Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 3

По курсу: Моделирование

На тему: Генераторы псевдослучайных чисел

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-76Б

Преподователь:

Рудаков Игорь Владимирович

Москва, 2021 г.

Содержание

1 Задание

Изучить и реализовать генератор псевдослучайных чисел программным и табличным методом. Разрядность чисел должна быть равна 1, 2, 3. Сравнить методы по определенному критерию и сделать выводы.

2 Теоритическая часть

В данной лабораторной работе рассматриваются 2 метода генерации случайных чисел:

1. программный;
2. табличный.

2.1 Программный генератор

Программный генератор формирует псевдослучайные числа. Каждое последующее число в такой последовательности зависит от предыдущего.

2.2 Табличный генератор

Табличный генератор использует таблицу проверенных некоррелированных цифр в качестве источника случайных чисел.

2.3 Критерий случайности

Для сравнения описанных методов генерации случайных чисел воспользуемся **критерием частотности**, который позволяет определить равномерность сгенерированных чисел.

Определяется количество чисел на интервале $(\mu - \sigma,\ \mu + \sigma)$, где μ — математическое ожидание, а σ — среднеквадратичное отклонение. Идеальным результатом будем считать отношение длины рассматриваемого интервала к длине всего промежутка, на котором генерируется последовательность. Под полученным результатом будем понимать отношение количества сгенерированных чисел на интервале к количеству всех сгенерированных чисел.

3 Результаты

```
$ python run.py
Input n: 10
Do you want input yourself (y/n) ?n
Программный метод:
 No          |   1 разр. |   2 разр. |   3 разр.
-------------+-----------+-----------+-----------
 1           |    7.0000 |   67.0000 |  448.0000
 2           |    1.0000 |   97.0000 |  119.0000
 3           |    7.0000 |   78.0000 |  478.0000
 4           |    2.0000 |   13.0000 |  524.0000
 5           |    8.0000 |   97.0000 |  992.0000
 6           |    2.0000 |   74.0000 |  387.0000
 7           |    3.0000 |   22.0000 |  152.0000
 8           |    2.0000 |   58.0000 |  242.0000
 9           |    2.0000 |   16.0000 |  828.0000
 10          |    4.0000 |   55.0000 |  901.0000
-------------+-----------+-----------+-----------
 Ожидаемый   |    0.4883 |    0.6619 |    0.6514
 Полученный  |    0.6000 |    0.5000 |    0.5000

Табличный метод:
 No          |   1 разр. |   2 разр. |   3 разр.
-------------+-----------+-----------+-----------
 1           |    5.0000 |   55.0000 |  723.0000
 2           |    2.0000 |   70.0000 |  294.0000
 3           |    9.0000 |   67.0000 |  921.0000
 4           |    2.0000 |   69.0000 |  925.0000
 5           |    2.0000 |   33.0000 |  707.0000
 6           |    0.0000 |   66.0000 |  903.0000
 7           |    0.0000 |   67.0000 |  179.0000
 8           |    7.0000 |   85.0000 |  122.0000
 9           |    2.0000 |   33.0000 |  860.0000
 10          |    6.0000 |   94.0000 |  961.0000
-------------+-----------+-----------+-----------
 Ожидаемый   |    0.5814 |    0.4114 |    0.6988
 Полученный  |    0.6000 |    0.6000 |    0.7000
```

