

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.

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 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:

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

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

• 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 Мерсённа** (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»:

```
PS D:\Modelling> python .\zerolab.py
Генерируем N случайных чисел SFC64
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.2] 188
(0.2, 0.4] 197
(0.4, 0.6] 204
(0.6, 0.8] 194
(0.8, 1.0] 217
dtype: int64

SFC64:  $\chi^2 = 2.4700000000000273$ 
Генерируем N случайных чисел Mersenne Twister
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.2] 192
(0.2, 0.4] 197
(0.4, 0.6] 205
(0.6, 0.8] 196
(0.8, 1.0] 210
dtype: int64

Mersenne Twister:  $\chi^2 = 1.070000000000005$ 
Генерируем N случайных чисел PCG64
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.2] 176
(0.2, 0.4] 205
(0.4, 0.6] 199
(0.6, 0.8] 213
(0.8, 1.0] 207
dtype: int64

PCG64:  $\chi^2 = 4.100000000000023$ 
N= 1000 k = 5
```

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

```
PS D:\Modelling> python.exe .\zerolab.py
Генерируем N случайных чисел SFC64
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.1] 94
(0.1, 0.2] 98
(0.2, 0.3] 104
(0.3, 0.4] 100
(0.4, 0.5] 82
(0.5, 0.6] 99
(0.6, 0.7] 104
(0.7, 0.8] 105
(0.8, 0.9] 109
(0.9, 1.0] 105
dtype: int64

SFC64:  $\chi^2 = 5.279999999999973$ 
Генерируем N случайных чисел Mersenne Twister
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.1] 103
(0.1, 0.2] 106
(0.2, 0.3] 102
(0.3, 0.4] 100
(0.4, 0.5] 98
(0.5, 0.6] 110
(0.6, 0.7] 90
(0.7, 0.8] 103
(0.8, 0.9] 80
(0.9, 1.0] 108
dtype: int64

Mersenne Twister:  $\chi^2 = 7.259999999999991$ 
Генерируем N случайных чисел PCG64
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.1] 105
(0.1, 0.2] 108
(0.2, 0.3] 98
(0.3, 0.4] 75
(0.4, 0.5] 99
(0.5, 0.6] 98
(0.6, 0.7] 97
(0.7, 0.8] 114
(0.8, 0.9] 103
(0.9, 1.0] 103
dtype: int64

PCG64:  $\chi^2 = 9.460000000000036$ 
N= 1000 k = 10
```

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

```

PS D:\Modelling> python .\zerolab.py
Генерируем N случайных чисел SFC64
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.2] 20143
(0.2, 0.4] 19919
(0.4, 0.6] 19988
(0.6, 0.8] 19971
(0.8, 1.0] 19979
dtype: int64

SFC64:  $\chi^2 = 1.421799999964377$ 
Генерируем N случайных чисел Mersenne Twister
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.2] 20015
(0.2, 0.4] 20104
(0.4, 0.6] 19938
(0.6, 0.8] 20014
(0.8, 1.0] 19929
dtype: int64

Mersenne Twister:  $\chi^2 = 1.006099999986961$ 
Генерируем N случайных чисел PCG64
Узнаём сколько чисел попало на каждый отрезок
(0.0, 0.2] 19981
(0.2, 0.4] 20112
(0.4, 0.6] 19806
(0.6, 0.8] 20067
(0.8, 1.0] 20034
dtype: int64

PCG64:  $\chi^2 = 2.8092999999935273$ 
N= 100000 k = 5

```

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

Для критерия автокорреляции:

```

PS D:\Modelling> python .\zerolab.py
N= 1000 k = 5
Автокорреляция для алгоритма SFC64
tau ( 1 ) = 0.03782894017605483
tau ( 10 ) = 0.0068890670957651445
tau ( 19 ) = -0.003627049803431396
tau ( 28 ) = 0.048401630985709446
tau ( 37 ) = -0.038041682188151016
tau ( 46 ) = 0.03510477588804914
tau ( 55 ) = -0.03583592975948335
tau ( 64 ) = -0.043030083731585435
tau ( 73 ) = 0.04014204951594218
tau ( 82 ) = -0.010781872136446338
tau ( 91 ) = -0.008544205797629406
Автокорреляция для алгоритма Вихрь Мерсенна
tau ( 1 ) = -0.09841432951997525
tau ( 10 ) = 0.04522091732204536
tau ( 19 ) = -0.028183214096609795
tau ( 28 ) = -0.03702507765235493
tau ( 37 ) = -0.011543356744119741
tau ( 46 ) = 0.0608848485599458
tau ( 55 ) = -0.05011702379474454
tau ( 64 ) = 0.03157306892204859
tau ( 73 ) = -0.006481978381998159
tau ( 82 ) = -0.010654825772921029
tau ( 91 ) = -0.01629832813731825
Автокорреляция для алгоритма PCG64
tau ( 1 ) = -0.012281948631052768
tau ( 10 ) = -0.03386905216761807
tau ( 19 ) = 0.008631674505840797
tau ( 28 ) = -0.04032909441510247
tau ( 37 ) = 0.0168390363340091
tau ( 46 ) = -0.04966827131326473
tau ( 55 ) = 0.021403758031852793
tau ( 64 ) = -0.029797715649877165
tau ( 73 ) = -0.02735224773167275
tau ( 82 ) = 0.0652266652354575
tau ( 91 ) = 0.00683610500011213

