МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ

ЛАБОРАТОРНАЯ РАБОТА

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Стаина Романа Игоревича

Проверил
доцент _____ И. И. Слеповичев

Саратов 2023

СОДЕРЖАНИЕ

1 Постановка задачи

Цель.

1. Сгенерировать псевдослучайную последовательность заданным методом.
2. Исследовать полученную псевдослучайную последовательность на случайность.

Исходные данные. Исходными данными для лабораторных занятий являются метод генерации псевдослучайных чисел, диапазон генерации случайных чисел, функция распределения, которой должны подчиняться случайные числа, количество генерируемых чисел.

В данной работе были исследованы ППСЧ длиной 10000 элементов по модулю 1024, реализованные в практической работе №1. Их входные переменные, передаваемые через параметр `/i`:

1. Линейный конгруэнтный метод: `/i:31104,625,6571,23`.
2. Аддитивный метод: `/i:30000,24,55,79,134,213,347,560,907,1467,2374,3841,6215,10056,16271,26327,12598,8925,21523,448,21971,22419,14390,6809,21199,28008,19207,17215,6422,23637,59,23696,23755,17451,11206,28657,9863,8520,18383,26903,15286,12189,27475,9664,7139,16803,23942,10745,4687,15432,20119,5551,25670,1221,26891,28112,23779,17506`.
3. Пятипараметрический метод: `/i:89,7,13,24,10,234122131`.
4. РСЛОС: `/i:10011011010011010,17`.
5. Нелинейная комбинация РСЛОС: `/i:00000001010101,01011100000111101,010101001100000,97,1234,345231,10`.
6. Вихрь Мерсенна: `/i:1024,1234`.
7. RC4: `/i:213,968,838,64,355,214,212,36,695,139,897,518,656,956,810,510,985,105,670,8,907,951,685,989,222,931,169,286,289,556,731,902,688,701,771,533,990,630,708,884,255,683,25,214,792,348,34,758,9,781,946,580,615,955,585,5,886,563,81,38,809,444,619,222,544,53,635,621,630,251,497,257,2,467,897,790,728,676,722,838,465,781,10,828,903,235,857,841,146,719,681,678,961,652,491,38,256,909,251,21,110,811,273,25,642,286,489,478,184,812,770,846,241,141,266,500,375,827,633,761,154,663,461,206,529,212,667,342,360,165,523,749,582,803,553,345,786,990,361,702,256,380,234,238,73,965,266,300,847,755,969,681,146,843,125,306,845,752,879,458,788,833,727,817,122,239,765,877,827,327,733,658,644,880,150,474,493,689,670,368,611,263,113,417,834,103,725,754,117,824,623,338,540,337,879,521,183,370,808,120,571,871,301,210,796,744,398,106,845,745,842,876,399,27,105,601,802,831,53,266,157,352,175,303,505,484,994,425,292,729,654,584,860,`

`420,412,49,281,417,703,400,48,404,772,389,733,152,271,585,404,333,381,696,928,609,659,180`.

8. RSA: `/i:10967,571,77,10`.
9. Блюм-Блюма-Шуба: `/i:239,10`.

2 Статистические свойства последовательности псевдослучайных чисел

2.1 Вычисление оценок для ППСЧ

Относительные погрешности измерялись для выборки из 5000 элементов.

2.1.1 Линейный конгруэнтный метод

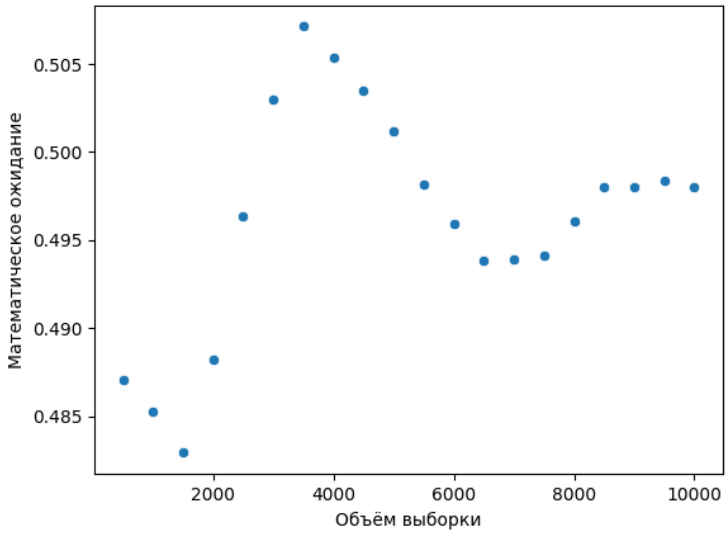

Рисунок 1 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 0.642%.

