Отчет по третьей лабораторной работе

Этот модуль содержит классы и функции для генерации случайных чисел, анализа их статистических свойств, а также для выполнения тестов на равномерность и других тестов случайности.

Полный код программы

```
import numpy as np
import time
import matplotlib.pyplot as plt
from scipy.stats import chisquare
from nistrng import check_eligibility_all_battery, SP800_22R1A_BATTERY,
run_all_battery


class SimpleRandom:
    def __init__(self, seed=5489):
        self.seed = seed

    def random(self):
        self.seed ^= (self.seed << 20)
        self.seed ^= (self.seed >> 35)
        self.seed ^= (self.seed << 5)
        return self.seed % 100


# Линейный конгруэнтный генератор (LCG)
class LCG:
    def __init__(self, seed=1, a=1664525, c=1013904223, m=2**32):
        self.seed = seed
        self.a = a
        self.c = c
        self.m = m
        self.state = seed

    def random(self):
        self.state = (self.a * self.state + self.c) % self.m
        return self.state / self.m


# Генерация выборок
def generate_samples(
    generator, num_samples=20, sample_size=100, range_min=0,
range_max=4999
):
    samples = []
    for _ in range(num_samples):
        sample = [
            int(generator.random() * (range_max - range_min + 1) +
```

```
range_min)
            for _ in range(sample_size)
        ]
        samples.append(sample)
    return samples


# Статистика выборок
def calculate_statistics(samples):
    statistics = []
    for sample in samples:
        mean = np.mean(sample)
        std_dev = np.std(sample)
        coeff_var = std_dev / mean if mean != 0 else 0
        statistics.append((mean, std_dev, coeff_var))
    return statistics


# Проверка на равномерность распределения (Хи-квадрат)
def chi_square_test(samples, num_bins=10):
    test_results = []
    for sample in samples:
        observed_freq, _ = np.histogram(
            sample, bins=num_bins, range=(min(sample), max(sample))
        )
        expected_freq = [len(sample) / num_bins] * num_bins
        chi2, p = chisquare(observed_freq, expected_freq)
        test_results.append((chi2, p))
    return test_results


# Проверка времени генерации чисел
def benchmark(generator, sizes):
    times = []
    for size in sizes:
        print(f"Benchmarked: {size}")
        start_time = time.time()
        [generator.random() for _ in range(size)]
        end_time = time.time()
        times.append(end_time - start_time)
    return times


# Вывод статистики
def print_statistics(generator_name, statistics, chi2_results):
    print(f"\nStatistics for {generator_name}:")
    for i, (mean, std_dev, coeff_var) in enumerate(statistics):
        print(
            f"Sample {i+1}: Mean={mean}, StdDev={std_dev}, CoeffVar=
{coeff_var}"
        )

    print(f"\nChi-Square Test Results for {generator_name}:")
    for i, (chi2, p) in enumerate(chi2_results):
```

```
        print(f"Sample {i+1}: Chi2={chi2}, p-value={p}")


# Сохранение статистики в файл
def save_statistics_to_file(
    filename, generator_name, statistics, chi2_results
):
    with open(filename, "w") as file:
        file.write(f"Statistics for {generator_name}:\n")
        for i, (mean, std_dev, coeff_var) in enumerate(statistics):
            file.write(
                f"Sample {i+1}: Mean={mean}, StdDev={std_dev}, CoeffVar=
{coeff_var}\n"
            )

        file.write(f"\nChi-Square Test Results for {generator_name}:\n")
        for i, (chi2, p) in enumerate(chi2_results):
            file.write(f"Sample {i+1}: Chi2={chi2}, p-value={p}\n")


# Проверка с помощью тестов NIST
def run_nist_tests(data):
    arr = np.array(data)
    eligible_battery: dict = check_eligibility_all_battery(arr,
SP800_22R1A_BATTERY)

    print("Eligible test from NIST-SP800-22r1a:")
    for name in eligible_battery.keys():
        print("\t" + name)

    results = run_all_battery(arr, eligible_battery, False)

    print("Test results:")
    for result, elapsed_time in results:
        if result.passed:
            print("\tPASSED - score: " + str(np.round(result.score, 3)) +
" - " + result.name + " - elapsed time: " + str(elapsed_time) + " ms")
        else:
            print("\tFAILED - score: " + str(np.round(result.score, 3)) +
" - " + result.name + " - elapsed time: " + str(elapsed_time) + " ms")


# Основная функция
def main():
    # Создаем генераторы
    lcg = LCG(seed=12345)
    simple_rnd = SimpleRandom(seed=12345)

    # Генерация выборок
    lcg_samples = generate_samples(lcg)
    custom_mt_samples = generate_samples(simple_rnd)

    # Рассчет статистики
    lcg_statistics = calculate_statistics(lcg_samples)
```

```
    custom_mt_statistics = calculate_statistics(custom_mt_samples)

    # Проверка на равномерность
    lcg_chi2 = chi_square_test(lcg_samples)
    custom_mt_chi2 = chi_square_test(custom_mt_samples)

    # Вывод и сохранение статистики
    print_statistics("LCG", lcg_statistics, lcg_chi2)
    print_statistics(
        "Simple Random", custom_mt_statistics, custom_mt_chi2
    )

    save_statistics_to_file(
        "lcg_statistics.txt", "LCG", lcg_statistics, lcg_chi2
    )
    save_statistics_to_file(
        "simple_rnd_statistics.txt",
        "Simple Random",
        custom_mt_statistics,
        custom_mt_chi2,
    )

    # Проверка времени генерации чисел
    sizes = [100, 1000, 10000, 100000]
    lcg_times = benchmark(lcg, sizes)
    custom_mt_times = benchmark(simple_rnd, sizes)
    np_times = benchmark(np.random, sizes)

    print("NIST TEST FOR 10 000")
    run_nist_tests([round(lcg.random() * 100) for _ in range(10000)])
    run_nist_tests([round(simple_rnd.random() * 100) for _ in
range(10000)])

    print("NIST TEST FOR 100 000")
    run_nist_tests([round(lcg.random() * 100) for _ in range(100000)])
    run_nist_tests([round(simple_rnd.random() * 100) for _ in
range(100000)])

    # Построение графиков
    plt.figure()
    plt.plot(sizes, lcg_times, label="LCG")
    plt.plot(sizes, custom_mt_times, label="Simple Random")
    plt.plot(sizes, np_times, label="NumPy Random")
    plt.xlabel("Size of Samples")
    plt.ylabel("Generation Time (seconds)")
    plt.legend()
    plt.show()


if __name__ == "__main__":
    main()
```

