Модуль random, случайные числа в Python

Ге распределений и псевдослучайных чисел

Мо аторы псевдослучайных чисел для различных распределений.

Дл вномерный выбор из диапазона. Для последовательностей существует равномерный выбор сл для генерации случайной перестановки списка на месте и функция для случайной выборки без замены.

На реальной линии есть функции для вычисления равномерного, нормального (гауссовского), логнормального, отрицательного экспоненциального, гамма и бета распределений. Для генерации распределений углов доступно распределение фон Мизеса.

Почти все функции модуля зависят от базовой функции random.random(), которая генерирует случайное число с плавающей точкой в полуоткрытом диапазоне [0.0, 1.0]. Python использует Mersenne Twister в качестве генератора ядра. Он генерирует 53-битные значения точности и имеет период 2 ** 19937-1. Базовая реализация в C является быстрой и поточно-ориентированной. Mersenne Twister является одним из наиболее тщательно протестированных генераторов случайных чисел из существующих. Однако, будучи полностью детерминированным, он не подходит для всех целей и совершенно не подходит для криптографических целей.

Функции, предоставляемые этим модулем, на самом деле являются связанными методами скрытого экземпляра класса random.Random. Вы можете создать свои собственные экземпляры Random(), чтобы получить генераторы, которые не делятся состоянием.

Класс Random() также можно разделить на подклассы, если вы хотите использовать другой базовый генератор вашего собственного устройства: в этом случае переопределите методы random.random(), random.seed(), random.getstate() и random.setstate(). При желании новый генератор может предоставить метод random.getrandbits() - это позволяет random.randrange() производить выборки в произвольно большом диапазоне.

Модуль random также предоставляет класс random.SystemRandom, который использует системную функцию os.urandom() для генерации случайных чисел из источников, предоставляемых операционной системой.

Предупреждение.

Псевдослучайные генераторы этого модуля не должны использоваться в целях безопасности. В целях безопасности или криптографического использования смотрите "Модуль secrets".

Замечания по воспроизводимости последовательностей.

Иногда полезно иметь возможность воспроизвести последовательности, заданные генератором псевдослучайных чисел. При повторном использовании начального значения seed, одна и та же последовательность должна воспроизводиться от запуска к запуску, пока не запущено несколько потоков.

Бо во алгоритмов и функций модуля могут изменяться в разных версиях Python, но два аспекта гарантированно не изменятся:

- Если будет добавлен новый метод, то обязательно будет предложена обратная совместимость.
- Метод генератора random() будет продолжать создавать ту же последовательность, если совместимому методу будет дано то же самое начальное число seed.

Примеры использования модуля random.

Базовое применение модуля:

```
>>> import random
# Случайное float:  0.0 <= x < 1.0
>>> random.random()
# 0.37444887175646646

# Случайное float:  2.5 <= x < 10.0
>>> random.uniform(2.5, 10.0)
# 3.1800146073117523

# Интервал между прибытием в среднем 5 секунд
>>> random.expovariate(1 / 5)
# 5.148957571865031

# Четное целое число от 0 до 100 включительно
>>> random.randrange(10)
# 7

# Even integer from 0 to 100 inclusive
>>> random.randrange(0, 101, 2)
26

# Один случайный элемент из последовательности
>>> random.choice(['win', 'lose', 'draw'])
'draw'

>>> deck = 'ace two three four'.split()
# Перемешать список
>>> random.shuffle(deck)
>>> deck
['four', 'two', 'ace', 'three']

# Четыре образца без замены
>>> random.sample([10, 20, 30, 40, 50], k=4)
# [40, 10, 50, 30]
```

Имитационные расчеты:

```
# Шесть вращений колеса рулетки (взвешенная выборка с заменой)
>>> choices(['red', 'black', 'green'], [18, 18, 2], k=6)
# ['red', 'green', 'black', 'black', 'red', 'black']

# Сдайте 20 карт без замены из колоды из 52 игральных карт
# и определите пропорцию карт с достоинством в:
# десять, валет, дама или король.
>>> dealt = sample(['tens', 'low cards'], counts=[16, 36], k=20)
>>> dealt.count('tens') / 20
# 0.15

# Оценка вероятности получения 5 или более попаданий из 7
# бросаний монеты, которая выпадает орлом в 60% случаев.
>>> def trial():
...     return choices('HT', cum_weights=(0.60, 1.00), k=7).count('H') >= 5
...
>>> sum(trial() for i in range(10_000)) / 10_000
# 0.4169

>>> # Вероятность того, что медиана из 5 выборок находится в средних двух квартилях
>>> def trial():
...     return 2_500 <= sorted(choices(range(10_000), k=5))[2] < 7_500
...
```

Вверх

```
>>> sum(trial() for i in range(10_000)) / 10_000
# 0.7958
```

Пример статистической начальной загрузки с использованием повторной выборки с заменой для оценки доверительного интервала для среднего значения выборки:


```
# http://statistics.about.com/od/Applications/a/Example-Of-Bootstrapping.htm
from statistics import fmean as mean
from random import choices

data = [41, 50, 29, 37, 81, 30, 73, 63, 20, 35, 68, 22, 60, 31, 95]
means = sorted(mean(choices(data, k=len(data))) for i in range(100))
print(f'The sample mean of {mean(data):.1f} has a 90% confidence '
      f'interval from {means[5]:.1f} to {means[94]:.1f}')
```

Пример теста перестановки повторной выборки для определения статистической значимости или P-значения наблюдаемой разницы между эффектами препарата и плацебо:

```
# Example from "Statistics is Easy" by Dennis Shasha and Manda Wilson
from statistics import fmean as mean
from random import shuffle

drug = [54, 73, 53, 70, 73, 68, 52, 65, 65]
placebo = [54, 51, 58, 44, 55, 52, 42, 47, 58, 46]
observed_diff = mean(drug) - mean(placebo)

n = 10_000
count = 0
combined = drug + placebo
for i in range(n):
    shuffle(combined)
    new_diff = mean(combined[:len(drug)]) - mean(combined[len(drug):])
    count += (new_diff >= observed_diff)

print(f'{n} label reshufflings produced only {count} instances with a difference')
print(f'at least as extreme as the observed difference of {observed_diff:.1f}.')
print(f'The one-sided p-value of {count / n:.4f} leads us to reject the null')
print(f'hypothesis that there is no difference between the drug and the placebo.')
```

Моделирование времени прибытия и доставки услуг для многосерверной очереди:

```
from heapq import heappush, heappop
from random import expovariate, gauss
from statistics import mean, median, stdev

average_arrival_interval = 5.6
average_service_time = 15.0
stdev_service_time = 3.5
num_servers = 3

waits = []
arrival_time = 0.0
servers = [0.0] * num_servers  # time when each server becomes available
for i in range(100_000):
    arrival_time += expovariate(1.0 / average_arrival_interval)
    next_server_available = heappop(servers)
    wait = max(0.0, next_server_available - arrival_time)
    waits.append(wait)
    service_duration = gauss(average_service_time, stdev_service_time)
    service_completed = arrival_time + wait + service_duration
    heappush(servers, service_completed)

print(f'Mean wait: {mean(waits):.1f}.  Stdev wait: {stdev(waits):.1f}.')
print(f'Median wait: {median(waits):.1f}.  Max wait: {max(waits):.1f}.')
```

Содержание раздела:

Вверх