МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и прикладной математики
Кафедра прикладной математики и экономико-математических методов

ОТЧЁТ

по дисциплине:

«Имитационное моделирование»

на тему:

«Моделирование случайных величин. Статистические испытания. Задание №2»

Направление: 01.03.02
Обучающийся: Бронников Егор Игоревич
Группа: ПМ-1901

Санкт-Петербург
2022

Задание №1

Реализовать генератор случайных чисел, используя метод серединных квадратов (фон Нейман). Проанализировать свойства полученной последовательности.

В методе серединных квадратов изначально задаётся количество разрядов числа (k) и начальное значение R_0. Далее число R_0 возводится в квадрат и из середины квадрата числа берётся k-значное число, которое снова возводится в квадрат, и так далее.
Обязательным условием является то, что количество разрядов (k) должно быть чётным числом.

Данный алгоритм был реализован на языке программирования Python. (Рисунок 1)

```python
def mid_square_method(init, digit, n = 10):
    if digit % 2 != 0: return

    r = init
    res = [r]
    mid_d = digit//2
    for i in range(n):
        r *= r
        digits = list(str(r))
        while len(digits) < digit:
            digits = ['0'] + digits
        r = int(''.join(digits[1:-1]))
        res.append(r)
    return res
```

Рис. 1: Реализация метода серединных квадратов на языке программирования Python

Алгоритм был запущен с параметрами $k = 10$, $R_0 = 25$, $n = 7$, где n – это количество сгенерированных случайных чисел. Можно проследить, что при достаточно малых разрядах и малом начальном значении быстро получается вырождение, что плохо. (Рисунок 2)

```
res = mid_square_method(25, 4, 7)
res
```

```
[25, 62, 84, 5, 2, 0, 0, 0]
```

```
plt.hist(res)
plt.show()
```

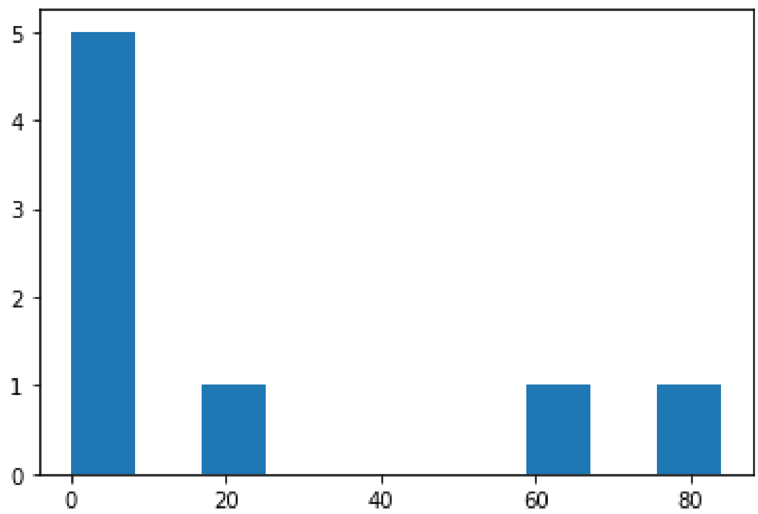

Рис. 2: Результаты генерации случайных чисел методом серединных квадратов

Задание №2

Реализовать линейный конгруэнтный датчик случайных чисел. Сгенерировать последовательность вещественных чисел, распределённых равномерно: 1) на интервале [0,1); 2) на интервале [a,b). Проанализировать полученные последовательности. Определить период, построить гистограмму.

Линейный конгруэнтный метод – это один из рекуррентных методов генерации случайных чисел. Следующий элемент последовательности может быть найден по следующей формуле:

$$r_{i+1} = (k \cdot r_i + b) \ mod \ M$$

Линейная конгруэнтная последовательность, определённая числами M, k, b, r_0 периодична с периодом, не превышающим M. При этом длина периода равна M тогда и только тогда, когда:

1. числа b и M взаимно простые;
2. $k-1$ кратно p для каждого простого p, являющегося делителем M;
3. $k-1$ кратно 4, если M кратно 4.