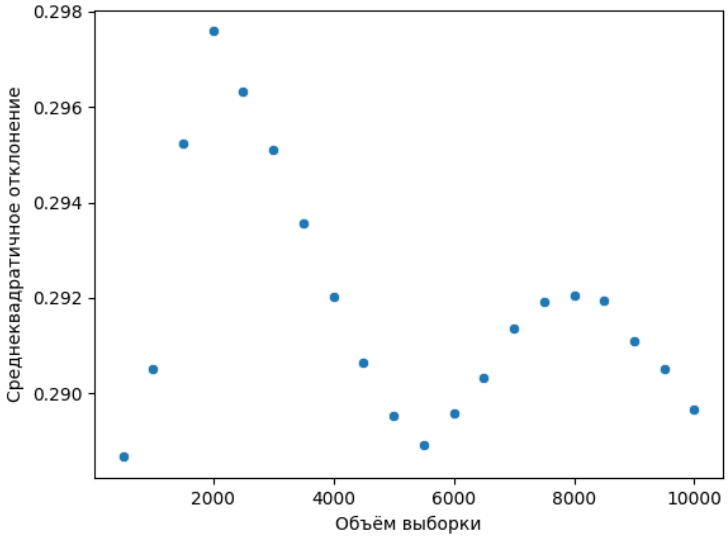

Рисунок 2 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 0.048%.

2.1.2 Аддитивный метод

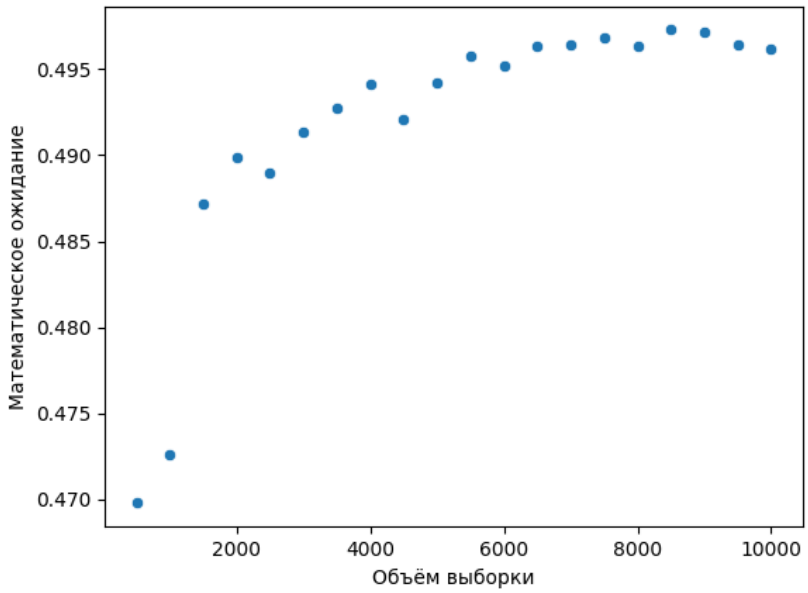

Рисунок 3 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 0.4%.

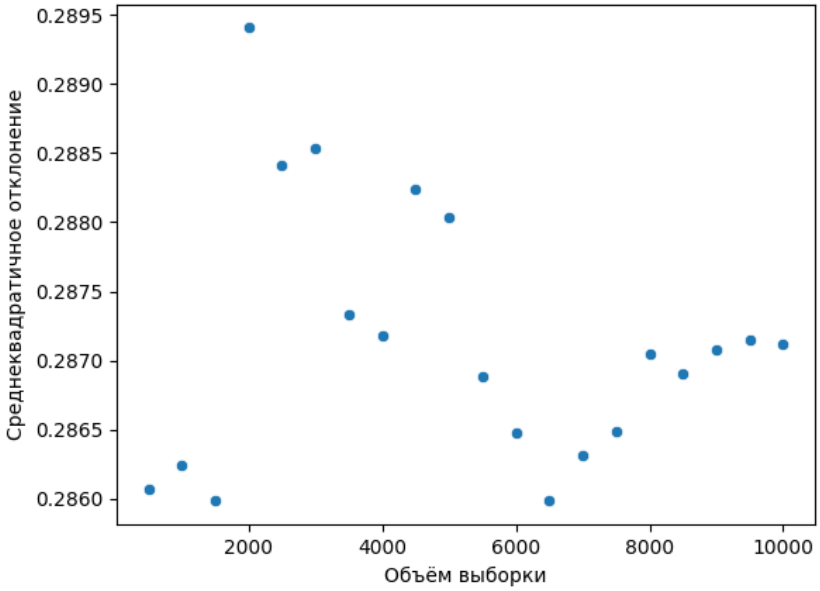

Рисунок 4 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 0.316%.

