Документация коду

Описание

Этот код предназначен для генерации случайных чисел двумя различными методами, анализа полученных выборок и сравнения времени выполнения этих методов с методом генерации случайных чисел из библиотеки NumPy.

Установка зависимостей

```
!pip install nistrng
```

Импорт библиотек

```
import random
import time
import numpy as np
import scipy.stats as stats
import timeit
from nistrng import *
import matplotlib.pyplot as plt
from math import log
```

- `random` : стандартная библиотека Python для генерации случайных чисел.
- `time`, `timeit` : для измерения времени выполнения.
- `numpy` : библиотека для работы с массивами и матрицами.
- `scipy.stats` : библиотека для статистических функций.
- `nistrng` : библиотека для тестирования случайных чисел.
- `matplotlib.pyplot` : библиотека для построения графиков.
- `math.log` : математические функции.

Генерация случайных чисел

Метод 1: Линейный конгруэнтный генератор

Функция `gen_1` генерирует последовательность случайных чисел, используя линейный конгруэнтный метод.

```
def gen_1(n, a=22695477, c=1, m=2**32):
    res = []
    s = [i for i in range(10000)]
    x = random.choice(s)
    for i in range(n):
        x = (a * x + c) % m
        res.append(x % 5000)
    return res

samples_1 = [gen_1(1000) for _ in range(20)]
```

Метод 2: Генератор на основе чисел Фибоначчи

Функция `gen_2` генерирует последовательность случайных чисел, используя генератор на основе чисел Фибоначчи.

```
def fib_gen(seed1, seed2, modul = 2**31 - 1):
    while True:
        next_seed = (seed1 + seed2) % modul
        seed1, seed2 = seed2, next_seed
        yield next_seed

def gen_2(n, samp_cnt = 20):
    samples = []
    for i in range(samp_cnt):
        seed1, seed2 = i, i + 1
        gen = fib_gen(seed1, seed2)
        sample = [next(gen) % 5000 for _ in range(1000)]
        samples.append(sample)
    return samples

samples_2 = gen_2(1000)
```

Анализ выборок

Статистический анализ

Выводятся среднее значение, стандартное отклонение и коэффициент вариации для каждой выборки.

```
for i, sample in enumerate(samples_1):
    print(f"Sample {i+1} (Method 1): Среднее = {np.mean(sample)}, Отклонение = {np.std(sample)}, Коэффицент вариации = {np.std(sampl
for i, sample in enumerate(samples_2):
    print(f"Sample {i+1} (Method 2): Среднее = {np.mean(sample)}, Отклонение = {np.std(sample)}, Коэффицент вариации = {np.std(sampl
```

Тесты на соответствие распределению

Проводится тест хи-квадрат для проверки гипотезы о соответствии распределению.

```
for i, sample in enumerate(samples_1):
    frequencies, _ = np.histogram(sample, bins=np.arange(5001))
    print(f"Выборка {i+1} (метод 1): Хи-квадрат не отвергается с вероятностью = {stats.chisquare(frequencies)[1]}")
for i, sample in enumerate(samples_2):
    frequencies, _ = np.histogram(sample, bins=np.arange(5001))
    print(f"Выборка {i+1} (метод 2): Хи-квадрат не отвергается с вероятностью = {stats.chisquare(frequencies)[1]}")
```

Тесты на случайность NIST

Проводится тестирование выборок с использованием тестов NIST-SP800-22r1a.

```
for samples in [samples_1[:5], samples_2[:5]]:
    for i, sample in enumerate(samples):
        binary_sequence: np.ndarray = pack_sequence(samples)
        eligible_battery: dict = check_eligibility_all_battery(binary_sequence, SP800_22R1A_BATTERY)
        print(f'Eligible test from NIST-SP800-22r1a for sample {i}:')
        for name in eligible_battery.keys():
            print("-" + name)
        results = run_all_battery(binary_sequence, eligible_battery, False)
        print("Test results:")
        for result, elapsed_time in results:
            if result.passed:
                print("- PASSED - score: " + str(np.round(result.score, 3)) + " - " + result.name + " - elapsed time: " + str(elapse
            else:
                print("- FAILED - score: " + str(np.round(result.score, 3)) + " - " + result.name + " - elapsed time: " + str(elapse
```

Сравнение времени генерации

Проводится измерение времени генерации случайных чисел разными методами.

```
 method_1 = []
method_2 = []
method_numpy = []

for n in [1000, 10000, 100000, 1000000]:
    start = timer()
    gen_1(n)
    print(f"Time for generating {n} numbers using Method 1: {timer() - start}")
    method_1.append(timer() - start)

    start = timer()
    gen_2(n)
    print(f"Time for generating {n} numbers using Method 2: {timer() - start}")
    method_2.append(timer() - start)

    start = timer()
    np.random.randint(0, 5000, size=n)
    print(f"Time for generating {n} numbers using numpy: {timer() - start}")
    method_numpy.append(timer() - start)
```

Построение графика

Построение графика для сравнения времени генерации случайных чисел.

```
 plt.plot([1000, 10000, 100000, 1000000], method_1, [1000, 10000, 100000, 1000000], method_2, [1000, 10000, 100000, 1000000], method_
plt.legend(['method_1', 'method_2', 'numpy'])
plt.xticks(range(1000, 1000000, 499000))
plt.show()
```

Заключение

Этот код позволяет генерировать случайные числа двумя различными методами, анализировать их статистические свойства и сравнивать производительность каждого метода.