Федеральное государственное бюджетное образовательное учреждение «Сибирский государственный университет телекоммуникаций и информатики»

Кафедра ВС

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К нулевой лабораторной работе по дисциплине «Моделирование»

Выполнил:
студент гр. ИВ-621
Дьяченко Д. В.
Проверила:
Ассистент кафедры ВС
Петухова Я.В.

Новосибирск, 2020

Оглавление

Задание на проектирование

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

Теория по проектной части

- Проверка по критерию «хи-квадрат»

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

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:

1. Диапазон от 0 до 1 разбивается на k равных интервалов
2. С помощью генератора случайных чисел получаем N чисел (по условию $\frac{N}{k} > 5$)
3. Определяется количество случайных чисел, попавших в каждый интервал
4. Вычисляется экспериментальное значение χ^2 по следующей формуле:

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

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

- Проверки по критерию автокорреляции

$Ex = \frac{1}{n}\sum_{i=1}^{N} x_i = \bar{x}$ – математическое ожидание

$S^2 = Ex^2 - (Ex)^2$ – выборочная дисперсия

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

x – множество случайных чисел

y – множество случайных чисел со смещением

Для проектирования были взяты следующие генераторы случайных чисел:

1. MT1997 - 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. PCG64 – 128-битная имплементация перестановочного конгруэнтного генератора.
3. Philox — это 64-битный генератор, который использует конструкцию на основе счетчика, основанную на более слабых (и более быстрых) версиях криптографических функций.

Результаты проектирования

```
Intervals:
from 0.00 to 0.20 = 221
from 0.20 to 0.40 = 204
from 0.40 to 0.60 = 207
from 0.60 to 0.80 = 175
from 0.80 to 1.00 = 193
MT1997 generator N=1000 k=5 chi_sqr=5.9000

Intervals:
from 0.00 to 0.20 = 201
from 0.20 to 0.40 = 226
from 0.40 to 0.60 = 168
from 0.60 to 0.80 = 185
from 0.80 to 1.00 = 220
PCG64 generator N=1000 k=5 chi_sqr=11.6300

Intervals:
from 0.00 to 0.20 = 188
from 0.20 to 0.40 = 197
from 0.40 to 0.60 = 215
from 0.60 to 0.80 = 228
from 0.80 to 1.00 = 172
Philox generator N=1000 k=5 chi_sqr=9.7300

```

Рисунок 1 – результаты проверки трех ГСЧ по критерию «хи-квадрат» с параметрами N, равной 1000, и k, равной 5.

```
Intervals:
from 0.00 to 0.10 = 125
from 0.10 to 0.20 = 96
from 0.20 to 0.30 = 101
from 0.30 to 0.40 = 103
from 0.40 to 0.50 = 101
from 0.50 to 0.60 = 106
from 0.60 to 0.70 = 86
from 0.70 to 0.80 = 89
from 0.80 to 0.90 = 98
from 0.90 to 1.00 = 95
MT1997 generator N=1000 k=10 chi_sqr=10.3400

Intervals:
from 0.00 to 0.10 = 111
from 0.10 to 0.20 = 90
from 0.20 to 0.30 = 104
from 0.30 to 0.40 = 122
from 0.40 to 0.50 = 93
from 0.50 to 0.60 = 75
from 0.60 to 0.70 = 101
from 0.70 to 0.80 = 84
from 0.80 to 0.90 = 118
from 0.90 to 1.00 = 102
PCG64 generator N=1000 k=10 chi_sqr=19.8000

Intervals:
from 0.00 to 0.10 = 93
from 0.10 to 0.20 = 95
from 0.20 to 0.30 = 103
from 0.30 to 0.40 = 94
from 0.40 to 0.50 = 118
from 0.50 to 0.60 = 97
from 0.60 to 0.70 = 118
from 0.70 to 0.80 = 110
from 0.80 to 0.90 = 95
from 0.90 to 1.00 = 77
Philox generator N=1000 k=10 chi_sqr=14.3000
```