Рис. 1: Пример работы для 10 чисел

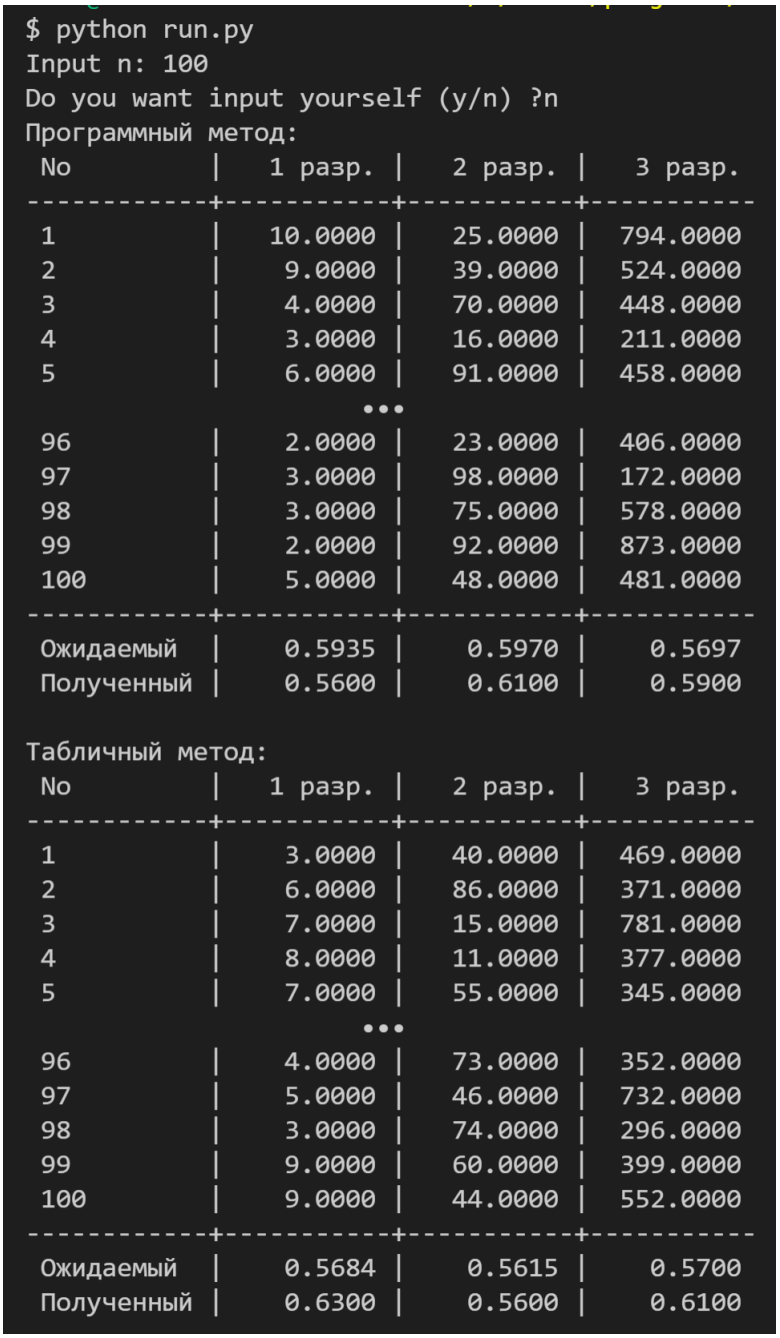

Рис. 2: Пример работы для 100 чисел

```
$ python run.py
Input n: 1000
Do you want input yourself (y/n) ?n
Программный метод:
 No          |   1 разр. |   2 разр. |   3 разр.
-------------+-----------+-----------+-----------
 1           |    6.0000 |   97.0000 |  690.0000
 2           |    9.0000 |   69.0000 |  809.0000
 3           |    3.0000 |   82.0000 |  250.0000
 4           |    3.0000 |   89.0000 |  142.0000
 5           |    4.0000 |   99.0000 |  678.0000
                    •••
 996         |    3.0000 |   91.0000 |  867.0000
 997         |    1.0000 |   68.0000 |  981.0000
 998         |    2.0000 |   62.0000 |  907.0000
 999         |    5.0000 |   48.0000 |  122.0000
 1000        |    8.0000 |   97.0000 |  150.0000
-------------+-----------+-----------+-----------
 Ожидаемый   |    0.5690 |    0.5953 |    0.5781
 Полученный  |    0.6190 |    0.5760 |    0.5660

Табличный метод:
 No          |   1 разр. |   2 разр. |   3 разр.
-------------+-----------+-----------+-----------
 1           |    1.0000 |   11.0000 |  854.0000
 2           |    5.0000 |   12.0000 |  907.0000
 3           |    0.0000 |   20.0000 |  291.0000
 4           |    9.0000 |   28.0000 |  960.0000
 5           |    6.0000 |   11.0000 |  770.0000
                    •••
 996         |    8.0000 |   86.0000 |  421.0000
 997         |    2.0000 |   14.0000 |  436.0000
 998         |    3.0000 |   25.0000 |  322.0000
 999         |    2.0000 |   42.0000 |  677.0000
 1000        |    7.0000 |   18.0000 |  175.0000
-------------+-----------+-----------+-----------
 Ожидаемый   |    0.5742 |    0.5870 |    0.5615
 Полученный  |    0.5960 |    0.5670 |    0.5680
```

Рис. 3: Пример работы для 1000 чисел

4 Листинг кода

```
def frequency_criterion(sequence, num_len):
    mean = numpy.mean(sequence)
    stdd = numpy.sqrt(numpy.var(sequence))
    cnt = 0
    for item in sequence:
        if (mean - stdd) < item < (mean + stdd):
        cnt += 1

    sequence_max_delta = 10
    if num_len > 1:
        sequence_max_delta = 9 * 10**(num_len-1)

    return (2 * stdd / sequence_max_delta), (cnt /  len(sequence))


class StandardRandom():
    def get(self, num_len: int):
        return random.randint(10**(num_len - 1), 10**num_len)


class TableRandom():
    def __init__(self, path_to_table: str = "./table.txt"):
        self.digits = ""

        with open(path_to_table) as table_file:
            for line in table_file:
                self.digits += line[:-1]

        self.idx_init()

        def idx_step(self, step: int = 1):
            self.idx += step

        if self.idx + step >= len(self.digits):
            self.idx_init(step)

        def idx_init(self, offset: int = 0):
```

```
38              now = datetime.datetime.now()
39              self.idx = int((now.microsecond % 60)/60 * \\
40                             (len(self.digits) - offset))
41
42          def get(self, num_len: int):
43              self.idx_step(num_len)
44              rnd_num = int(self.digits[self.idx:self.idx+num_len])
45
46          while len(str(rnd_num)) < num_len:
47              self.idx_step()
48              rnd_num = rnd_num * 10 + int(self.digits[self.idx])
49
50          return rnd_num
```

Листинг 1: Программная реализация генерации псевдослучайных чисел программным и табличным методом

5 Вывод

Из результатов проведённой лабораторной работы справедливо сделать вывод, что, чем больше количество генерируемых случайных чисел, тем равномернее они распределены.