Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет
телекоммуникаций и информатики»

Кафедра вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к нулевой лабораторной работе по дисциплине
«Моделирование»

Выполнил:
студент гр. ИВ-621
Евтушенко Д.А

Проверила:
Ассистент кафедры ВС
Петухова Я.В.

Новосибирск, 2020

Содержание

Задание на проектирование 3
Теория по проектной части 3
Пример работы программы 5
Выводы по результатам проектирования 8
Листинг программы 9
Список используемой литературы 11

Задание на проектирование

Убедиться в равномерности генератора псевдослучайных чисел, используя параметры «хи-квадрат» и автокорреляция.

Теория по проектной части

• Проверка по критерию «хи-квадрат»

У нас есть нулевая гипотеза, она заключается в следующем: Мы ожидаем, что компилятор случайных чисел имеет равномерное распределение, то есть те результаты, которые мы хотим получить (фактические) не будут противоречить ожидаемым. Если этот так, то разброс будет относительно небольшим, в пределах случайных колебаний. Критерий Хи-квадрат или критерий согласия Пирсона, позволяет оценить значимость различий между фактическим (выявленным в результате исследования) количеством, попадающих в каждую категорию, и теоретическим количеством, которое можно ожидать в изучаемых группах при справедливости нулевой гипотезы.

pi – теоретическая вероятность попадания чисел в i-ый интервал (всего этих интервалов k); она равна $pi = \frac{1}{k}$

N – общее количество сгенерированных чисел

n_i– количество попаданий в i-ый интервал

χ^2– критерий, позволяющий определить, удовлетворяет ли генератор случайных чисел требованиям равномерного распределения или нет

Процедура проверки имеет следующий вид.

1. Диапазон от 0 до 1 разбивается на k равных интервалов.
2. Запускается ГСЧ N раз (N должно быть велико, например, $\frac{N}{k} > 5$).
3. Определяется количество случайных чисел, попавших в каждый интервал
4. Вычисляется экспериментальное значение χ^2 по следующей формуле:

$$\chi^2 = \frac{(n_1 - p_1 * N)^2}{p_1 * N} + \frac{(n_2 - p_2 * N)^2}{p_2 * N} + \dots + \frac{(n_k - p_k * N)^2}{p_k * N}$$

$$\chi^2 = \sum_{i=1}^{k} \frac{(n_k - p_k * N)^2}{p_k * N} = \frac{1}{N} \sum_{i=1}^{k} \left(\frac{{n_i}^2}{p_i}\right) - N$$

• **Проверка по критерию автокорреляции**

$Ex = \bar{x}$ – математическое ожидание

s^2 - выборочная дисперсия

$\hat{a}(\tau)$ – автокорреляция

x_i – множество псевдослучайных чисел

τ - смещение

$$Ex = \frac{1}{N}\sum_{i=1}^{N} x_i = \bar{x}$$

$$s_n^2 = \sum \frac{{x_i}^2}{N} - \bar{x}^2$$

$$\hat{a}(\tau) = \frac{1}{(N-\tau) * s^2}\sum_{i=1}^{N-\tau}(x_i - \bar{x})(x_{i+\tau} - \bar{x})$$

- **Используемые генераторы**
  1. Компилятор случайных чисел из библиотек Random. Который использует алгоритм PRNG под названием Mersenne Twister в качестве корневого генератора. **Вихрь Мерсе́нна** (англ. Mersenne twister, MT) — генератор псевдослучайных чисел (ГПСЧ), основывающийся на свойствах простых чисел Мерсенна и обеспечивающий быструю генерацию высококачественных по критерию случайности псевдослучайных чисел.
  2. Компилятор случайных чисел из библиотеки NumPy основанный на алгоритме PCG64. **PCG64**– это 128-битная реализация конгруэнтного генератора перестановок О'Нила. Разработан в 2014 году, с помощью функции перестановок улучшает статистические свойства линейного конгруэнтного генератора по модулю 2.
  3. Компилятор случайных чисел из библиотеки NumPy основанный на алгоритме **SFC64**. Это 256-битная реализация малого быстрого хаотического PRNG Криса Доти-Хамфри. **SFC64** включает 64-разрядный счетчик который значит что абсолютная минимальная длина цикла является 2^{64}, и что различные инициализации генератора не будут сталкиваться друг с другом по крайней 2^{64}мере в течение нескольких итераций.