Рисунок 2 – результаты проверки трех ГСЧ по критерию «хи-квадрат» с параметрами N, равной 1000, и k, равной 10.

```
Intervals:
from 0.00 to 0.10 = 952
from 0.10 to 0.20 = 1025
from 0.20 to 0.30 = 965
from 0.30 to 0.40 = 1005
from 0.40 to 0.50 = 1038
from 0.50 to 0.60 = 957
from 0.60 to 0.70 = 1008
from 0.70 to 0.80 = 1037
from 0.80 to 0.90 = 1009
from 0.90 to 1.00 = 1004
MT1997 generator N=10000 k=10 chi_sqr=9.0020

Intervals:
from 0.00 to 0.10 = 998
from 0.10 to 0.20 = 1020
from 0.20 to 0.30 = 989
from 0.30 to 0.40 = 1051
from 0.40 to 0.50 = 1036
from 0.50 to 0.60 = 954
from 0.60 to 0.70 = 991
from 0.70 to 0.80 = 987
from 0.80 to 0.90 = 1009
from 0.90 to 1.00 = 965
PCG64 generator N=10000 k=10 chi_sqr=8.0940

Intervals:
from 0.00 to 0.10 = 1045
from 0.10 to 0.20 = 973
from 0.20 to 0.30 = 961
from 0.30 to 0.40 = 1051
from 0.40 to 0.50 = 936
from 0.50 to 0.60 = 988
from 0.60 to 0.70 = 999
from 0.70 to 0.80 = 953
from 0.80 to 0.90 = 1043
from 0.90 to 1.00 = 1051
Philox generator N=10000 k=10 chi_sqr=17.7760

```

Рисунок 3 – результаты проверки трех ГСЧ по критерию «хи-квадрат» с параметрами N, равной 10000, и k, равной 10.

