A small library for simulating and analysing sums of fair dice

Group N

Abstract

We present `dice`, a small Python library for simulating sums of fair s-sided dice and comparing the empirical distribution with the exact theoretical distribution. We use the standard library throughout: `random` for the simulation, `statistics` for the summary numbers, and `fractions.Fraction` for the exact distribution. We recover the textbook values $P(s = 2) = 1/36$, $P(s = 7) = 1/6$ and $P(s = 12) = 1/36$ for two six-sided dice, and we show that the empirical distribution from 10000 simulated rolls agrees with the theoretical values to within sampling error. The library follows the Diataxis framework for documentation [2] and the conventions of the open-source textbook *Python for Mathematics* [1].

1 Introduction

The distribution of the sum of n fair s-sided dice is a standard probability example. For small n the distribution can be written out by hand: for $n = 2$ and $s = 6$ the well-known triangular distribution has $P(s = 2) = 1/36$, $P(s = 7) = 6/36$, and the rest of the values fall symmetrically between. For larger n the closed-form expression in terms of binomial coefficients is awkward to state and easy to get wrong.

We wanted a small library that lets us both simulate the sum and compute the exact distribution, with the latter as a sanity check on the former. The standard library provides everything we need: `random` for the simulation, `statistics` for the summary numbers, and `fractions.Fraction` for the rationals.

We present a five-function library, `dice`; we recover the textbook values for two six-sided dice; and we show that the empirical distribution from 10000 rolls is within a comfortable margin of the theoretical values.

2 The library

The library lives in the single module `dice.py` and follows the probability chapter of *Python for Mathematics* [1].

`roll(number_of_dice, sides, seed)` returns a tuple of face values, each drawn uniformly from $\{1, 2, \ldots, s\}$. An optional "seed" argument lets us reproduce a roll.

`simulate_sums(number_of_dice, number_of_rolls, sides, seed)` returns a list of sums, one per roll.

`empirical_distribution(samples)` returns a dictionary mapping each observed value to its relative frequency.

`theoretical_distribution(number_of_dice, sides)` computes the exact distribution by repeated convolution of the single-die distribution. We use `fractions.Fraction` so that the probabilities are exact rationals.

`summary_statistics(samples)` returns the mean, median, and sample standard deviation, using `statistics.mean`, `statistics.median` and `statistics.stdev`.

3 Worked example

We illustrate the library on the sum of two six-sided dice. The theoretical distribution is the triangular distribution shown in Table 1. We then simulate 10000 rolls with seed 0; the empirical probability of rolling a 7 is 0.16∼0.17, which lies inside the interval $[0.13, 0.20]$ that we use in the doctest.

Sum	2	3	4	5	6	7	8	9	10	11	12
Prob. (×36)	1	2	3	4	5	6	5	4	3	2	1

Table 1: **The exact distribution of the sum of two six-sided dice.** The probability of each total is shown as a multiple of 1/36. The mode is at 7, with probability $6/36 = 1/6$. The library returns these values exactly using `fractions.Fraction`.

4 Discussion

The probability chapter of *Python for Mathematics* reaches for `sympy.Rational`, which would also work. We chose `fractions.Fraction` because the rest of the library is standard-library only, and bringing in `sympy` would be disproportionate for the small amount of exact arithmetic we need.

We only handle fair dice. A natural extension would be to accept a custom probability mass function on the faces of each die, so that we can model loaded dice or dice with non-standard faces. The convolution code already accepts a dictionary, so this extension would be small.

5 Conclusion

We have presented `dice`, a small library for simulating sums of fair dice and comparing them against the exact distribution. The library recovers the textbook values for two six-sided dice and the empirical distribution from a seeded simulation agrees with them to within sampling error.

References

[1] Vincent Knight. Python for Mathematics, 2024.

[2] Daniele Procida. Diátaxis: a systematic framework for technical documentation authoring, 2017.