Пример работы программы:

Для критерия «хи-квадрат»:

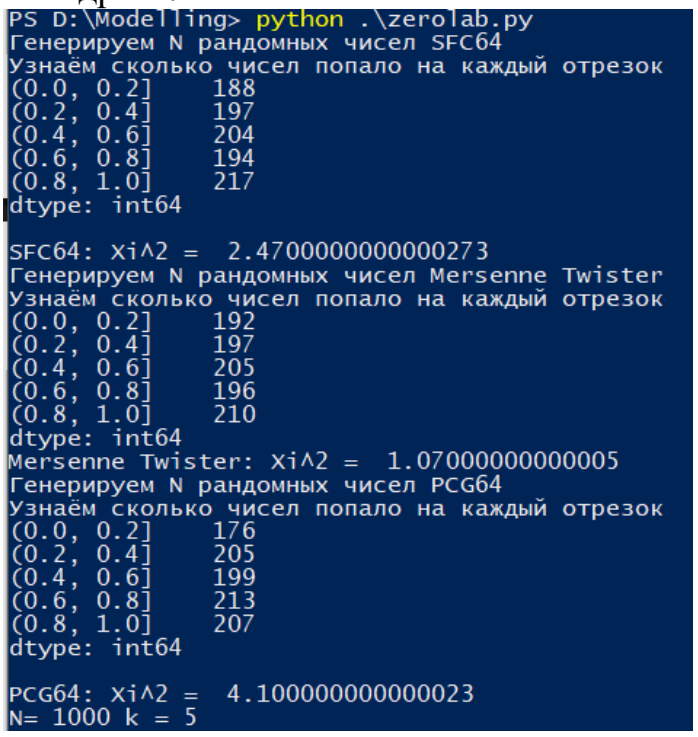

Рисунок 1. Работа программы при значениях **N=1000, k = 5**

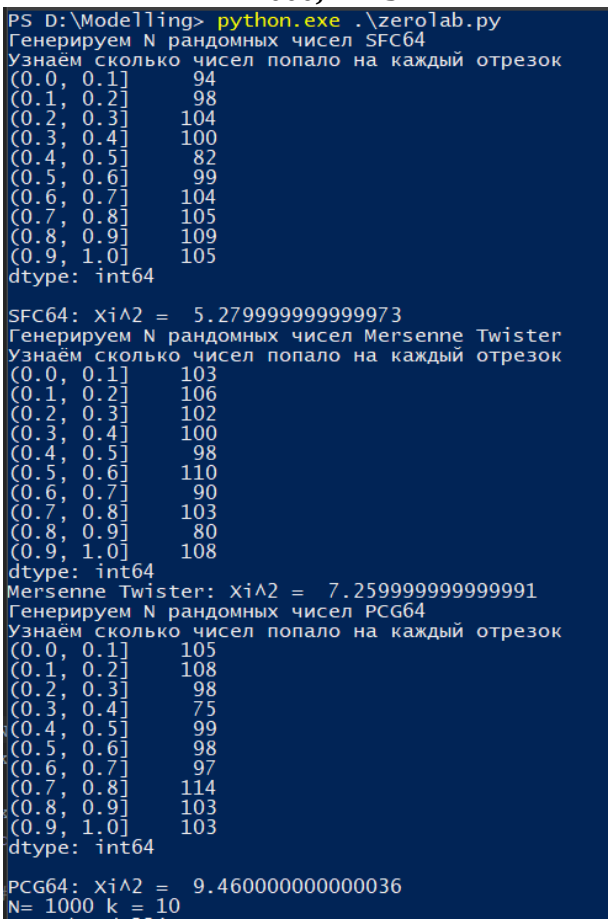

Рисунок 2 Работа программы при значениях **N=1000, k = 10**

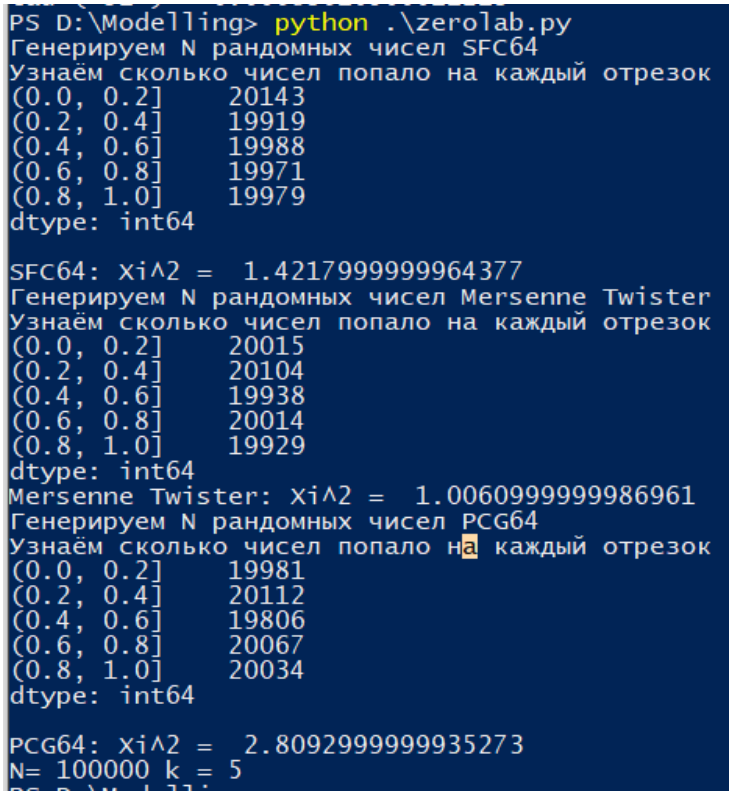

Рисунок 3 Работа программы при значениях **N=100000, k = 5**

Для критерия автокорреляции:

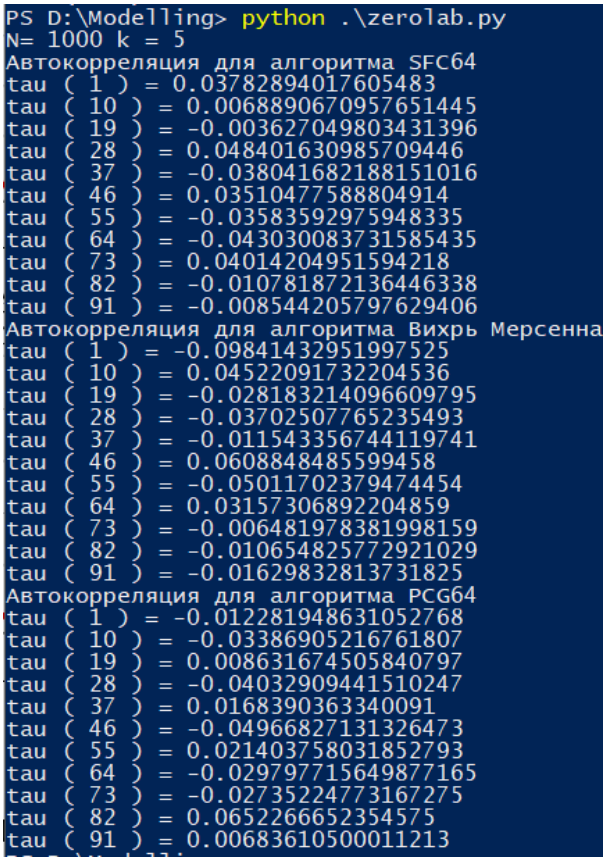

Рисунок 4 Работа программы при значениях **N=1000, k = 5**

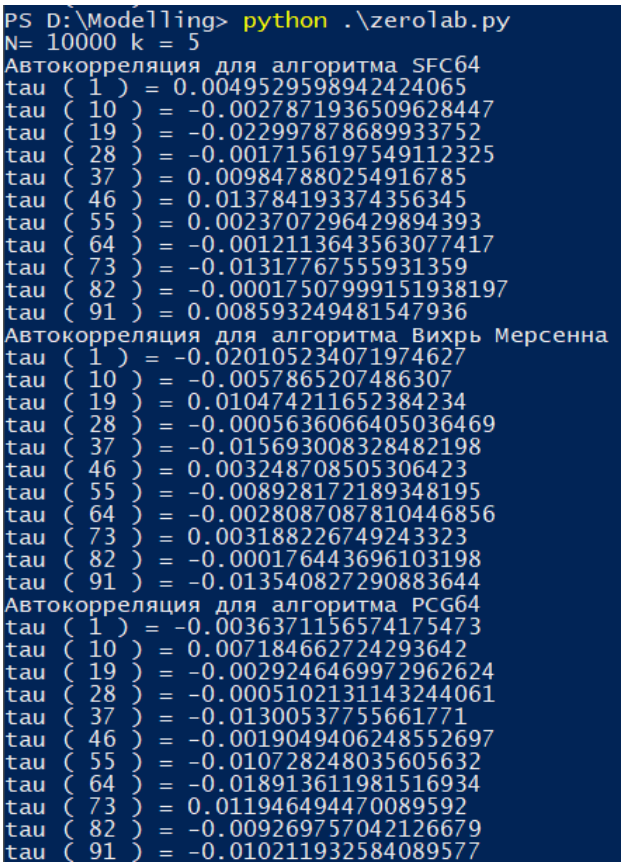

Рисунок 5 Работа программы при значениях **N=10000, k = 5**

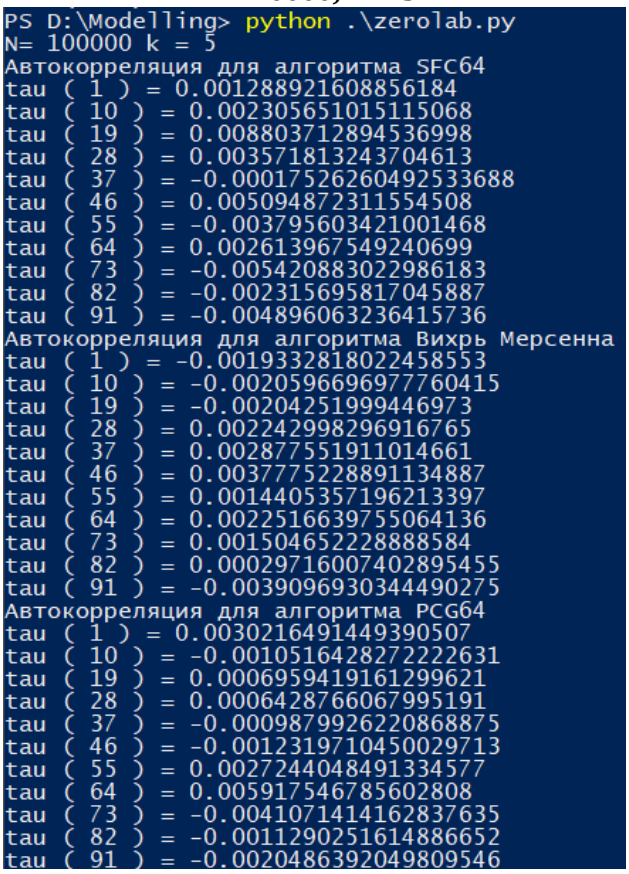

Рисунок 6 Работа программы при значениях **N=100000, k = 5**

Выводы по результатам проектирования

От качества работы ГСЧ зависит качество работы всей системы и точность результатов. Поэтому случайная последовательность, порождаемая ГСЧ, должна удовлетворять целому ряду критериев. В данной работе мы проверили на качество генераторы случайных чисел основанных на алгоритмах Mersenne Twister, PCG64, SFC64 для языка программирования python по двум критериям: Хи-квадрат распределение или критерий согласия Пирсона, и по критерию автокорреляции. Получив значения Хи квадрат и зная число степеней свободы(m = k – r – 1, где k – это количество интервалов, а r – это количество параметров в конкретной функции распределения, для равномерного 2 параметра) можно обратиться к таблице значений Хи-квадрат и определить меру разброса. Чтобы различать значимые и незначимые результаты, обычно используют уровень a=0.05. Критическое табличное значение при 2 степенях свободы и уровне значимости 0.05 = **5.99**. Сравним фактические значения(N=1000, k = 5), полученные для 3 разных компиляторов SFC64 = **2.47,** Mersenne Twister = **1.07,** PCG64 = **4.10** с табличным значением. Все 3 компилятора показали расчетные критерии меньше чем табличные, но генератор, основанный на алгоритме Mersenne Twister, показал лучшие результаты, так как его отклонение полученных значений от теоретических оказалось больше. Из этого можно сделать вывод, что гипотеза о равенстве (согласии) частот не отклоняется. Следовательно, нулевая гипотеза для всех 3 ГСЧ подтверждена. Коэффициенты автокорреляции принимают как положительные, так и отрицательные значения в окрестности нуля. Это говорит о том, что статистическая взаимосвязь между исходной и сдвинутой последовательностью имеет очень маленькую погрешность. Из этого можно сделать вывод, что числа генерируемые компилятором близки к случайным.

