How to Easily Generate Uniformly Distributed Random Numbers with Python

Authored by
stats writer

December 2, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Generate Uniformly Distributed Random Numbers with Python*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=103558

ARABPSYCHOLOGY.COM

The Uniform Distribution stands as a foundational concept in statistics and is immensely useful when modeling scenarios where every outcome within a specific range is equally likely. In the realm of data science and computation, particularly within Python, this distribution provides a powerful tool for generating random data and calculating probabilities.

Understanding how to implement the Uniform Distribution allows practitioners to simulate real-world phenomena--such as arrival times, measurement errors, or random sampling--where there is no inherent bias toward any particular value between a defined minimum and maximum. This article will guide you through using essential Python libraries, specifically NumPy for generating random variates and SciPy for calculating probabilities related to this specific probability distribution.

We will delve into the theoretical definition, explore the practical implementation using code examples, and demonstrate how to solve common probability questions related to continuous uniform random variables. Mastering these techniques is crucial for anyone involved in statistical modeling or Monte Carlo simulations in Python.

Defining the Continuous Uniform Distribution (Theoretical Foundation)

A continuous Uniform Distribution is a type of probability distribution characterized by the fact that every value within a defined interval, stretching from a lower bound, *a*, to an upper bound, *b*, has the exact same likelihood of occurring. If you visualize this distribution, its probability density function appears as a flat rectangle over the interval , hence the alternative name: the rectangular distribution. This concept is vital for modeling situations where we assume complete uncertainty or neutrality regarding intermediate values.

The critical characteristic of this distribution is its constant probability density across the range. Outside of the interval (a, b), the probability density is zero, meaning no values outside of this range are possible. This simplicity makes the Uniform Distribution an excellent starting point for understanding more complex distributions and is often used in situations where there is complete uncertainty regarding which value within a known range will materialize, such as rounding errors or the phase of a randomly observed cyclical process.

When working with real-world applications, identifying the boundaries *a* and *b* is the first step. These boundaries define the support of the distribution and dictate all subsequent calculations of probability and expected values. For instance, if a timing mechanism is known to produce delays uniformly between 10 seconds and 30 seconds, a=10 and b=30, forming the basis for all predictive analysis using this distribution model.

ARABPSYCHOLOGY.COM

Calculating Probability: The Uniform Distribution Formula

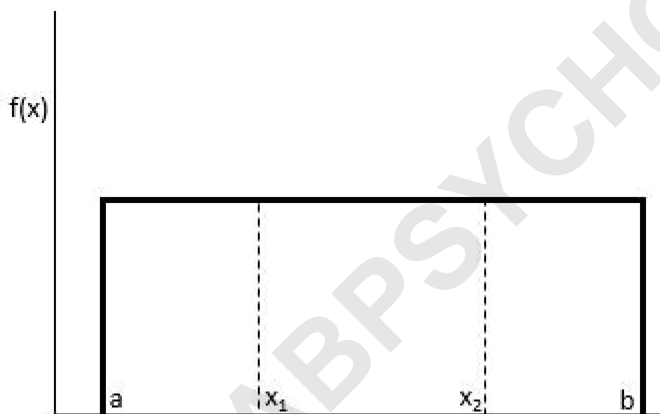
One of the primary goals of using any probability distribution is to calculate the likelihood of observing values within a specific sub-range. For the continuous Uniform Distribution, the calculation is remarkably straightforward, relying purely on the ratio of the length of the desired interval to the total length of the distribution's support.

The probability (P) that we will obtain a value between two points, x1 and x2, where both fall within the total interval from *a* to *b*, can be found using the following intuitive formula. This calculation essentially measures the proportion of the entire distribution range covered by the sub-range of interest, demonstrating the equal density property across the interval:

P(obtain value between x1 and x2) = (x2 - x1) / (b - a)

Here, (x2 - x1) represents the length of the interval of interest, and (b - a) represents the total length of the distribution's range. This formula elegantly captures the equal likelihood property: the wider the interval , the higher the probability, proportional only to its size relative to the total range . The result must always be between 0 and 1, inclusive.

ARABPSYCHOLOGY.COM



Generating Random Data with NumPy's Uniform Function

While the mathematical formula is useful for theoretical calculations, modern data science often requires generating a large volume of random numbers that follow the Uniform Distribution. This is typically necessary for tasks such as Monte Carlo simulations, initializing weights in machine learning models, or creating synthetic test datasets. Python's NumPy library provides the highly efficient function `numpy.random.uniform()` for this purpose, which is the standard tool for generating arrays of random data.