Сначала был реализован алгоритм для интервала [0;1) на языке программирования Python. (Рисунок 3)

```python
def linear_congruent_gauge_0_1(init, k, b, M, n):
    r = init
    unique = 0
    res = []
    for i in range(n):
        r = (k*r + b) % M
        if r/M not in res:
            unique += 1
        res.append(r/M)
    return res, unique
```

Рис. 3: Реализация линейного конгруэнтного счётчика на интервале [0,1)

Для того чтобы получить случайные числа в интервале от [0,1) нужно поделить каждый случайный сгенерированный элемент последовательности на M. Также данная функция выводит период сгенерированной последовательности.

```python
res = linear_congruent_gauge_0_1(3, 2, 1, 10, 12)
res
```

```
([0.7, 0.5, 0.1, 0.3, 0.7, 0.5, 0.1, 0.3, 0.7, 0.5, 0.1, 0.3], 4)
```

```python
plt.hist(res[0])
plt.show()
```

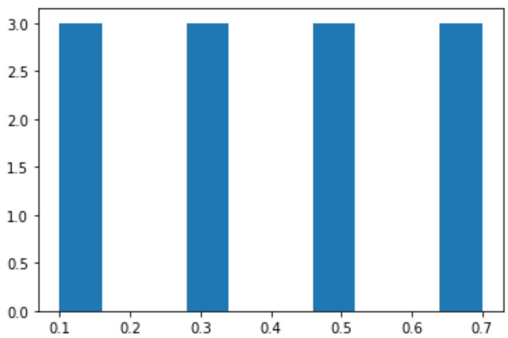

Рис. 4: Результаты генерации случайных чисел линейным конгруэнтным счётчиком на интервале [0,1)

В качестве аргументов были выбраны следующие значения: $r_0 = 3$, $k = 2$, $b = 1$, $M = 10$, $n = 12$, где n – это количество сгенерированных случайных чисел. Можно видеть, что при данном наборе аргументов длина периода составила 4. (Рисунок 4)

Далее был реализован алгоритм для интервала [a;b). (Рисунок 5)

```python
def linear_congruent_gauge_a_b(init, k, b, M, n, a_param, b_param):
    r = init
    unique = 0
    res = []
    for i in range(n):
        r = (k*r + b) % M
        val = (1-r/M)*a_param + (r/M)*b_param
        if val not in res:
            unique += 1
        res.append(val)
    return res, unique
```

Рис. 5: Реализация линейного конгруэнтного счётчика на интервале [a,b)

Для того чтобы получить случайные числа в интервале от [a,b) нужно проделать следующее преобразование:

$$(1 - \frac{r_i}{M})/a + \frac{r_i}{M} \cdot b$$

То есть сначала мы генерируем числа в интервале от [0,1), а дальше преобразуем их к интервалу от [a,b).

```python
res = linear_congruent_gauge_a_b(3, 2, 1, 10, 10, 100, 200)
res
```

```
([170.0, 150.0, 110.0, 130.0, 170.0, 150.0, 110.0, 130.0, 170.0, 150.0], 4)
```

```python
plt.hist(res[0])
plt.show()
```

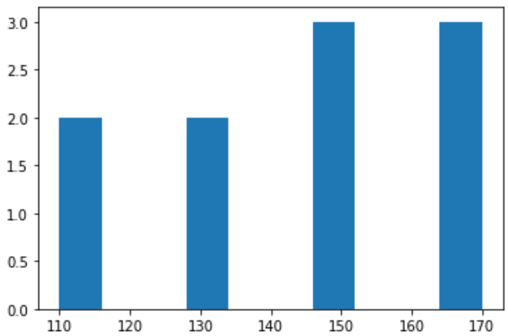

Рис. 6: Результаты генерации случайных чисел линейным конгруэнтным счётчиком на интервале [a,b)

В качестве аргументов были выбраны следующие значения: $r_0 = 3$, $k = 2$, $b = 1$, $M = 10$, $n = 10$, $a_{param} = 100$, $b_{param} = 200$, где n – это количество сгенерированных случайных чисел. Можно видеть, что при данном наборе аргументов длина периода составила 4. (Рисунок 6)

Задание №3

Используя метод обратной функции, получить последовательность случайных чисел, распределённых экспоненциально с заданным параметром λ. Проанализировать полученную последовательность. Оценить математическое ожидание и дисперсию, построить гистограмму.