```

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

```

PS D:\Modelling> python .\zerolab.py
N= 10000 k = 5
Автокорреляция для алгоритма SFC64
tau ( 1 ) = 0.0049529598942424065
tau ( 10 ) = -0.0027871936509628447
tau ( 19 ) = -0.022997878689933752
tau ( 28 ) = -0.0017156197549112325
tau ( 37 ) = 0.009847880254916785
tau ( 46 ) = 0.013784193374356345
tau ( 55 ) = 0.0023707296429894393
tau ( 64 ) = -0.0012113643563077417
tau ( 73 ) = -0.01317767555931359
tau ( 82 ) = -0.00017507999151938197
tau ( 91 ) = 0.008593249481547936
Автокорреляция для алгоритма Вихрь Мерсенна
tau ( 1 ) = -0.020105234071974627
tau ( 10 ) = -0.0057865207486307
tau ( 19 ) = 0.010474211652384234
tau ( 28 ) = -0.0005636066405036469
tau ( 37 ) = -0.015693008328482198
tau ( 46 ) = 0.003248708505306423
tau ( 55 ) = -0.008928172189348195
tau ( 64 ) = -0.0028087087810446856
tau ( 73 ) = 0.003188226749243323
tau ( 82 ) = -0.000176443696103198
tau ( 91 ) = -0.013540827290883644
Автокорреляция для алгоритма PCG64
tau ( 1 ) = -0.0036371156574175473
tau ( 10 ) = 0.007184662724293642
tau ( 19 ) = -0.0029246469972962624
tau ( 28 ) = -0.0005102131143244061
tau ( 37 ) = -0.01300537755661771
tau ( 46 ) = -0.0019049406248552697
tau ( 55 ) = -0.010728248035605632
tau ( 64 ) = -0.018913611981516934
tau ( 73 ) = 0.011946494470089592
tau ( 82 ) = -0.009269757042126679
tau ( 91 ) = -0.010211932584089577

```

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

```

PS D:\Modelling> python .\zerolab.py
N= 100000 k = 5
Автокорреляция для алгоритма SFC64
tau ( 1 ) = 0.001288921608856184
tau ( 10 ) = 0.002305651015115068
tau ( 19 ) = 0.008803712894536998
tau ( 28 ) = 0.003571813243704613
tau ( 37 ) = -0.00017526260492533688
tau ( 46 ) = 0.005094872311554508
tau ( 55 ) = -0.003795603421001468
tau ( 64 ) = 0.002613967549240699
tau ( 73 ) = -0.005420883022986183
tau ( 82 ) = -0.002315695817045887
tau ( 91 ) = -0.004896063236415736
Автокорреляция для алгоритма Вихрь Мерсенна
tau ( 1 ) = -0.0019332818022458553
tau ( 10 ) = -0.0020596696977760415
tau ( 19 ) = -0.00204251999446973
tau ( 28 ) = 0.002242998296916765
tau ( 37 ) = 0.002877551911014661
tau ( 46 ) = 0.003775228891134887
tau ( 55 ) = 0.0014405357196213397
tau ( 64 ) = 0.0022516639755064136
tau ( 73 ) = 0.001504652228888584
tau ( 82 ) = 0.00029716007402895455
tau ( 91 ) = -0.0039096930344490275
Автокорреляция для алгоритма PCG64
tau ( 1 ) = 0.0030216491449390507
tau ( 10 ) = -0.0010516428272222631
tau ( 19 ) = 0.0006959419161299621
tau ( 28 ) = 0.0006428766067995191
tau ( 37 ) = -0.0009879926220868875
tau ( 46 ) = -0.0012319710450029713
tau ( 55 ) = 0.0027244048491334577
tau ( 64 ) = 0.005917546785602808
tau ( 73 ) = -0.0041071414162837635
tau ( 82 ) = -0.0011290251614886652
tau ( 91 ) = -0.0020486392049809546

```

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 $\alpha=0.05$. 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 случайных 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 случайных 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 случайных 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 случайных чисел с помощью Numpy random")
    # print(series)

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

    # print("\nГенерируем %N случайных 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 случайных 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>