The `numpy.random.uniform()` function is designed to quickly produce an array of random floating-point numbers drawn from a continuous uniform distribution. It requires three primary

parameters to define the desired output array and the distribution boundaries, ensuring that the generated numbers accurately reflect the specified range:

low: The lower boundary (inclusive) of the output interval (equivalent to *a*). The default value is 0.0.
high: The upper boundary (exclusive) of the output interval (equivalent to *b*). The default value is 1.0.
size: The shape or dimensions of the output array. This allows the generation of single numbers, 1D arrays, or multi-dimensional matrices of uniform random data, offering great flexibility for various computational needs.

By defining these parameters, a user can generate numbers precisely tailored to their simulation needs. For example, generating 10,000 random numbers between 5 and 10 is accomplished simply by calling `numpy.random.uniform(low=5, high=10, size=10000)`. This simplicity, coupled with the speed of NumPy's underlying C implementation, is why it remains the de facto standard for efficient random number generation in Python statistical computing, particularly when dealing with large datasets or complex simulations.

Analyzing Probabilities with SciPy's uniform Module

When the requirement shifts from generating random data to calculating exact probabilities--such as finding the likelihood of an event occurring within a specific range--we turn to the comprehensive statistical functions offered by the SciPy library. Specifically, the `scipy.stats.uniform` module provides methods for calculating the Probability Density Function (PDF), the Percent Point Function (PPF), and most commonly for solving probability problems, the Cumulative Distribution Function (CDF).

The Cumulative Distribution Function (CDF) is mathematically defined as the probability that a random variable X will take a value less than or equal to x (i.e., $P(X \leq x)$). This function is monotonic and ranges from 0 to 1 across the support of the distribution. When using SciPy to find probabilities over an interval , we utilize the fundamental relationship that the probability of being within the interval is the difference between the CDF evaluated at the upper point and the CDF evaluated at the lower point: $P(x1 < X \leq x2) = CDF(x2) - CDF(x1)$.

The basic syntax for utilizing the `scipy.stats.uniform` functions is structured somewhat differently than NumPy, as it relies on location and scale parameters rather than explicit lower and upper bounds. The function signature is structured as follows, where it is critical to correctly map the real-world constraints to these statistical parameters:

scipy.stats.uniform(x, loc, scale)

The parameters used by SciPy are defined in relation to the distribution's boundaries (a and b):

ARABPSYCHOLOGY.COM

x: The specific value at which the function (e.g., CDF) is evaluated.
loc: The location parameter, which corresponds exactly to the minimum possible value (*a*).
scale: The scale parameter, which represents the width of the distribution, calculated as the difference between the maximum and minimum values (b - a). Thus, the maximum possible value (*b*) is `loc + scale`.

The following detailed examples illustrate how to apply the SciPy framework to solve common continuous Uniform Distribution problems using the CDF method, showcasing the transition from theoretical problem setup to precise programmatic calculation.

Practical Application 1: Time Interval Probability (Bus Example)

Consider a scenario involving public transit where a bus adheres strictly to a schedule, showing up at a specific bus stop every 20 minutes. Due to the continuous nature of human arrival, if you arrive randomly at the bus stop, the time you wait until the next bus is a continuous uniform random variable. Our goal is to determine the probability that the bus will show up in 8 minutes or less from the moment you arrive.

In this example, the minimum wait time (a, or `loc`) is 0 minutes, and the maximum wait time (b) is 20 minutes, as the longest one would ever have to wait is just under the 20-minute cycle. Therefore, the scale of the distribution is 20 - 0 = 20. We are interested in the probability $P(X \leq 8)$. Since the minimum value of the distribution is 0, $P(X \leq 8)$ is equivalent to $P(0 \leq X \leq 8)$, which must be calculated using the CDF difference: CDF(8) - CDF(0). This calculation isolates the area under the uniform density function corresponding to the first 8 minutes of the cycle.

We implement the calculation in Python using the `uniform.cdf` method, where `loc=0` and `scale=20`:

```
from scipy.stats import uniform

# Calculate the cumulative probability up to 8 minutes
uniform.cdf(x=8, loc=0, scale=20) - uniform.cdf(x=0, loc=0, scale=20)

0.4
```

The result, **0.4**, indicates that there is a 40% chance that the bus will arrive within 8 minutes of your random arrival. This perfectly aligns with the theoretical calculation based on the ratio of the interval lengths: (8 - 0) / (20 - 0) = 8/20 = 0.4. This confirms the accuracy and validity of the SciPy implementation for basic uniform probability problems.

