A small library for the binomial distribution

Group N

Abstract

We present `binomial`, a small Python library for the binomial distribution $X \sim \mathrm{Bin}(n, p)$. The library exposes five short functions: the probability mass function, the cumulative distribution function, the mean, the variance, and a simulator. The binomial coefficient is computed exactly with `scipy.special.comb` [4]. We verify the library on the fair-coin case $n = 10$, $p = 1/2$: the PMF and CDF agree with the textbook values, the PMF sums to one, and the empirical mean of 5000 simulated draws of $\mathrm{Bin}(20, 1/2)$ lies within 0.3 of the theoretical mean of 10. The library follows the Diataxis framework for documentation [2] and the combinatorics and probability chapters of the open-source textbook *Python for Mathematics* [1].

1 Introduction

The binomial distribution describes the number of successes in n independent Bernoulli trials, each with success probability $p \in [0, 1]$. Writing $X \sim \mathrm{Bin}(n, p)$, the probability mass function and the cumulative distribution function are

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad P(X \le k) = \sum_{j=0}^{k} \binom{n}{j} p^j (1-p)^{n-j},$$

and the mean and variance are $\mathbb{E}[X] = np$ and $\mathrm{Var}(X) = np(1-p)$ [3].

The binomial distribution is built into `scipy.stats.binom` [4], which provides every function we implement here. We wanted a small library that we could read and understand line by line, with the binomial coefficient computed exactly and the simulator written out as Bernoulli trials.

We present a five-function library, `binomial`; we verify it on the fair-coin case $n = 10$, $p = 1/2$; and we show that the empirical mean of 5000 simulated draws of $\mathrm{Bin}(20, 1/2)$ is close to the theoretical mean.

2 The library

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

`pmf(number_of_successes, number_of_trials, success_probability)` returns $\binom{n}{k} p^k (1-p)^{n-k}$. The binomial coefficient is computed exactly with `scipy.special.comb(..., exact=True)` [4]; we then multiply by the probabilities.

`cdf(number_of_successes, number_of_trials, success_probability)` returns the partial sum of the PMF up to k. Summing eleven floats for a small n is cheap, so we did not reach for `scipy.stats.binom.cdf`.

`mean(number_of_trials, success_probability)` and `variance(number_of_trials, success_probability)` return np and $np(1-p)$ respectively.

`simulate(number_of_trials, success_probability, number_of_simulations, seed)` draws each binomial sample as the number of n independent uniform draws that fall below p. The "seed" argument lets us reproduce a simulation.

3 Worked example

We illustrate the library on the fair-coin case $n = 10$, $p = 1/2$. The PMF and CDF agree with the textbook values shown in Table 1. The PMF sums to one and the CDF reaches one at $k = n$.

k	$P(X = k)$	$P(X \le k)$
0	1/1024	1/1024
1	10/1024	11/1024
2	45/1024	56/1024
3	120/1024	176/1024
4	210/1024	386/1024
5	252/1024	638/1024

Table 1: **The PMF and CDF of $X \sim \mathbf{Bin}(10, 1/2)$.** The mode is at $k = 5$, with probability 252/1024. The library returns these values to floating-point precision, with the binomial coefficient computed exactly.

For $\mathrm{Bin}(20, 1/2)$ the theoretical mean is 10 and the theoretical variance is 5. The empirical mean of 5000 simulated draws lies within 0.3 of the theoretical mean in the seeded run that we used as a test.

4 Discussion

`scipy.stats.binom` would have covered everything. We wrote a small library because we wanted to read the formulas inside, and because each function ends up being a single line that maps to the definition in the probability chapter [3].

The default `scipy.special.comb` returns a float, which is fast but already rounded for moderately large n. With `exact=True` the function returns a Python integer, so the factor in front of $p^k(1-p)^{n-k}$ is exact and only the probabilities contribute rounding error.

The library handles only the binomial distribution. A natural extension is the multinomial distribution, which generalises to more than two outcomes per trial. The same approach (an exact multinomial coefficient and an explicit simulator) would carry over.

5 Conclusion

We have presented `binomial`, a small library for the binomial distribution. The library agrees with the textbook values for the fair-coin case and the simulator produces samples whose empirical mean is close to the theoretical mean.

References

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

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

[3] Sheldon M. Ross. *A First Course in Probability*. Pearson, 9 edition, 2014.

[4] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020.