ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____
ПРЕПОДАВАТЕЛЬ

д-р техн. наук, профессор		Т.М. Татарникова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

МОДЕЛИРОВАНИЕ БАЗОВОЙ СЛУЧАЙНОЙ ВЕЛИЧИНЫ

Вариант 6

по курсу: МОДЕЛИРОВАНИЕ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4128		В.А.Тарапанов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

1 Цель и постановка задачи

1.1 Цель работы

Построить датчик базовой случайной величины по заданному алгоритму и выполнить тестирование датчика на соответствие основным свойствам базовой случайной величины.

1.2 Задание

1. Построить датчик БСВ с периодом T>500.

2. Оценить математическое ожидание и дисперсию псевдослучайных значений zi и сравнить их с теоретическими значениями M и D.

3. Проверить датчик БСВ на равномерность и построить гистограмму распределения относительных частот p1,…, pk на K отрезках интервала [0,1].

4. Проверить датчик БСВ на независимость, определяя коэффициент корреляции для разных значений s и T. Построить в одном графическом окне графики зависимости Rˆ = f (T) для s=2, s=5, s=10.

1.3 Вариант задания

Построить мультипликативно-конгруэнтный датчик

$$A_i = (A_{i-55} + A_{i-24}) \bmod (2^{32})$$
$$B_i = (B_{i-57} + B_{i-7}) \bmod (2^{32})$$
$$C_i = (C_{i-58} + C_{i-19}) \bmod (2^{32})$$
$$z_i = \frac{A_i}{2^{32}},\ z_{i+1} = \frac{B_i}{2^{32}},\ z_{i+2} = \frac{C_i}{2^{32}} \quad (1)$$

Состоит из трех аддитивных генераторов A_i, B_i и C_i. Начальные состояния этих генераторов создаются Random. При генерации сравниваются биты переноса при сложении. Если все три одинаковы (все нули или все единицы), то тактируются все три генератора. Если нет, то тактируются только два совпадающих генератора.

2 Ход работы

Работа выполнялась при помощи пакета прикладных программ для решения задач технических вычислений.

При помощи формулы 1, был запрограммирован мультипликативно-конгруэнтный датчик. Код включает в себя определение класса PeakGenerator, который реализует генератор псевдослучайных чисел с использованием мультипликативно-конгруэнтного метода.

Этот класс использует три массива (A, B, C) и три индекса (index_A, index_B, index_C), которые обновляются при каждой генерации числа. Каждый из массивов содержит начальные случайно сгенерированные значения длиной 55, 57 и 58 соответственно. Новое число генерируется путем вычисления суммы двух предыдущих чисел в соответствующем массиве и последующего взятия остатка от деления на 2^{32}.

2.1 Сравнение числовых значений

Как видно из рисунка 1, на основе проведенных вычислений можно сделать вывод о том, что реализованный мультипликативно-конгруэнтный датчик показывает хорошие результаты, близкие к теоретическим ожидаемым значениям для математического ожидания и дисперсии.

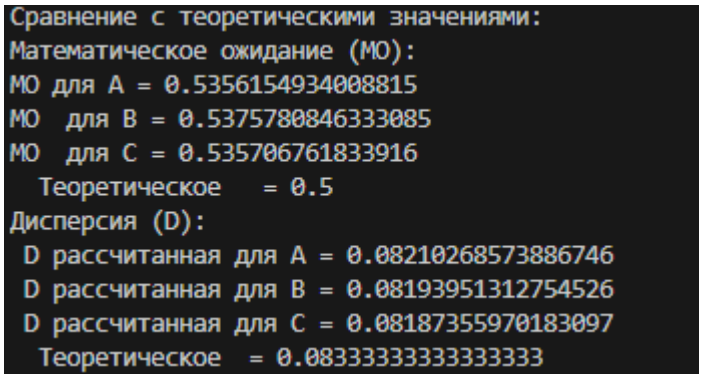

Рисунок 1 – Теоретические и экспериментальные значения математического ожидания и дисперсии

2.2 Равномерность

На рисунке 2 изображена гистограмма распределения относительных частот.