Листинг программы

```
import random

import _random
import pandas as pn
import numpy as np
import time
from numpy.random import Generator, SFC64, PCG64

def xi2(N, k):
    sum = 0
    V = 0
    pi = 1/k
    np.random.seed(int(time.time()))
    print("Генерируем N рандомных чисел SFC64")
    series = pn.Series([Generator(SFC64()).random() for s in range(N)])
    # print("NumPy Random = ", series)

    print("Узнаём сколько чисел попало на каждый отрезок")
    bins = np.linspace(0, 1, k+1)

    res = series.groupby(pn.cut(series, bins=bins)).count()
    print(res)
    ymass = []
    for i in res:
        ymass.append(i)
        sum += (i * i) / pi

    V = (sum / N) - N

    random.seed()
    print("\nSFC64: Xi^2 = ", V)

    sum = 0
    print("Генерируем N рандомных чисел Mersenne Twister")
    massiv = []
    for i in range(N):
        massiv.append(random.random())

    series2 = pn.Series(massiv)
    # print("Random = ", series2)

    print("Узнаём сколько чисел попало на каждый отрезок")
    bins1 = np.linspace(0, 1, k + 1)

    res1 = series2.groupby(pn.cut(series2, bins=bins1)).count()
    print(res1)
    ymass1 = []
    for i in res1:
        ymass1.append(i)
        sum += (i * i) / pi

    V1 = (sum / N) - N
    print("Mersenne Twister: Xi^2 = ", V1)
    sum = 0
    print("Генерируем N рандомных чисел PCG64")
    series = pn.Series([Generator(PCG64()).random() for s in range(N)])
    # print("NumPy Random = ", series)

    print("Узнаём сколько чисел попало на каждый отрезок")
    bins = np.linspace(0, 1, k + 1)
```

```
    res = series.groupby(pn.cut(series, bins=bins)).count()
    print(res)
    ymass = []
    for i in res:
        ymass.append(i)
        sum += (i * i) / pi

    V = (sum / N) - N

    random.seed()
    print("\nPCG64: Xi^2 = ", V)

def new_cor(series, N):
    math = 0
    for i in range(1, N):
        math += series[i]

    math = math / N
    # print(math)
    disp = 0
    for i in range(1, N):
        disp += series[i] * series[i]

    disp = (disp / N) - math ** 2
    # print(disp)
    sum = 0
    tau_mass = []
    for tau in range(1, 100, 9):
        for i in range(1, N - tau):
            sum += (series[i] - math) * (series[i + tau] - math)
        tau_mass.append(sum / (disp * (N - tau)))
        print("tau (",tau, ") =", sum / (disp * (N - tau)))
        tau = 0
        # print(tau_mass)
    return np.mean(tau_mass)

def atocor(N, k):
    np.random.seed(int(time.time()))
    series = pn.Series([Generator(SFC64()).random() for s in range(N)])
    # print("Генерируем N рандомных чисел с помощью  Numpy random")
    # print(series)

    print("Автокорреляция для алгоритма SFC64 ")
    new_cor(series, N)

    # print("\nГенерируем %N рандомных чисел Вихрь Мерсенна")
    massiv = []
    for i in range(N):
        massiv.append(random.random())

    series2 = pn.Series(massiv)
    # print("Random = ", series2)
    print("Автокорреляция для алгоритма Вихрь Мерсенна")
    new_cor(series2, N)
    # print("\nГенерируем N рандомных чисел PCG64")
    massiv1 = []
    for i in range(N):
        massiv1.append(random.SystemRandom().random())

    series3 = pn.Series([Generator(PCG64()).random() for s in range(N)])

    print("Автокорреляция для алгоритма PCG64")

    new_cor(series3, N)
```

Список использованной литературы

1. http://stratum.ac.ru/education/textbooks/modelir/lection22.html
2. https://ru.wikipedia.org/wiki/Критерий_согласия_Пирсона
3. https://statanaliz.info/statistica/proverka-gipotez/kriterij-soglasiya-pirsona-khi-kvadrat/
4. https://www.intuit.ru/studies/courses/623/479/lecture/21088?page=3
5. https://ru.wikipedia.org/wiki/Автокорреляция