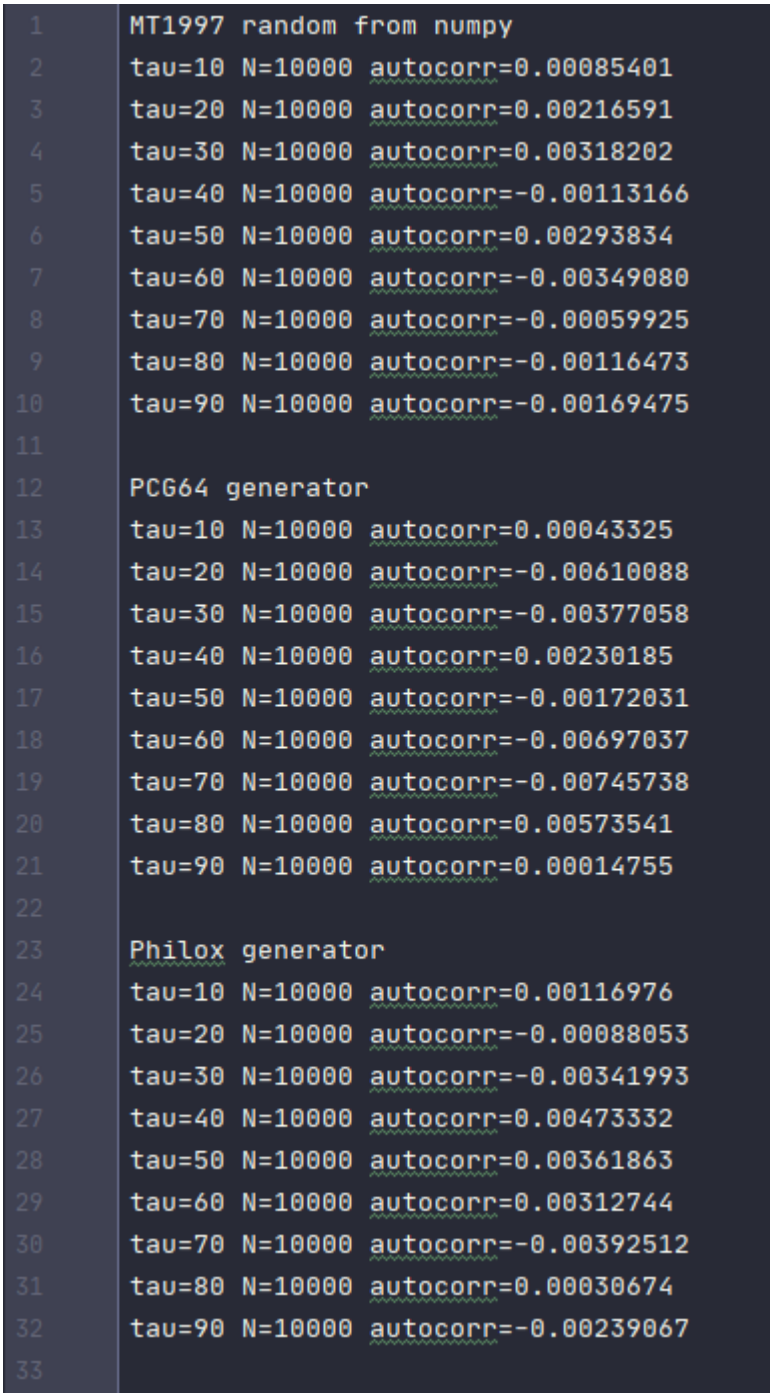

Рисунок 4 – результаты проверки трех ГСЧ по критерию «автокорреляция» с параметрами N равной 10000.

```
MT1997 random from numpy
tau=10 N=1000000 autocorr=-0.00020968
tau=20 N=1000000 autocorr=-0.00005446
tau=30 N=1000000 autocorr=-0.00007635
tau=40 N=1000000 autocorr=0.00040714
tau=50 N=1000000 autocorr=-0.00052770
tau=60 N=1000000 autocorr=-0.00029218
tau=70 N=1000000 autocorr=-0.00005377
tau=80 N=1000000 autocorr=-0.00008215
tau=90 N=1000000 autocorr=-0.00045079

PCG64 generator
tau=10 N=1000000 autocorr=-0.00013370
tau=20 N=1000000 autocorr=0.00033251
tau=30 N=1000000 autocorr=0.00003752
tau=40 N=1000000 autocorr=-0.00014502
tau=50 N=1000000 autocorr=-0.00057780
tau=60 N=1000000 autocorr=0.00049858
tau=70 N=1000000 autocorr=0.00008709
tau=80 N=1000000 autocorr=-0.00026331
tau=90 N=1000000 autocorr=0.00023849

Philox generator
tau=10 N=1000000 autocorr=0.00059910
tau=20 N=1000000 autocorr=0.00006492
tau=30 N=1000000 autocorr=-0.00001133
tau=40 N=1000000 autocorr=0.00028878
tau=50 N=1000000 autocorr=0.00022970
tau=60 N=1000000 autocorr=-0.00069348
tau=70 N=1000000 autocorr=0.00061003
tau=80 N=1000000 autocorr=-0.00079371
tau=90 N=1000000 autocorr=-0.00016312
```

Рисунок 5 – результаты проверки трех ГСЧ по критерию «автокорреляция» с параметрами N равной 1000000.

Выводы

Для анализа критерия «хи-квадрат» по таблице было выбрано значение нормали равное 14.1 по параметрам $k = 10 - 2 - 1 = 7$ (число степеней свободы, которое считается по формуле $k = m - r - 1$, где m – это количество интервалов, а r – это количество параметров в конкретной функции

распределения) и $a = 0.05$ (уровень значимости, выбранное случайным образом).

Из полученных результатов по критерию «хи-квадрат» можно сделать следующие выводы:

- Для генератора MT1997, показал результаты ниже критического в 14.1. Следовательно можно сделать вывод, что ни увеличение количества точек, ни увеличение количества интервалов, не повлияло на его распределение. Другими словами, результаты подтверждают, что данный генератор генерирует последовательность чисел, подчиняющуюся равномерному распределению.
- Для генератора PCG64, показал значения как ниже критического, так и выше, хоть и не значительно. Превышение критического значения наблюдается при увеличении количества интервалов. Из чего можно сделать вывод, что данный генератор выдает последовательность чисел, подчиняющуюся равномерному распределению с некой погрешностью.
- Для генератора Philox, показал значения как ниже критического, так и выше. Повышенные значения наблюдаются как при увеличении количества точек, так и при увеличении количества интервалов. Из чего можно сделать вывод, что данный генератор выдает последовательность чисел, не подчиняющуюся равномерному распределению.

Коэффициенты корреляции могут быть положительными или отрицательными. В первом случае предполагается, что мы можем определить только наличие или отсутствие связи, а во втором — также и её направление. Если предполагается, что на значениях переменных задано отношение строгого порядка, то отрицательная корреляция — корреляция, при которой увеличение одной переменной связано с уменьшением другой. При этом коэффициент корреляции будет отрицательным. Положительная корреляция в таких условиях — это такая связь, при которой увеличение одной переменной

связано с увеличением другой переменной. Возможна также ситуация отсутствия статистической взаимосвязи — например, для независимых случайных величин. По результатам проектирования можно увидеть, что все полученные значения от всех трех генераторов находятся около 0, а значит все исследуемые генераторы выдают независимые случайные величины.