2.1.3 Пятипараметрический метод

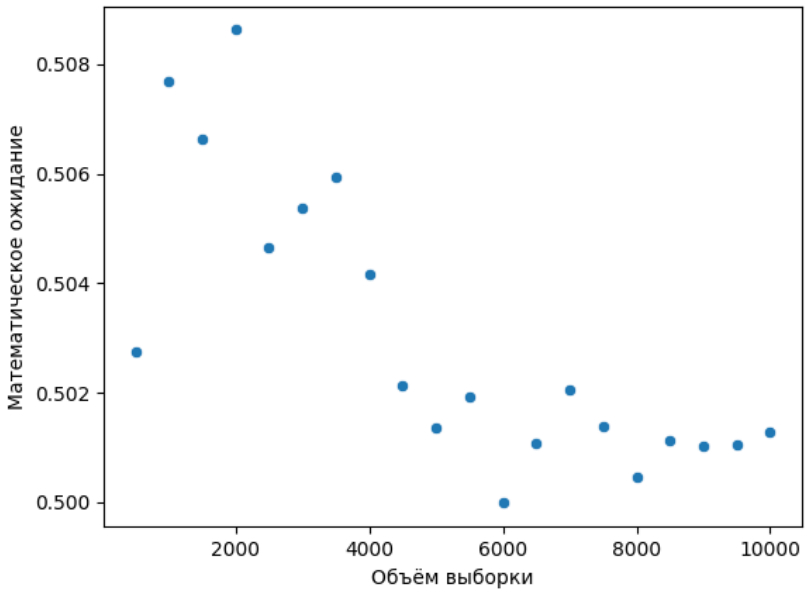

Рисунок 5 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 0.013%.

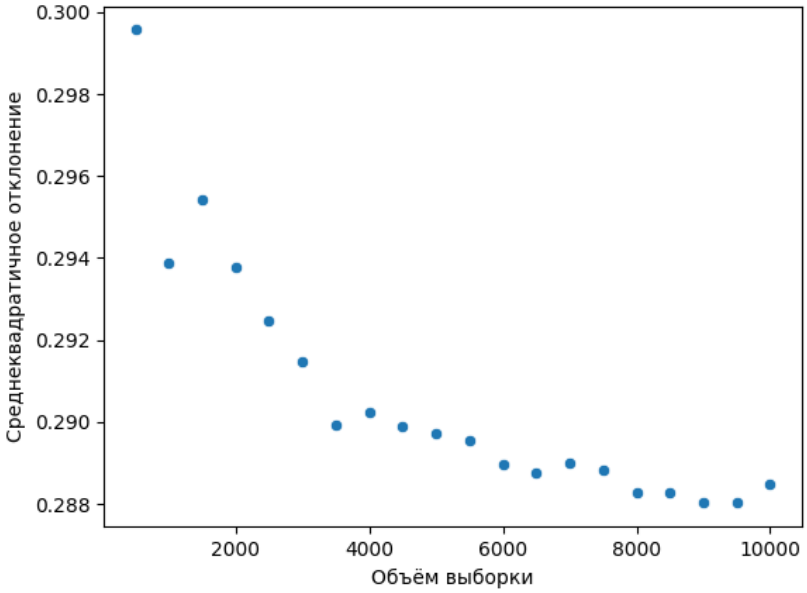

Рисунок 6 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 0.421%.