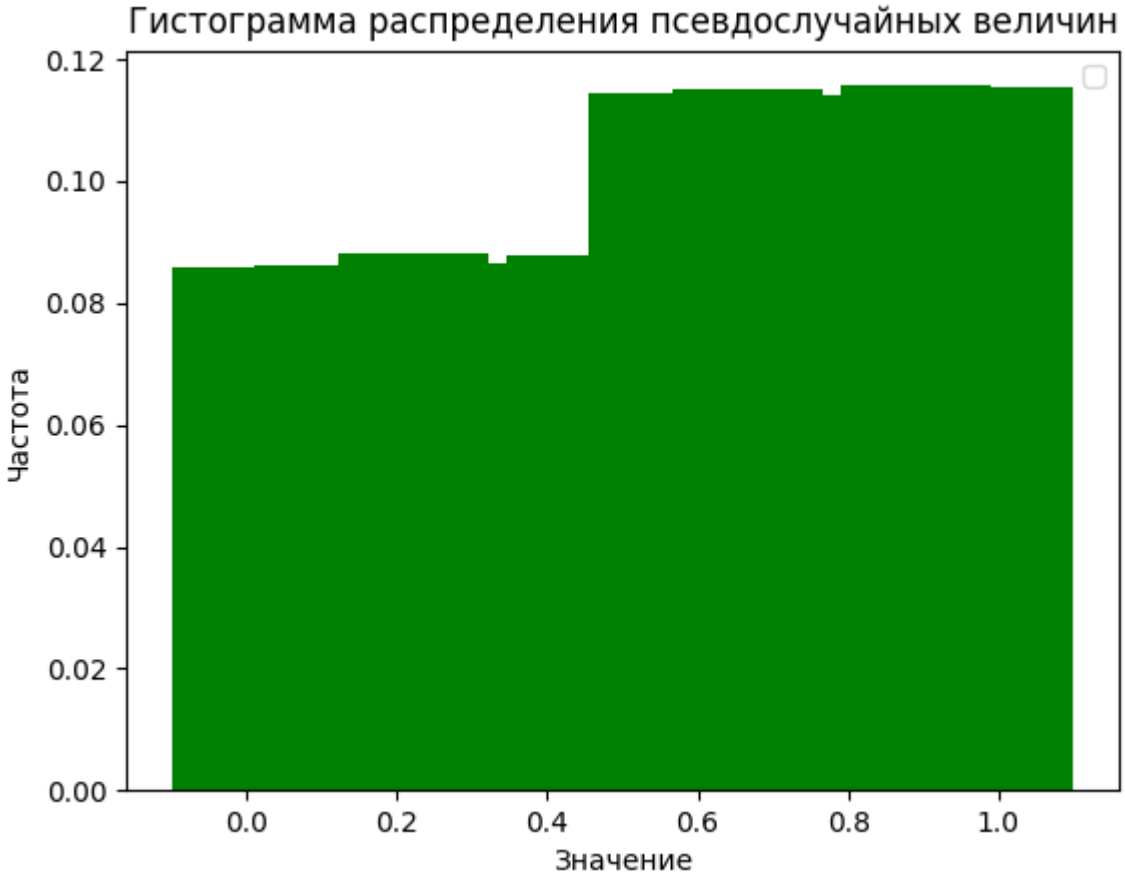

Рисунок 2 – Гистограмма распределения относительных частот

2.3 Независимость

При помощи формулы

$$\hat{R} = 12\frac{1}{T-s}\left(\sum_{i=1}^{T-s} z_i z_{i+s}\right) - 3 \qquad (2)$$

были построены графики зависимости коэффициента корреляции $\hat{R} = f(T)$ для s=2, s=5 и s=10. Они представлены на рисунке 3, из которого следует, что рассматриваемый датчик является статистически независимым для достаточно больших периодов, т.к. коэффициенты корреляции стремятся к нулю с ростом периода.