Приложение

1. Листинг программы

```
import random
import math
import sys
from numpy.random import Philox, PCG64, Generator
from numpy import random as nprandom


result_chi_square = open("result_chi_square.txt", "w")
result_autocorr = open("result_autocorr.txt", "w")


def compute_chi_square(N=100, k=10):
    step = 1 / k
    # print(f"k={k} step={step} N={N}")

    randlist = [(random.randrange(0, sys.maxsize) / sys.maxsize) for i in range(N)]
    intervals = {i: 0 for i in range(k)}

    for i in range(N):
        for j in range(k):
            if randlist[i] < ((j + 1) * step):
                intervals[j] += 1
                break
    # print(f"intervals={intervals}")

    # result_chi_square.write("intervals:\n")
    # for i in range(k):
    #     result_chi_square.write(f"from {step * i}-{step * (i + 1)} {intervals[i]}\n")

    acc = 0.0
    for i in range(k):
        acc += math.pow(intervals[i], 2) * k

    return (acc / N) - N


def compute_chi_square(randlist, k=10):
    N = len(randlist)
    step = 1 / k
    # print(f"k={k} step={step} N={N}")

    # randlist = [(random.randrange(0, sys.maxsize) / sys.maxsize) for i in range(N)]
    intervals = {i: 0 for i in range(k)}

    for i in range(N):
        for j in range(k):
            if randlist[i] < ((j + 1) * step):
                intervals[j] += 1
                break
```

```
    # print(f"intervals={intervals}")

    result_chi_square.write("Intervals:\n")
    for i in range(k):
        # result_chi_square.write(f"from {step * i} to {step * (i + 1)} = {intervals[i]}\n")
        result_chi_square.write("from {0:.2f} to {1:.2f} = {2:d}\n".format(step * i, step * (i + 1), intervals[i]))

    acc = 0.0
    for i in range(k):
        acc += math.pow(intervals[i], 2) * k

    return (acc / N) - N


def math_expect(list):
    acc = 0.0
    n = len(list)
    for i in range(n - 1):
        acc += list[i]
    return acc / n


def dispersion(list, x):
    return math_expect(list) * math.pow(x, 2) - math.pow(math_expect(list) - x, 2)


def autocorrelation(N=100, offset=0):
    list = [random.randrange(0, sys.maxsize) for i in range(N)]
    expect = math_expect(list)

    upValue = 0.0
    for i in range(N - offset):
        upValue += (list[i] - expect) * (list[i + offset] - expect)

    bottomValue = 0.0
    for i in range(N):
        bottomValue += math.pow(list[i] - expect, 2)

    return upValue / bottomValue


def autocorrelation2(x_i, tau=500):
    N = len(x_i)
    sum_x_i = 0.0

    for i in range(N):
        sum_x_i += x_i[i]

    sum_x_i_sqr = sum_x_i / N
    ksi_sqr = sum_x_i_sqr / N - math.pow(sum_x_i_sqr, 2.0)

    need_print = True
```

```
101.
102.     for i in range(tau):
103.         acc = 0.0
104.         for j in range(1, N - tau):
105.             acc += ((x_i[j] - sum_x_i_sqr) * (x_i[j + tau] - sum_x_i_sqr)) / ksi_sqr
106.         if need_print:
107.             result_autocorr.write("tau={0:d} N={1:d} autocorr={2:.8f}\n".format(tau, N, acc / (N - tau)))
108.             need_print = False
109.     # result_autocorr.write("\n")
110.
111.
112. def autocorrelation(list, offset=0):
113.     N = len(list)
114.     if N < 1:
115.         return 0
116.     expect = math_expect(list)
117.
118.     upValue = 0.0
119.     for i in range(N - offset):
120.         upValue += (list[i] - expect) * (list[i + offset] - expect)
121.
122.     bottomValue = 0.0
123.     for i in range(N):
124.         bottomValue += math.pow(list[i] - expect, 2)
125.
126.     return upValue / bottomValue
127.
128.
129. def test_chi_square():
130.     for r in range(3):
131.         for N in [10000]:
132.             for k in [10]:
133.                 result_chi_square.write("k={0:d} N={1:d} chi={2:.5f}\n".format(k, N, compute_chi_square(N, k)))
134.             result_chi_square.write("\n")
135.         result_chi_square.write("\n")
136.
137. def test_autocorr(N=10, k=10, offset=50):
138.     step = 1 / k
139.
140.     result_autocorr.write(f"k={k} step={step} N={N}\n")
141.
142.     randlist = [(random.randrange(0, sys.maxsize) / sys.maxsize) for i in range(N)]
