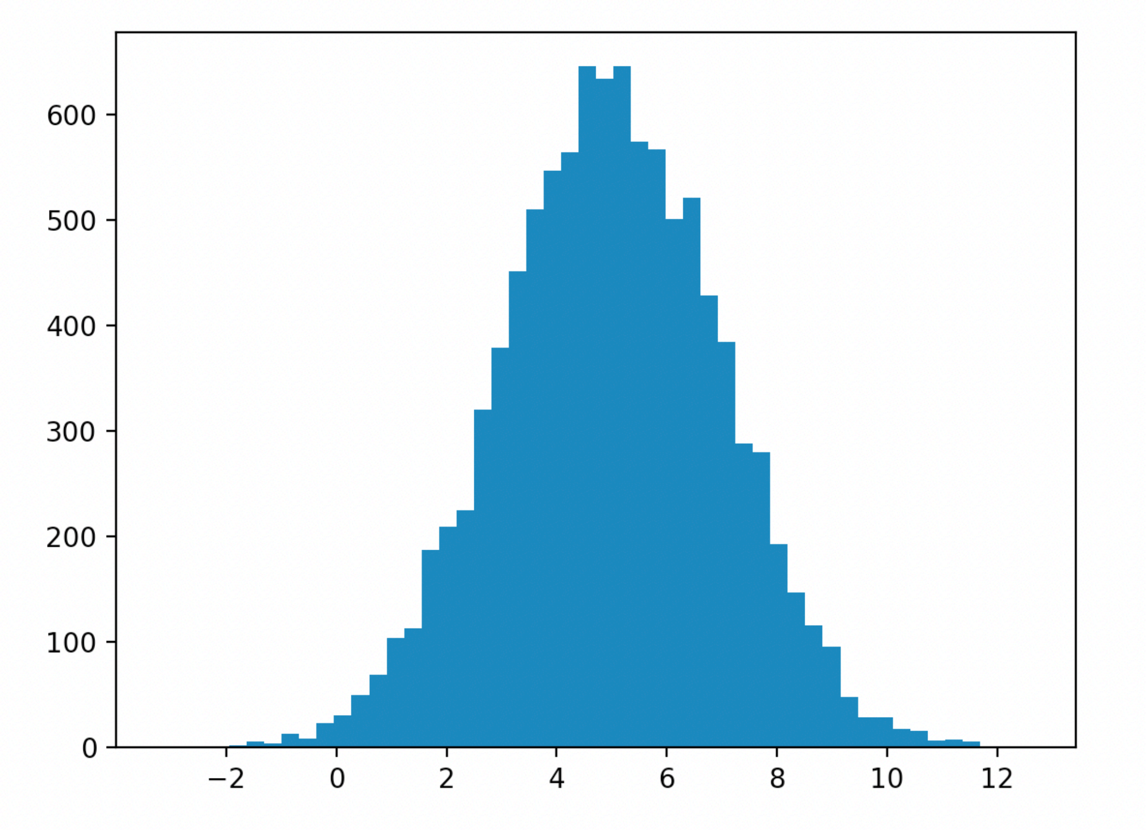
- How to get random numbers:
  - `rd.random( )` gives a random floating point number between 0 and 1
  - 100 pts with rd.random between 0 and 1

  - Generalized by `rd.uniform(a,b)`
    - 100 pts with rd.uniform(0,2)



Random Module

- `rd.normalvariate(mu, sigma)` gives normally distributed values
    - Centered around mu
    - “Average” distance from center of sigma



Random Module

- Law of large numbers:
  - Perform an experiment with an outcome $X \in \mathbb{R}$ independently n times, $n \to \infty$
    - mean $\bar{X}_n = \dfrac{X_1 + X_2 + \ldots + X_n}{n}$ looks more and more normally distributed
  - Mathematically:
    - $\sqrt{n}(\bar{X}_n - \mu) \to \mathcal{N}(0, \sigma^2)$ in distribution

Random Module

- Example: Coin toss with a fair coin:
    - Number of “heads” after *n* tosses is x with probability
        - $\dfrac{\binom{n}{r}}{2^n}$
            - (number of ways of arraigning r heads over the 2^n possible arraignments)