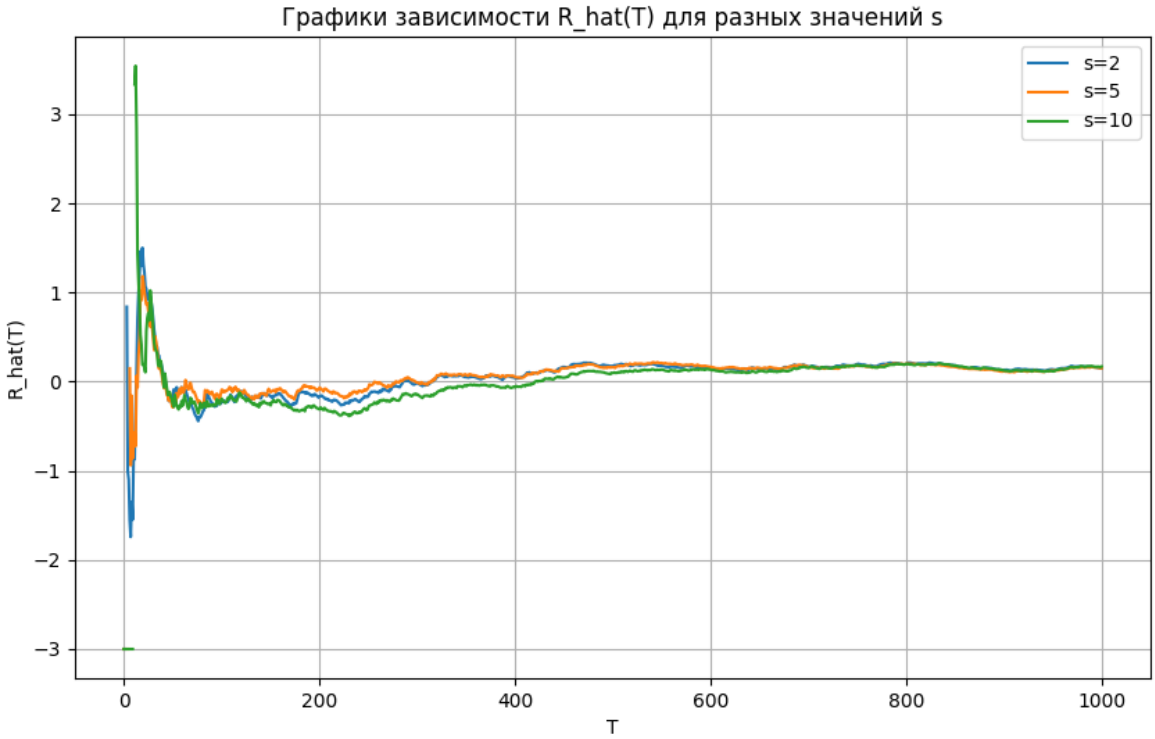

Рисунок 3 – График зависимости коэффициента корреляции от T

ВЫВОД

В результате проведенных вычислений,можно сделать следующие выводы о поведении последовательностей псевдослучайных чисел (БСВ), полученных с помощью мультипликативно-конгруэнтного датчика:

1. **Средние значения (МО)**:
    - Средние значения для всех трех последовательностей (A, B, C) близки к ожидаемому математическому ожиданию, равному 0.5. Это свидетельствует о том, что датчик генерирует числа, распределенные равномерно на отрезке [0, 1].
2. **Дисперсия (D)**:
    - Рассчитанные значения дисперсии также близки к теоретической дисперсии, которая для равномерного распределения на отрезке [0, 1] составляет 1/12 или приблизительно 0.083. Это указывает на то, что разброс значений вокруг среднего также соответствует ожидаемому.
3. **Качество генерации**:
    - Полученные результаты говорят о том, что реализованный мультипликативно-конгруэнтный датчик обладает хорошим качеством генерации псевдослучайных чисел. Средние значения и дисперсия близки к теоретическим ожидаемым значениям, что указывает на то, что он хорошо моделирует равномерное распределение на отрезке [0, 1].
4. **Эффективность**:
    - Важно отметить, что эффективность работы датчика также зависит от выбора параметров и методов генерации. Настройка параметров (например, длин массивов и методов обновления) может повлиять на качество и скорость генерации последовательностей псевдослучайных чисел.

Результаты моделирования показывают, что реализованный метод генерации псевдослучайных чисел достаточно точен и может быть

использован в различных приложениях, требующих случайных чисел. Однако, для критически важных задач, возможно, потребуется использование более сложных и проверенных методов генерации случайных чисел.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
    import numpy as np
import matplotlib.pyplot as plt
import random

class PeakGenerator:
    def __init__(self, seed=None):
        self.A = [random.randint(0, 2**32 - 1) for _ in range(55)]
        self.B = [random.randint(0, 2**32 - 1) for _ in range(57)]
        self.C = [random.randint(0, 2**32 - 1) for _ in range(58)]
        self.index_A = 0
        self.index_B = 0
        self.index_C = 0

    def peak(self):
        while True:
            self.A[self.index_A] = (
                self.A[(self.index_A - 55) % 55] + self.A[(self.index_A - 24) %
55]) % (2**32)
            self.B[self.index_B] = (
                self.B[(self.index_B - 57) % 57] + self.B[(self.index_B - 7) %
57]) % (2**32)
            self.C[self.index_C] = (
                self.C[(self.index_C - 58) % 58] + self.C[(self.index_C - 19) %
58]) % (2**32)