Описание функций и классов

Классы

`SimpleRandom`

Класс для генерации случайных чисел с использованием простого генератора.

Методы:

- `__init__(self, seed=5489)`: Инициализирует генератор с заданным начальным значением `seed`.
- `random(self)`: Генерирует случайное число, используя простые побитовые операции.

`LCG`

Класс для генерации случайных чисел с использованием линейного конгруэнтного генератора (LCG).

Методы:

- `__init__(self, seed=1, a=1664525, c=1013904223, m=2**32)`: Инициализирует генератор с заданными параметрами `seed`, `a`, `c` и `m`.
- `random(self)`: Генерирует случайное число, используя формулу линейного конгруэнтного генератора.

Функции

`generate_samples(generator, num_samples=20, sample_size=100, range_min=0, range_max=4999)`

Генерирует выборки случайных чисел, используя заданный генератор.

Аргументы:

- `generator`: Экземпляр генератора случайных чисел.
- `num_samples` (int): Количество выборок.
- `sample_size` (int): Размер каждой выборки.
- `range_min` (int): Минимальное значение в диапазоне.
- `range_max` (int): Максимальное значение в диапазоне.

Возвращает:

- `samples` (list): Список выборок.

`calculate_statistics(samples)`

Рассчитывает статистические показатели для каждой выборки.

Аргументы:

- `samples` (list): Список выбор

ок.

Возвращает:

- `statistics` (list): Список кортежей, содержащих среднее значение, стандартное отклонение и коэффициент вариации для каждой выборки.

`chi_square_test(samples, num_bins=10)`

Проверяет равномерность распределения выборок с помощью критерия хи-квадрат.

Аргументы:

- `samples` (list): Список выборок.
- `num_bins` (int): Количество бинов для гистограммы.

Возвращает:

- `test_results` (list): Список кортежей, содержащих значение хи-квадрат и p-значение для каждой выборки.

`benchmark(generator, sizes)`

Измеряет время генерации случайных чисел для различных размеров выборок.

Аргументы:

- `generator`: Экземпляр генератора случайных чисел.
- `sizes` (list): Список размеров выборок.

Возвращает:

- `times` (list): Список времен генерации для каждого размера выборки.

`print_statistics(generator_name, statistics, chi2_results)`

Выводит статистические показатели и результаты теста хи-квадрат для генератора.

Аргументы:

- `generator_name` (str): Название генератора.
- `statistics` (list): Статистические показатели.
- `chi2_results` (list): Результаты теста хи-квадрат.

`save_statistics_to_file(filename, generator_name, statistics, chi2_results)`

Сохраняет статистические показатели и результаты теста хи-квадрат в файл.

Аргументы:

- `filename` (str): Имя файла.
- `generator_name` (str): Название генератора.
- `statistics` (list): Статистические показатели.
- `chi2_results` (list): Результаты теста хи-квадрат.

`run_nist_tests(data)`

Выполняет тесты NIST для проверки случайности данных.

Аргументы:

- `data` (list): Данные для тестирования.

`main()`

Основная функция, выполняющая все шаги анализа.

Алгоритм:

1. Создает экземпляры генераторов `LCG` и `SimpleRandom`.
2. Генерирует выборки случайных чисел.
3. Рассчитывает статистические показатели для выборок.
4. Проверяет равномерность распределения выборок с помощью критерия хи-квадрат.
5. Выводит и сохраняет статистические показатели.
6. Измеряет время генерации случайных чисел для различных размеров выборок.
7. Выполняет тесты NIST для данных.
8. Строит графики времени генерации случайных чисел.

Заключение

В данном отчете представлен код для генерации случайных чисел с использованием различных генераторов, анализа их статистических свойств, проверки на равномерность распределения и выполнения тестов NIST. Были объяснены функции и классы, их параметры и суть используемых алгоритмов. Программа генерирует выборки случайных чисел, рассчитывает их статистические показатели, проверяет равномерность распределения, измеряет время генерации и выполняет тесты NIST. Результаты визуализируются с помощью графиков.