

Python implementations for 35B402

Pieter Kleer

Table of contents

1	Probability and statistics	2
1.1	Simulation-based inference	2
1.1.1	Monte Carlo Estimation of π	2

Chapter 1

Probability and statistics

In this chapter we will show various Python implementations of concepts we have seen in the course Probability and statistics (35B402).

1.1 Simulation-based inference

In this section we illustrate some of the examples seen in Chapter 7 of the reader of this course.

1.1.1 Monte Carlo Estimation of π

Recall that if we have n independent pairs (X_i, Y_i) of random variables uniformly from $[-1, 1]^2$ for $i = 1, \dots, n$ and count which fraction falls in the unit disk $D = \{(x, y) : x^2 + y^2 \leq 1\}$, then a quarter of this estimated fraction forms an approximation of π .

That is, the estimator is given by

$$\hat{\pi}_n = 4 \cdot \frac{1}{n} \sum_{i=1}^n \mathbb{I}\{X_i^2 + Y_i^2 \leq 1\}.$$

We can generate uniformly random numbers using the `uniform(a,b)` function from the `random` package within NumPy. This function takes as input two numbers a and b that specify the interval $[a, b]$ from which the number is generated uniformly at random. The syntax for generating such a number is `np.random.uniform(a,b)`.

```
# Import NumPy
import numpy as np

# Random number from interval [-1,1]
x = np.random.uniform(-1,1)

# Print number
print(x)
```

0.8844424212352004

If you rerun the command above (e.g., in a Jupyter Notebook) you will see a different number everytime. Let us execute the same code again to illustrate this.

```
# Import NumPy
import numpy as np

# Random number from interval [-1,1]
x = np.random.uniform(-1,1)

# Print number
print(x)
```

0.39427236481154937

To compute $\hat{\pi}_n$ we can use a for-loop approach, where we repeatedly generate two random numbers (x, y) , representing X_i and Y_i , in $[-1, 1]$ and check whether $x^2 + y^2 \leq 1$.

In every iteration of the for-loop, we increase an, initially equal to 0, variable `count` by 1 if the point generated in that iteration lies in the unit disk D . This will then give us the sum $\sum_{i=1}^n \mathbb{I}\{X_i^2 + Y_i^2 \leq 1\}$ for a collection of sampled values (X_i, Y_i) for $i = 1, \dots, n$.

To get the final estimate we multiply the expression with $4/n$.

```
# Number of samples
n_samples = 1000

# Initial count
count = 0

# Repeatedly sample value and check if contained in unit disk
for i in range(n_samples): # Index i is actually not used
    x = np.random.uniform(-1,1)
    y = np.random.uniform(-1,1)
    if x**2 + y**2 <= 1:
        count = count + 1

# Formula for estimate
pi_estimate = 4 * (1/n_samples) * count

# Print estimate
print(pi_estimate)
```

3.168

We can also write this code more neatly, by introducing a Python function that takes as input a number of samples `n_samples` and returns the estimate $\hat{\pi}_n$. Below we give a complete self-contained piece of code for this.

```

import numpy as np

def monte_carlo_pi(n_samples):
    count = 0
    for i in range(n_samples):
        x = np.random.uniform(-1,1)
        y = np.random.uniform(-1,1)
        if x**2 + y**2 <= 1:
            count = count + 1

    pi_estimate = 4 * (1/n_samples) * count
    return pi_estimate

# Example usage
n_samples = 100000
pi_hat = monte_carlo_pi(n_samples)
print(f"Estimated pi with {n_samples} samples: {pi_hat}")

```

Estimated pi with 100000 samples: 3.1594800000000003

To see how the approximation progresses as the number of samples increases, play around with the following animation, that you can also open in full screen by clicking [here](#) (does not work in PDF).