Random Module

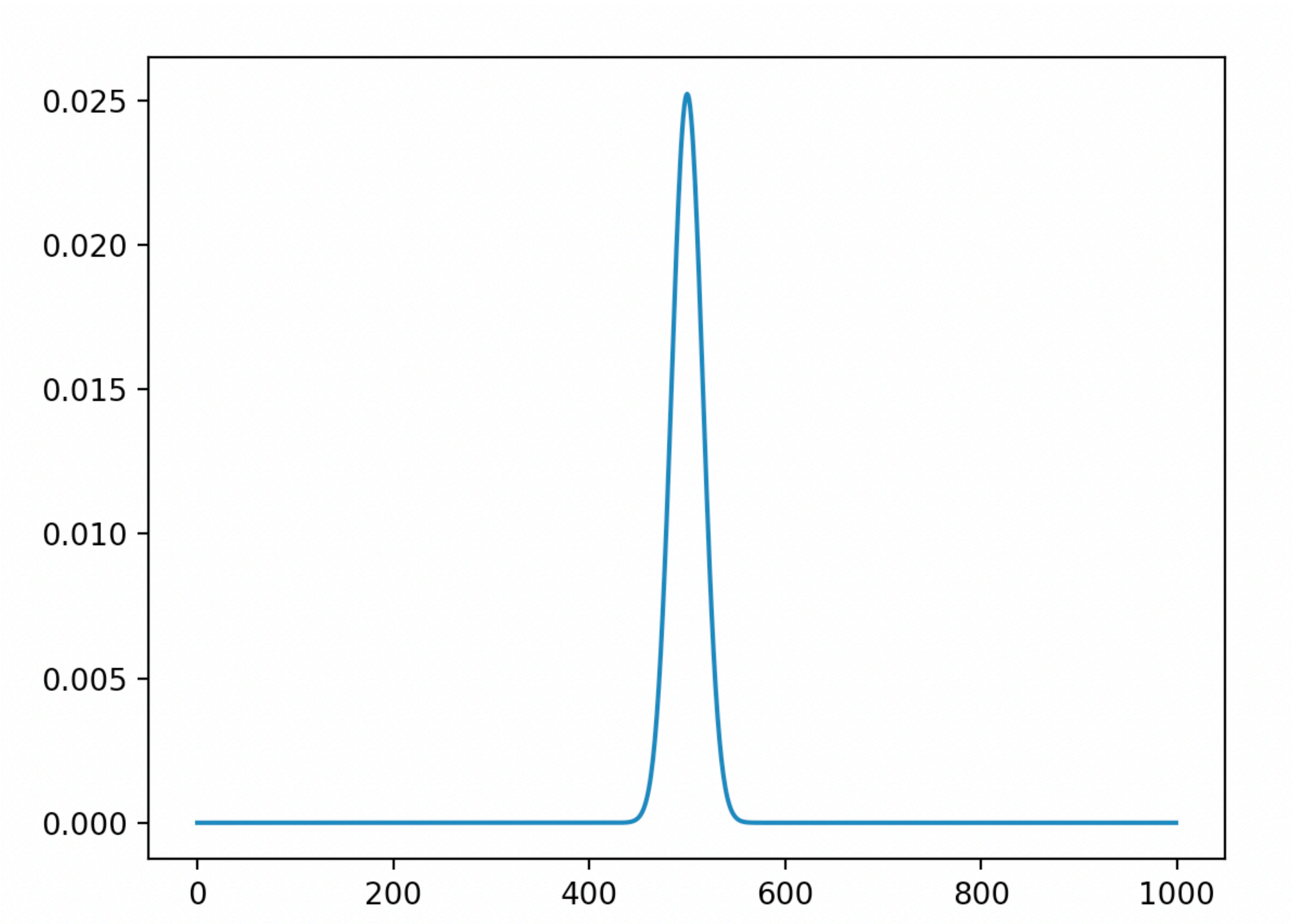
- We can use math.comb to calculate the probability

```
import math
import matplotlib.pyplot as plt

def prob_coin_toss(nn, i):
    return math.comb(nn,i)/2**nn

nn=1000
plt.plot(range(nn), [prob_coin_toss(nn,i) for i in range(nn)])
plt.show()
```