ARABPSYCHOLOGY.COM

Practical Application 2: Weight Range Probability (Frog Example)

Statistical models often use the Uniform Distribution to describe natural characteristics when data collection is limited or when variation is assumed to be evenly spread across a known range. Suppose the weight of a particular species of frog is uniformly distributed between 15 grams and 25 grams. If a frog is randomly selected from this population, we want to find the probability that its weight falls specifically between 17 grams and 19 grams.

Here, the distribution parameters are clearly defined: the lower bound (a, or `loc`) is 15 grams, and the upper bound (b) is 25 grams. Consequently, the scale parameter, which defines the total span of weights, is 25 - 15 = 10. We are interested in calculating the probability P(17 < X < 19), which requires calculating the difference between the Cumulative Distribution Function evaluated at 19 and 17, respectively. Both 17 and 19 are valid points within the distribution's support.

The Python implementation using the `uniform.cdf` function looks as follows, ensuring the correct `loc=15` and `scale=10` parameters are used for the distribution defined by the 15 to 25 gram range:

from scipy.stats import uniform

```
# Calculate the probability that weight is between 17 and 19 grams
uniform.cdf(x=19, loc=15, scale=10) - uniform.cdf(x=17, loc=15, scale=10)
```

0.2

The resulting probability is **0.2**. This means there is a 20% chance that a randomly selected frog will weigh between 17 and 19 grams. We can confirm this using the manual formula: the desired interval length is (19 - 17) = 2, and the total interval length is (25 - 15) = 10, resulting in 2 / 10 = 0.2. This showcases how the SciPy module automates the application of the theoretical formula.

Practical Application 3: Duration Probability (NBA Game Example)

The Uniform Distribution is also useful for modeling duration or length when extreme precision is not required or when complex factors make a normal distribution inappropriate. Assume that the total length of an NBA game (including overtime and stoppages) is uniformly distributed between 120 minutes and 170 minutes. We want to determine the probability that a randomly selected NBA game lasts more than 150 minutes.

In this scenario, the total range is from 120 (a, or `loc`) to 170 (b) minutes. The scale of the distribution is 170 - 120 = 50. We are looking for the probability P(X > 150). Since this is a continuous distribution, finding the probability of the upper tail requires calculating P(150 < X ≤ 170), which corresponds to the difference: CDF(170) - CDF(150).

We execute the calculation in Python, setting the boundaries correctly using `loc=120` and `scale=50`:

```
from scipy.stats import uniform

# Calculate the probability that duration is between 150 minutes and the maximum (170 minutes)
uniform.cdf(x=170, loc=120, scale=50) - uniform.cdf(x=150, loc=120, scale=50)

0.4
```

The probability that the game lasts more than 150 minutes is calculated as **0.4**. This result is confirmed by the ratio: the desired interval is (170 - 150) = 20 minutes, and the total span is 50 minutes, resulting in 20 / 50 = 0.4. This example demonstrates the utility of SciPy for quickly solving conditional probability questions within a uniform framework, which is crucial for risk analysis or predictive scheduling.

ARABPSYCHOLOGY.COM

Verification and Further Exploration of Distributions

The examples above illustrate the fundamental methodology for working with the continuous Uniform Distribution in Python using SciPy's `uniform.cdf` method. It is important to remember that for simple uniform problems, the manual formula provides an excellent way to double-check the code output, ensuring that the parameters (loc and scale) have been set correctly and that the conceptual model aligns with the programmatic implementation.

Bonus Tip: You can double check the solution to each example by using the manual probability formula: P = (x2 - x1) / (b - a). When x2 is the upper bound *b*, the probability simplifies to the length of the upper interval divided by the total scale.

While the Uniform Distribution is straightforward, it serves as a gateway to understanding more complex probability models. Many real-world phenomena are better described by other shapes, such as the Gaussian (Normal) distribution, the Exponential distribution, or the Binomial distribution. The techniques demonstrated here--using CDFs to calculate interval probabilities and relying on libraries like NumPy and SciPy--are transferable across virtually all statistical distributions in Python.

The following tutorials explain how to use other common distributions in Python, building upon the foundational knowledge of handling distribution functions in SciPy and NumPy:

Using the Normal Distribution for continuous data modeling.
Exploring the Poisson Distribution for count data analysis.
Implementing the Binomial Distribution for discrete success/failure trials.