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

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №0

по дисциплине «Моделирование»

Выполнил:
Студент гр. ИВ-622
Евтушенко Н. А.

Проверила:
Ассистент Кафедры ВС
Петухова Я.В.

Новосибирск 2020

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Взять готовую реализацию трёх генераторов псевдослучайных чисел и убедиться в их равномерном распределении, используя такие параметры как «хи-квадрат» и автокорреляция.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Пусть X - исследуемая случайная величина. Требуется проверить гипотезу H_0 о том, что данная случайная величина подчиняется закону распределения *F(x)*. Для этого необходимо произвести выборку из n независимых наблюдений над случайной величиной X: $X^n = (x_1, \dots , x_n), x_i \in [a, b], \forall_i = 1 \dots n$, и по ней построить эмпирический закон распределения *F'(x)* случайной величины X.

Гипотеза H_0': X^n порождается ранее упомянутой функцией *F'(x)*. Разделим [a,b] на k непересекающихся интервалов $[a_i, b_i], i = 1 \dots k$.

Пусть n_j — количество наблюдений в j-м интервале;

$p_j = F\left(b_j\right) - F(a_j)$ - вероятность попадания наблюдения в j-й интервал при выполнении гипотезы H_0';

$E_j = np_j$ - ожидаемое число попаданий в j-й интервал; тогда распределение Хи-квадрат с числом степеней свободы k−1 будет иметь следующую статистику:

$$\chi^2 = \sum_{j=1}^{k} \frac{(n_j - E_j)^2}{E_j} \sim \chi_{k-1}^2$$

В зависимости от значения критерия χ^2 гипотеза H_0 может приниматься либо отвергаться:

- $\chi_1^2 < \chi^2 < \chi_2^2$ - гипотеза H_0 выполняется.
- $\chi^2 \le \chi_1^2$ — попадает в левый «хвост» распределения. Следовательно, теоретические и практические значения очень близки. Если, к примеру, происходит проверка генератора случайных чисел, который сгенерировал *n* чисел из отрезка [0,1] и выборка X^n распределена равномерно на [0,1], то генератор нельзя называть случайным (гипотеза случайности не выполняется), т.к. выборка распределена слишком равномерно, но гипотеза H_0 выполняется.

- $\chi^2 \geq \chi_2^2$ — попадает в правый «хвост» распределения, гипотеза H_0 отвергается.

Автокорреляционная функция – предназначена для оценки корреляции между смещенными копиями последовательностей.

$$a(\tau) = \frac{\sum_{i=1}^{N-\tau}(x_i - Ex)(x_{i+\tau} - Ex)}{(N-\tau) * S^2(x)}, \text{где}$$

Ex - математическое ожидание – среднее значение случайной величины при стремлении количества выборок или количества её измерений (иногда говорят – количества испытаний) к бесконечности:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$S^2(x)$ - выборочная дисперсия случайной величины – это оценка теоретической дисперсии распределения, рассчитанная на основе данных выборки:

$$S^2(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \acute{x})^2 = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - (\acute{x})^2;$$

$x_{i+\tau}$ – множество значений другой случайной величины (полученной из значений прошлой случайной величины, но с некоторым смещением);

n – мощность множества случайных величин;

τ – смещение последовательности.

ГЕНЕРАТОРЫ СЛУЧАЙНЫХ ЧИСЕЛ

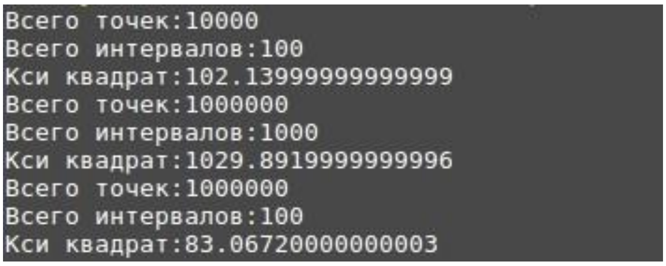
В качестве генератора случайных чисел были взяты:

- Стандартный генератор для языка Python
- Второй генератор случайных чисел альтернативный стандартному – SystemRandom. В модуле random есть класс SystemRandom, внутри реализации которого идет обращение как раз к os.urandom. В этом альтернативном генераторе реализованы те же функции. Она выдает так же псевдослучайные данные, но они зависят дополнительно и от операционной системы. Результаты используются в криптографии. Есть недостаток – то что функции SystemRandom отрабатывают в несколько раз дольше.
- Встроенный в NumPy генератор псевдослучайных чисел в подмодуле random. Числа являются псевдослучайными, в том плане что, они сгенерированы детерминистически из порождающего элемента (seed number), но рассредоточены в статистическом сходстве с случайным образом. Для генерации NumPy использует особенный алгоритм, который имеет название Mersenne Twister – это генератор случайных чисел на основе алгоритма «Вихря Марсенна». Вихрь Мерсенна — генератор псевдослучайных чисел основывается на свойствах простых чисел Мерсенна (отсюда название) и обеспечивает быструю генерацию высококачественных по критерию случайности псевдослучайных чисел. Вихрь Мерсенна лишён многих недостатков, присущих другим ГПСЧ, таких как малый период, предсказуемость, легко выявляемые статистические закономерности.

РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

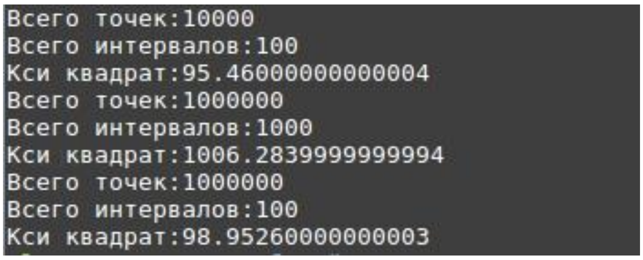
В ходе лабораторной работы были использованы все три ГСЧ. При помощи них были сгенерированы случайные значения в диапазоне [0, 1]. Данный диапазон был поделен на 100 и 1 000 равномерных отрезков. Далее узнаём сколько было попаданий на отрезки.

На скриншотах ниже показан результаты работы разных генераторов случайных чисел при изменении количества чисел и интервалов.



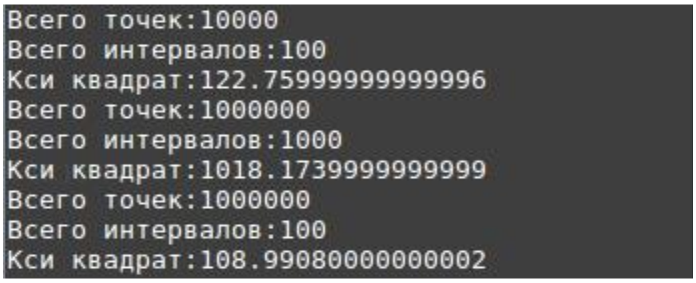
```
Всего точек:10000
Всего интервалов:100
Кси квадрат:102.13999999999999
Всего точек:1000000
Всего интервалов:1000
Кси квадрат:1029.8919999999996
Всего точек:1000000
Всего интервалов:100
Кси квадрат:83.06720000000003
```

Рисунок 1– Результат расчёта «хи-квадрат» для ГСЧ: Стандартный random



```
Всего точек:10000
Всего интервалов:100
Кси квадрат:95.46000000000004
Всего точек:1000000
Всего интервалов:1000
Кси квадрат:1006.2839999999994
Всего точек:1000000
Всего интервалов:100
Кси квадрат:98.95260000000003
```

Рисунок 2 – Результаты расчёта «хи-квадрат» для ГСЧ: SystemRandom



```
Всего точек:10000
Всего интервалов:100
Кси квадрат:122.75999999999996
Всего точек:1000000
Всего интервалов:1000
Кси квадрат:1018.1739999999999
Всего точек:1000000
Всего интервалов:100
Кси квадрат:108.99080000000002
```