Random Module



Random Module

- Or we can approximate the probability with the normal distribution with mean $\mu = \frac{nn}{2}$ and $\sigma = \sqrt{\frac{nn}{4}}$

- Formula is

- $P(i \text{ heads}) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp(-\frac{1}{2}(\frac{i-\mu}{\sigma})^2)$

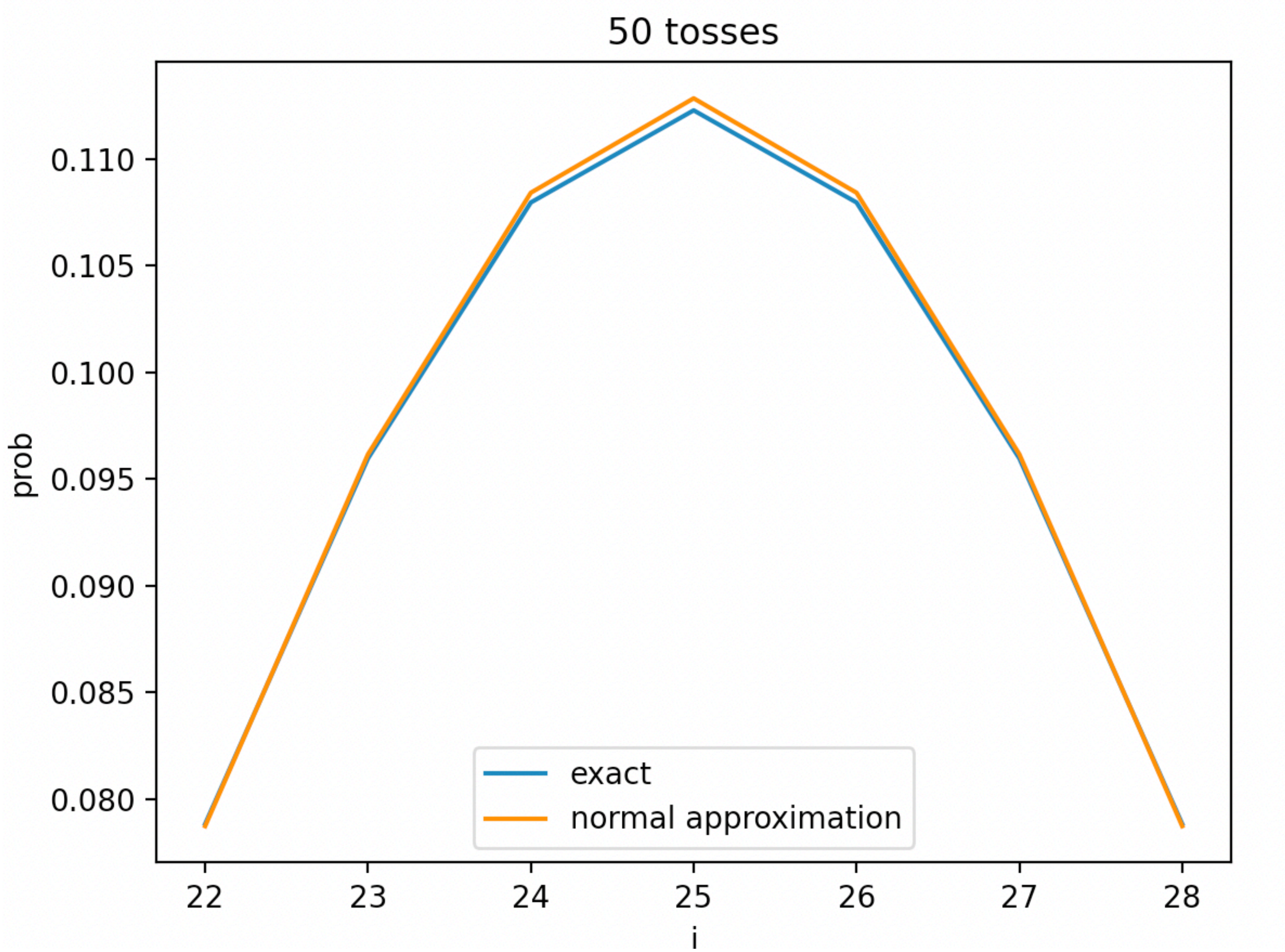
Random Module

- Probability according to normal distribution

```
def normal_pdf(nn, i):
    sigma = math.sqrt(nn/4)
    mu = nn/2
    factor = 1/math.sqrt(2*math.pi*nn/4)
    exponent = -0.5*(i-mu)**2/sigma**2
    return factor * math.exp(exponent)
```