Плотность распределения экспоненциального закона:

$$f(x) = \begin{cases} \lambda e^{-\lambda \cdot x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Функция распределения экспоненциального закона:

$$F(x) = \begin{cases} 1 - e^{-\lambda \cdot x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Получается, что обратная функция $F^{-1}(x)$ будет выглядеть следующим образом:

$$x = \frac{-ln(1-y)}{\lambda} = -\frac{ln(y)}{\lambda}$$

Если подставлять вместо y случайные равномерно распределённые значения, то можно получать требуемые числа.

Таким образом, была реализована функция на языке программирования Python. (Рисунок 7)

```
def exp_inverse_function_method(lambda_, n):
    return [-np.log(np.random.random())/lambda_ for _ in range(n)]
```

Рис. 7: Реализация метода обратной функции для экспоненциального закона

При $\lambda = 3$ и $n = 1000$ получается следующий результат. (Рисунок 8)

```
lambda_ = 3
res = exp_inverse_function_method(lambda_, 1000)
# res
```

```
plt.hist(res)
plt.show()
```

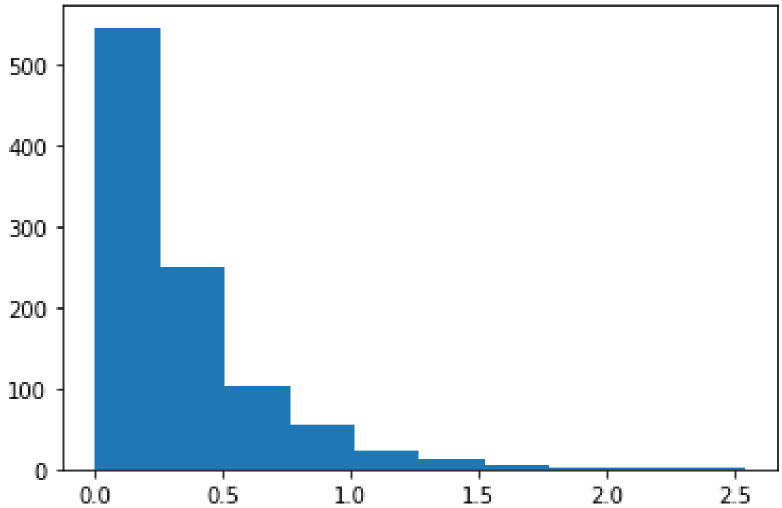

Рис. 8: Реализация метода обратной функции для экспоненциального закона

Математическое ожидание экспоненциального распределения:

$$E = \frac{1}{\lambda}$$

Если рассчитывать математическое ожидание как среднее значение в выборке, то получается следующий результат. (Рисунок 9)

Математическое ожидание:

```
m = sum(res)/len(res)
m
```

```
0.331958674573819
```

```
1/lambda_
```

```
0.3333333333333333
```

Рис. 9: Теоретическое и расчётное значения математического ожидания ($\lambda = 3$)

Можно заметить, что при $n = 1000$ значения получились достаточно близкими.
Дисперсия экспоненциального распределения:

$$D = \frac{1}{\lambda^2}$$

Далее можно рассчитать дисперсию как среднее квадратное отклонение от среднего значения выборки. (Рисунок 10)

Дисперсия:

```
d = sum((x-m)**2 for x in res) / len(res)
d
```

```
0.1084274985543402
```

```
1/lambda_**2
```

```
0.1111111111111111
```

Рис. 10: Теоретическое и расчётное значения дисперсии ($\lambda = 3$)

Можно заметить, что при $n = 1000$ значения получились достаточно близкими.

Задание №4