2.1.4 РСЛОС

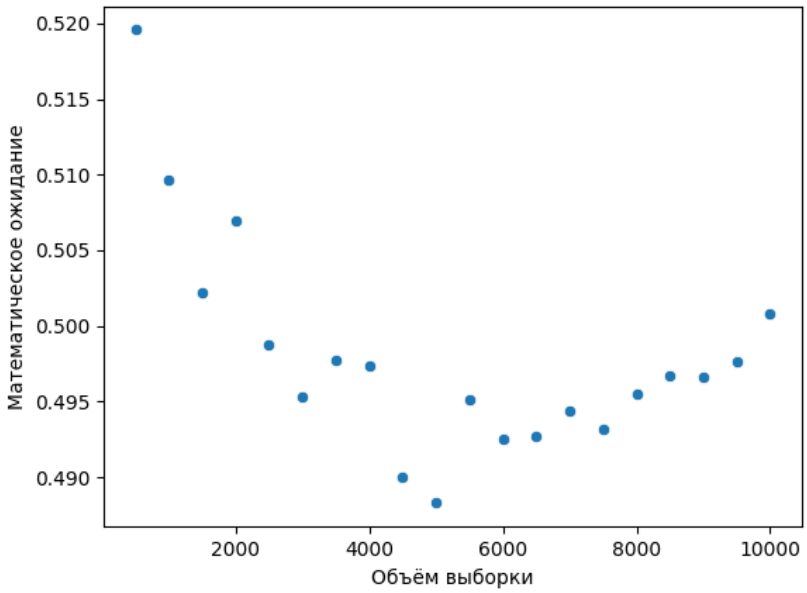

Рисунок 7 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 2.561%.

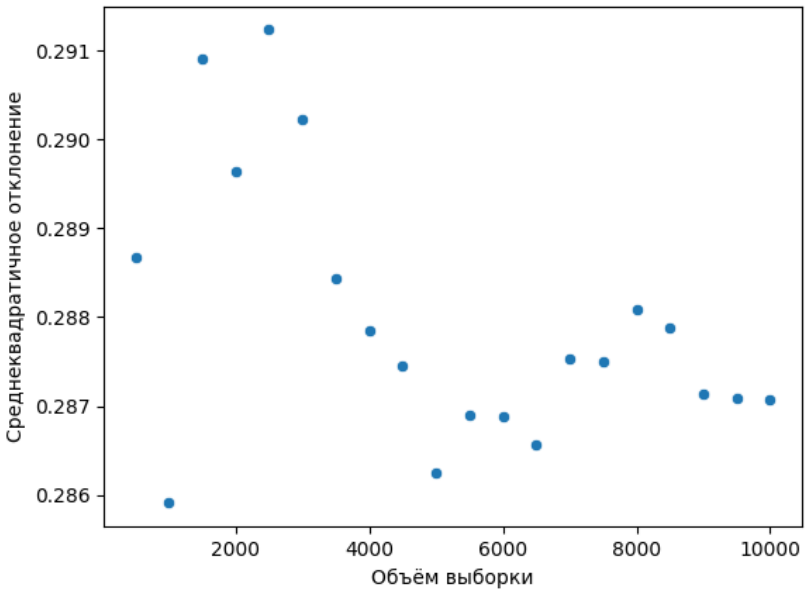

Рисунок 8 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 0.287%.

2.1.5 Нелинейная комбинация РСЛОС

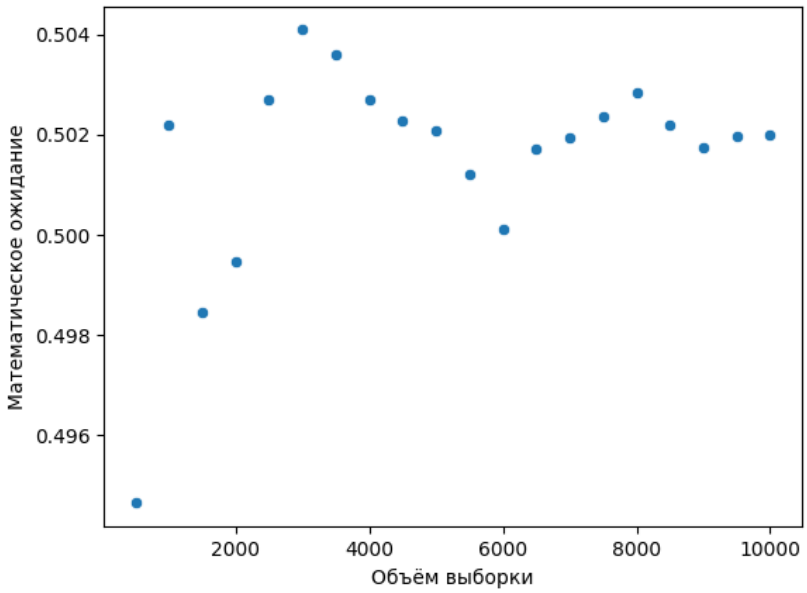

Рисунок 9 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 0.013%.