            # Проверяем биты переноса
            carry_A = self.A[self.index_A] >> 31
            carry_B = self.B[self.index_B] >> 31
            carry_C = self.C[self.index_C] >> 31

            # Если все три бита переноса одинаковы, то тактируем все три
генератора
            if carry_A == carry_B == carry_C:
                if carry_B:
                    yield self.A[self.index_A] / (2 ** 32), self.B[self.index_B]
/ (2 ** 32), self.C[self.index_C] / (2 ** 32)
            # Если нет, тактируем только два совпадающих генератора
            else:
                if carry_A == carry_B:
                    yield self.A[self.index_A] / (2 ** 32), self.B[self.index_B]
/ (2 ** 32), self.C[self.index_C] / (2 ** 32)
                elif carry_A == carry_C:
                    yield self.A[self.index_A] / (2 ** 32), self.B[self.index_B]
/ (2 ** 32), self.C[self.index_C] / (2 ** 32)
                elif carry_B == carry_C:
```

```
                    yield self.A[self.index_A] / (2 ** 32), self.B[self.index_B]
/ (2 ** 32), self.C[self.index_C] / (2 ** 32)

            self.index_A = (self.index_A + 1) % 55
            self.index_B = (self.index_B + 1) % 57
            self.index_C = (self.index_C + 1) % 58

peak_gen = PeakGenerator(seed=55)
sequence = []
for _ in range(100000):
    sequence.append(next(peak_gen.peak()))


# МО и Дисперсия
values_A, values_B, values_C = zip(*sequence)
theoretical_mean = 0.5
theoretical_variance = 1/12
mean_A = np.mean(values_A)
variance_A = np.var(values_A)
mean_B = np.mean(values_B)
variance_B = np.var(values_B)
mean_C = np.mean(values_C)
variance_C = np.var(values_C)

print("Сравнение с теоретическими значениями:")
print("Математическое ожидание (МО):")
print(f"МО для A = {mean_A}")
print(f"уМО  для B = {mean_B}")
print(f"МО  для C = {mean_C}")
print(f"  Теоретическое   = {theoretical_mean}")
print("Дисперсия (D):")
print(f" D рассчитанная для A = {variance_A}")
print(f" D рассчитанная для B = {variance_B}")
print(f" D рассчитанная для C = {variance_C}")
print(f"  Теоретическое  = {theoretical_variance}")

# Гистограмма
counts_A, _ = np.histogram(values_A, bins=10, range=(0, 1))
counts_B, _ = np.histogram(values_B, bins=10, range=(0, 1))
counts_C, _ = np.histogram(values_C, bins=10, range=(0, 1))

frequencies_A = counts_A / len(values_A)
frequencies_B = counts_B / len(values_B)
frequencies_C = counts_C / len(values_C)
x_values = np.linspace(0, 1, num=10, endpoint=True)

# Построение
plt.bar(x_values , frequencies_A, width=0.2, color='green')
plt.bar(x_values , frequencies_B, width=0.2, color='green')
plt.bar(x_values , frequencies_C, width=0.2, color='green')
plt.xlabel('Значение')
```

```
plt.ylabel('Частота')
plt.title('Гистограмма распределения псевдослучайных величин')
plt.legend()
plt.show()

# Функция для вычисления R_hat(T)
def calculate_R_hat(T, s, z):
    R = []
    t = np.arange(0, T)
    for s_val in s:
        R.clear()
        for k in range(T):
            R.append(0)
            for i in range(t[k] - s_val):
                R[k] += z[i] * z[i+s_val]
            R[k] *= 12 / (t[k] - s_val)
            R[k] -= 3
        plt.plot(t, R, label=f's={s_val}')

# Задание параметров
T = 1000
s = [2, 5, 10]
z = np.random.rand(T)

# Создание графика
plt.figure(figsize=(10, 6))
calculate_R_hat(T, s, z)
plt.title('Графики зависимости R_hat(T) для разных значений s')
plt.xlabel('T')
plt.ylabel('R_hat(T)')
plt.legend()
plt.grid(True)
plt.show()
```