Рисунок 3 – Результаты расчёта «хи-квадрат» для ГСЧ: NumPY

Была рассчитана автокорреляция для каждого ГСЧ при N = 1.000.000 и k = 1.000, где $a(\tau)$ – автокорреляция, τ – смещение интервалов. Исходя из ниже представленных данных (рисунки 4, 5, 6) можно сделать вывод, что числа действительно являются случайными, так как значения автокорреляции стремятся к нулю.

```
t: 1
a(t): -0.00015510106307124803
t: 2
a(t): -9.502794609951921e-05
t: 3
a(t): -3.293030287239702e-05
t: 4
a(t): 5.6717856130100635e-06
t: 5
a(t): 3.70662890042748e-05
t: 6
a(t): 3.1229065847522166e-05
t: 7
a(t): 1.1552471783893596e-05
t: 8
a(t): -6.694497435108717e-05
t: 9
a(t): -2.407503600131013e-05
t: 10
a(t): 3.3139513743229316e-05
t: 11
a(t): 3.134106124032582e-05
t: 12
a(t): -3.4915659166686736e-05
t: 13
a(t): 4.4351040541599696e-05
t: 14
a(t): -5.126711908074029e-05
t: 15
a(t): -0.00014914539104447569
t: 16
a(t): -8.737268881258275e-05
t: 17
a(t): -0.00015208758477398284
t: 18
a(t): 7.507619071155462e-05
t: 19
a(t): -2.971716913247061e-05
t: 20
a(t): 4.1653585051034456e-05
```

Рисунок 4 - Коэффициент корреляции генератором NumPY

```
t: 1
a(t): -8.906397061711737e-06
t: 2
a(t): 3.3880150198798624e-05
t: 3
a(t): 0.00012524557321904445
t: 4
a(t): -7.078808210108819e-05
t: 5
a(t): -0.00017826665708479153
t: 6
a(t): 7.279629682775103e-05
t: 7
a(t): -0.00010300330901444865
t: 8
a(t): 0.00011752774911759268
t: 9
a(t): -2.028110087604914e-05
t: 10
a(t): 1.4892584828803171e-06
t: 11
a(t): -7.598183139808489e-05
t: 12
a(t): -8.375638179571828e-05
t: 13
a(t): 0.00015353724919966812
t: 14
a(t): 0.00011720040638821478
t: 15
a(t): -1.0728820064648161e-05
t: 16
a(t): -0.00010635763771899102
t: 17
a(t): -1.1042324988726052e-05
t: 18
a(t): -6.879178768772822e-06
t: 19
a(t): -5.294894101875272e-05
t: 20
a(t): 2.7980420650833437e-05
```

Рисунок 5 - Коэффициент корреляции генератором SystemRandom

```
t: 1
a(t): 7.623779722957886e-05
t: 2
a(t): 8.635717053142516e-05
t: 3
a(t): 3.036687340900454e-05
t: 4
a(t): 3.563566891960648e-05
t: 5
a(t): -6.701104736598013e-05
t: 6
a(t): 5.047517400933642e-06
t: 7
a(t): -8.710437118159039e-05
t: 8
a(t): 9.839616589100559e-05
t: 9
a(t): -7.622048919319192e-06
t: 10
a(t): -3.933716429982644e-05
t: 11
a(t): -8.101133226410921e-05
t: 12
a(t): -0.00010861760702480494
t: 13
a(t): -0.00012282543943977042
t: 14
a(t): 7.3064918188079le-06
t: 15
a(t): 8.853925571094057e-05
t: 16
a(t): 5.108167443304017e-05
t: 17
a(t): 0.00010987666633659189
t: 18
a(t): 6.909283970276065e-05
t: 19
a(t): 0.00012797761570141106
t: 20
a(t): -6.344652899330486e-05
```

Рисунок 6 - Коэффициент корреляции генератором Стандартный random

Заключение

Были проведены эксперименты с генераторами случайных чисел, С помощью полученных данных провели анализ равномерности распределения выбранных нами ГСЧ по критерию Пирсона и автокорреляции.