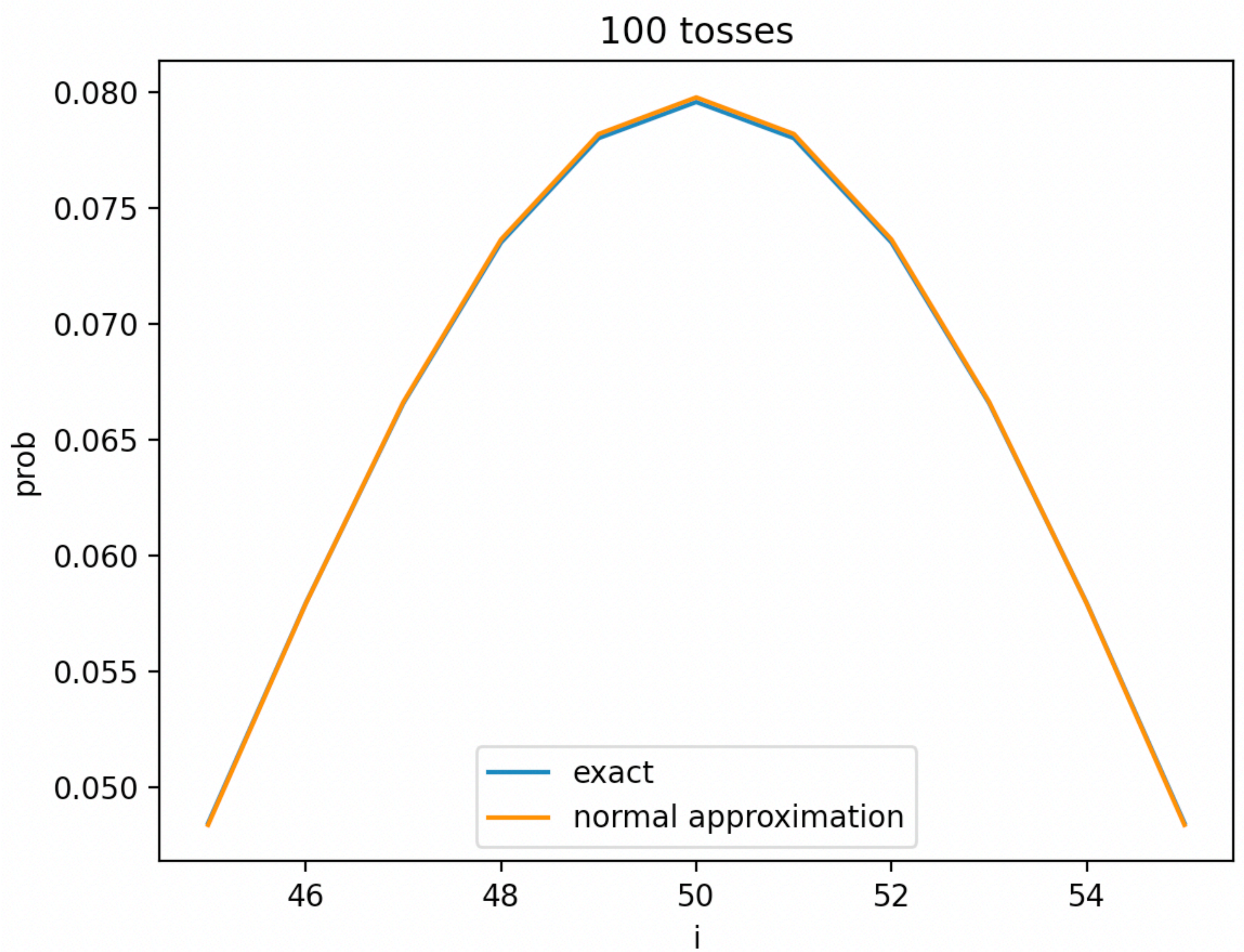
Random Module

- Coin toss experiment

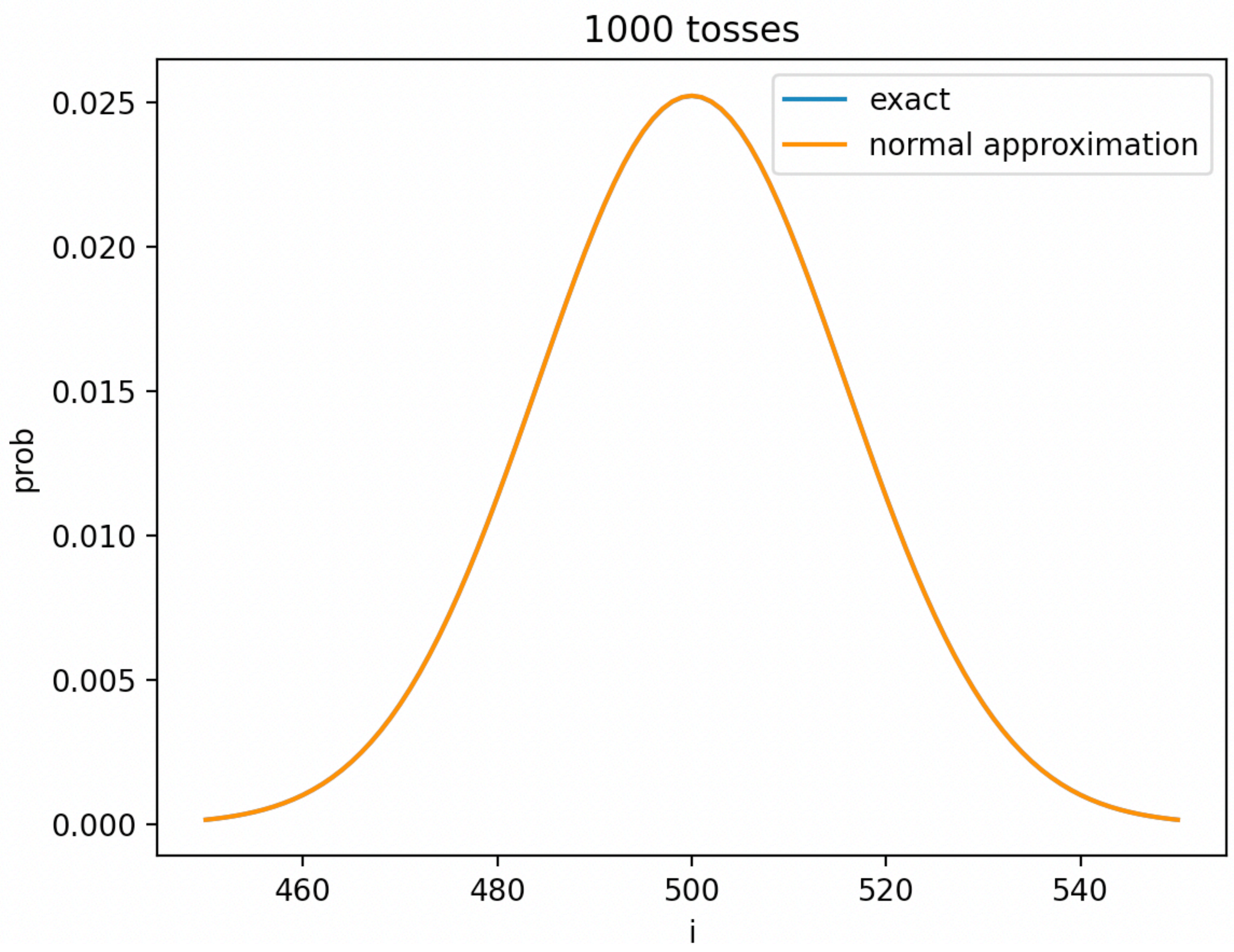


Random Module

- Coin toss experiment



Random Module



Random Module

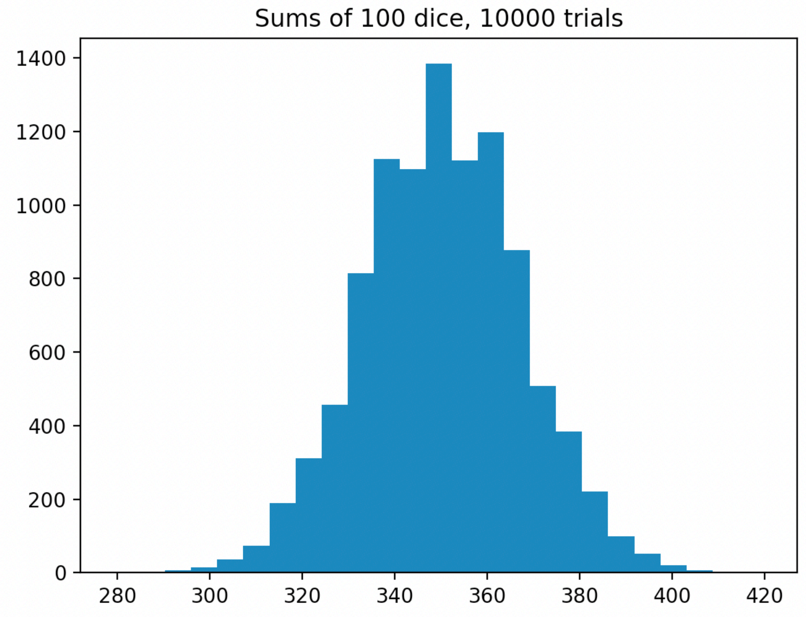
- This is why the normal distribution is so important:
  - In social sciences, a parameter with unknown probability distribution is often replaced with a normal distribution

Random Module

- Python uses a sophisticated pseudo-random number generator
  - There is a trade-off between speed and unpredictability
    - Python random prefers speed and the results are not usable for cryptographic purposes.
  - For repeatability, you set the seed of the pseudo-random generator:
    - `random.seed(12345)`
    - The argument can also be a string:
    - `rd.seed("India expect all men to do their duty.”)`

Random Module

- Selection:
  - `rd.randint(a,b)` A random number (with equal probability) between a and b, both ends included
  - Example:
    - The sum of throwing hundred dice



Random Module

```
def sum_of_dice(n):
    suma = 0
    for _ in range(n):
        suma += rd.randint(1,6)
    return suma
```

Random Module

- To get statistics, we place them into a list
    - Using list-comprehension, which we still have to learn

```
def stats_sum_of_dice(n):
    return  [sum_of_dice(100) for _ in range(10000)]
```

Random Module

- And now we use the histogram function in matplotlib.plt

```
plt.hist(stats_sum_of_dice(100), bins=25)
plt.title("Sums of 100 dice, 10000 trials")
plt.show()
```

Random Module

- How often does die A beat die B?
  - Analytical answer:
    - In about 1/6 of all cases, there is equality
    - Die A beats die B in half the remaining cases
    - I.e. with probability 2.5/6

Random Module

- How often does die A beat die B?
  - Experimental answer:
    - Let's repeat this a million times and count

```
def a_beats_b(nn):
    count = 0
    for _ in range(nn):
        a = rd.randint(1,6)
        b = rd.randint(1,6)
        if a>b:
            count += 1
    return count/nn
```

Random Module

- How often does die A beat die B?
  - Experimental answer:
    - Let's repeat this a million times and count

```
>>> a_beats_b(1000000)
0.41662
>>> 2.5/6
0.4166666666666667
```

    - Close to the real value

Random Module

- `rd.choice(a_list)` selects a random element from a list (or a sequence type like a string)

```
>>> rd.choice("hello world")
'e'
```

Random Module

- Example: The random Python insult generator

```
list_adj = ['wart-covered', 'clumpsy', 'despairing', 'ignominous']
list_adj1 = ['Belgian', 'French', 'Flemish', 'Kraut', 'Frog',
'Cheeseburgher']
list_ani = ['striped badger', 'wart-hog', 'pot-bellied pig’]

def insult():
    return f'''You son of a {rd.choice(list_adj)} {rd.choice(list_adj1)}
{rd.choice(list_ani)}, I cough in your general direction!'''
```

Random Module

- To shuffle a list, use:
  - `rd.shuffle(a_list)`