143.     intervals = [[] for i in range(k)]
144.
145.     for i in range(N):
146.         for j in range(k):
147.             if randlist[i] < ((j + 1) * step):
148.                 intervals[j].append(randlist[i])
149.                 break
150.
151.     for i in range(k):
152.         result_autocorr.write("{0:.2f}-{1:.2f} = {2:d}\n".format(step * i, step * (i + 1), len(intervals[i])))
```

```
153.     result_autocorr.write("\n")
154.
155.     result_autocorr.write(f"autocorrelation = {autocorrelation(randlist, offset)}\n\n")
156.
157. def test_autocorr3(randlist, k=10, offset=50):
158.     N = len(randlist)
159.     step = 1 / k
160.
161.     result_autocorr.write(f"k={k} step={step} N={N}\n")
162.
163.     # randlist = [(random.randrange(0, sys.maxsize) / sys.maxsize) for i in range(N)]
164.     intervals = [[] for i in range(k)]
165.
166.     for i in range(N):
167.         for j in range(k):
168.             if randlist[i] < ((j + 1) * step):
169.                 intervals[j].append(randlist[i])
170.                 break
171.
172.     for i in range(k):
173.         result_autocorr.write("{0:.2f}-{1:.2f} = {2:d}\n".format(step * i, step * (i + 1), len(intervals[i])))
174.     result_autocorr.write("\n")
175.
176.     result_autocorr.write(f"autocorrelation = {autocorrelation(randlist, offset)}\n\n")
177.
178. def test_autocorrelation():
179.     offset = 50
180.     test_autocorr(100, 10, offset)
181.     test_autocorr(1000, 10, offset)
182.     test_autocorr(100, 5, offset)
183.
184.
185. def test_autocorrelation2():
186.     test_autocorr3(N=10000)
187.     test_autocorr3(N=1000000)
188.
189. def test_autocorr3(N=100):
190.     result_autocorr.write("MT1997 random from numpy\n")
191.     randlist = [nprandom.random() for i in range(N)]
192.     for tau in range(0, 100, 10):
193.         autocorrelation2(randlist, tau)
194.
195.     result_autocorr.write("\nPCG64 generator\n")
196.     randlist = [Generator(PCG64()).random() for i in range(N)]
197.     for tau in range(0, 100, 10):
198.         autocorrelation2(randlist, tau)
199.
200.     result_autocorr.write("\nPhilox generator\n")
201.     randlist = [Generator(Philox()).random() for i in range(N)]
202.     for tau in range(0, 100, 10):
203.         autocorrelation2(randlist, tau)
204.
```

```
205.  def test_chi_square2():
206.      N = 1000
207.      k = 5
208.
209.      randlist1 = [nprandom.random() for i in range(N)]
210.      result_chi_square.write("MT1997 generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist1, k)))
211.
212.      randlist2 = [Generator(PCG64()).random() for i in range(N)]
213.      result_chi_square.write("PCG64 generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist2, k)))
214.
215.      randlist3 = [Generator(Philox()).random() for i in range(N)]
216.      result_chi_square.write("Philox generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist3, k)))
217.
218.      k = 10
219.
220.      result_chi_square.write("MT1997 generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist1, k)))
221.      result_chi_square.write("PCG64 generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist2, k)))
222.      result_chi_square.write("Philox generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist3, k)))
223.
224.      N = 10000
225.
226.      randlist = [nprandom.random() for i in range(N)]
227.      result_chi_square.write("MT1997 generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist, k)))
228.
229.      randlist = [Generator(PCG64()).random() for i in range(N)]
230.      result_chi_square.write("PCG64 generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist, k)))
231.
232.      randlist = [Generator(Philox()).random() for i in range(N)]
233.      result_chi_square.write("Philox generator N={0:d} k={1:d} chi_sqr={2:.4f}\n\n".format(N, k, compute_chi_square(randlist, k)))
234.
235.
236.  if __name__ == "__main__":
237.      # test_chi_square()
238.      # test_autocorrelation()
239.      test_chi_square2()
240.      test_autocorrelation2()
```