Основываясь на результаты экспериментов, мы узнали, что критерии «хи-квадрат» примерно равны $\chi^2_{эксп,\ random} = 991.89$, $\chi^2_{эксп,\ SystemRa} = 1015.063$, $\chi^2_{эксп,\ NumPy} = 1023.724$, также можем заметить, что «хи-квадрат» табличного равен $\chi^2_{таб} = 1142.848$. Следовательно, $\chi^2_{эксп} < \chi^2_{таб}$ указывает на то, что гипотезы о равновероятном распределении в генераторах случайных чисел принимается. Если бы значение $\chi^2_{эксп}$ попало бы в критическую область, то гипотеза бы не подтвердилась. Для всех трех различных генераторов, которые мы взяли, гипотеза принимается.

Результаты расчёта критерия Пирсона говорят о том, что чем больше отклонение, тем меньше распределение. Значения этого критерия в ходе эксперимента не выходили за рамки критических, это говорит о том, что гипотезу о равномерном распределении нельзя отвергнуть.

Во всех экспериментах автокорреляционная функция после изменения τ (смещение в последовательности) приближена к нулю, что говорит о слабой корреляции. Это показывает, что зависимость от данных крайне мала.

ПРИЛОЖЕНИЕ

```
import random
import numpy as np
from math import sqrt
import typing as tp

def get_float_rand() -> float:
    return float(random.random())


def get_SystemRandom() -> float:
    r = random.SystemRandom()
    rand = r.random()
    return float(rand)


def get_NumPy_rand() -> float:
    rnd = np.random.sample()
    return float(rnd)


def mat_waiting(N: int, x: tp.Sequence[int]) -> float:
    i = 0
    Ex = 0
    while i < N:
        Ex += x[i] * (1 / N)
        i += 1

    return float(Ex)


def sample_variance(N: int, x: tp.Sequence[int]) -> float:
    i = 0
    Sx = 0
    Ex = mat_waiting(N, x)
    while i < N:
        Sx += (x[i] + Ex) ** 2
        i += 1

    Sx = Sx / N

    return float(Sx)


def autocorrelation(offset: int, N: int, x: tp.Sequence[int], Ex: float, S2x: float) -> float:
    R = 0
    i = 0
    while i < N - offset:
        R += (x[i] - Ex) * (x[i+offset] - Ex)
        i += 1
```

```
    R = R / ((N - offset) * S2x)

    return float(R)

N = 1000000
K = 1000
y = range(N)
x = []
array = [0] * K
ksi = 0
offset = 1
for k in y:
    rand = get_float_rand()
    x.append(rand)
    inter = rand / (1 / K)
    array[int(inter)] += 1

print('Всего точек:' + str(N))
print('Всего интервалов:' + str(K))

avg = 0
for n in x:
    avg += n
avg = avg / (N)
print('Сред. арифм. :'  + str(avg))

i = 0
summary = 0
while i < len(x):
    summary += (x[i] - avg) ** 2
    i += 1
deviation = sqrt((1 / N) * summary)
print('Сред. квадр. откл. :'  + str(deviation))

p_i = 1 / K

i = 0
while i < K:
    ksi += (((array[i] - p_i * N) ** 2) / (p_i * N))
    i += 1
print('Кси квадрат:' + str(ksi))

Ex = mat_waiting(N, x)
S2x = sample_variance(N, x)
i = 0
offset = 1
while i < K / 2:
    R = autocorrelation(offset, N, x, Ex, S2x)
    print('offset: ' + str(offset))
    print('R: ' +  str(R))
```



```
    i += 1
    offset +=1
```