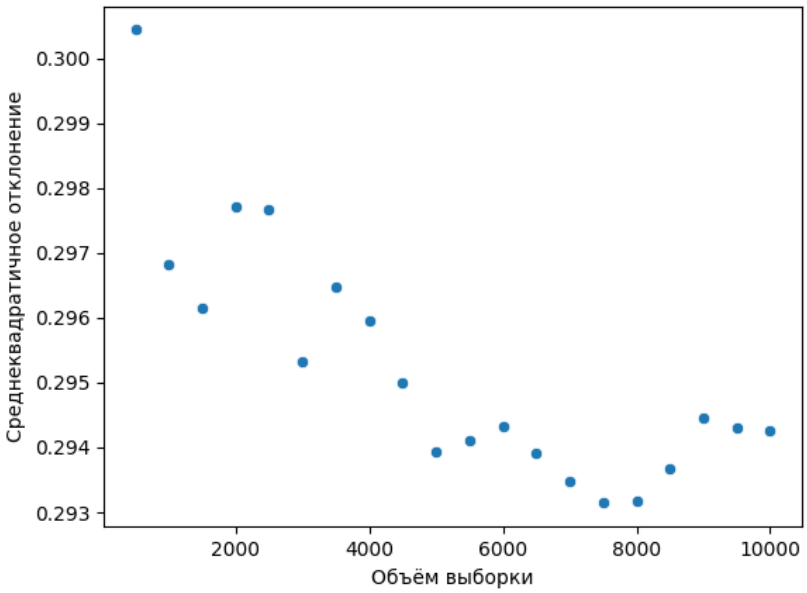

Рисунок 10 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 0.109%.

2.1.6 Вихрь Мерсенна

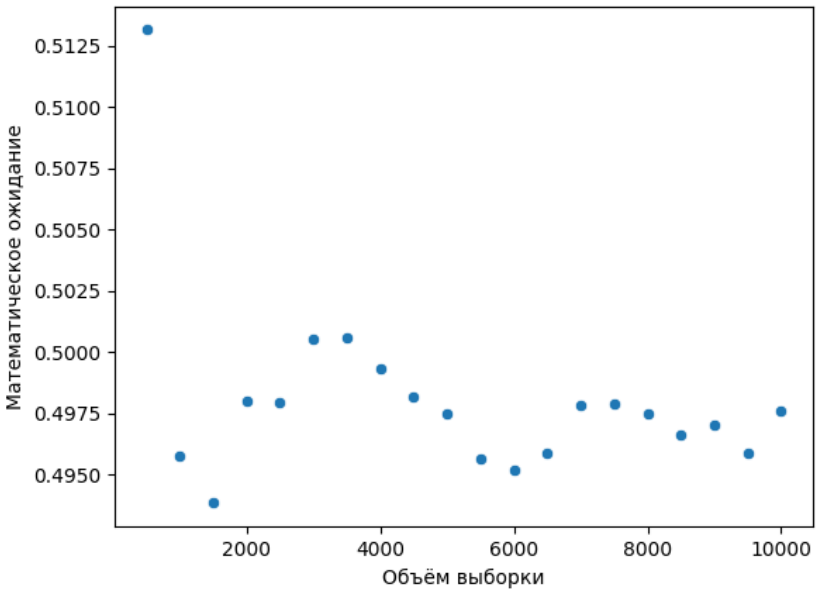

Рисунок 11 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 0.023%.

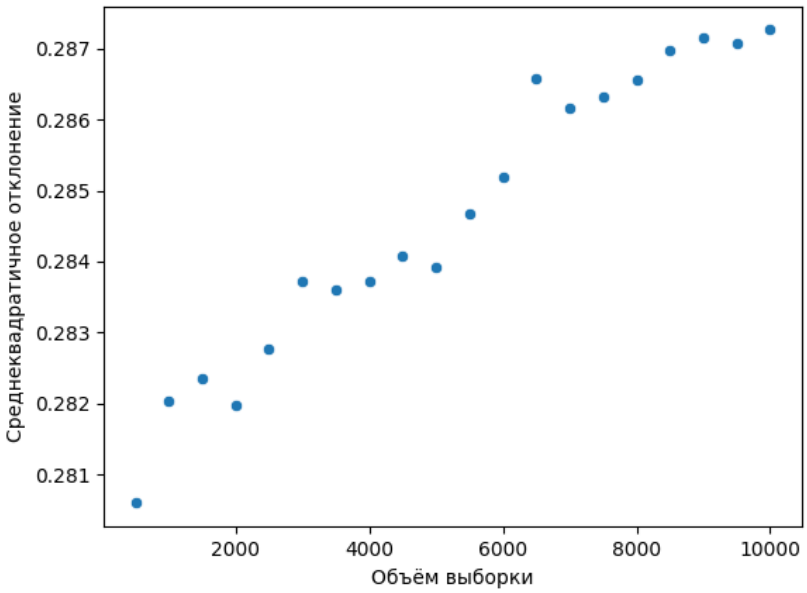

Рисунок 12 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 1.176%.

2.1.7 RC4

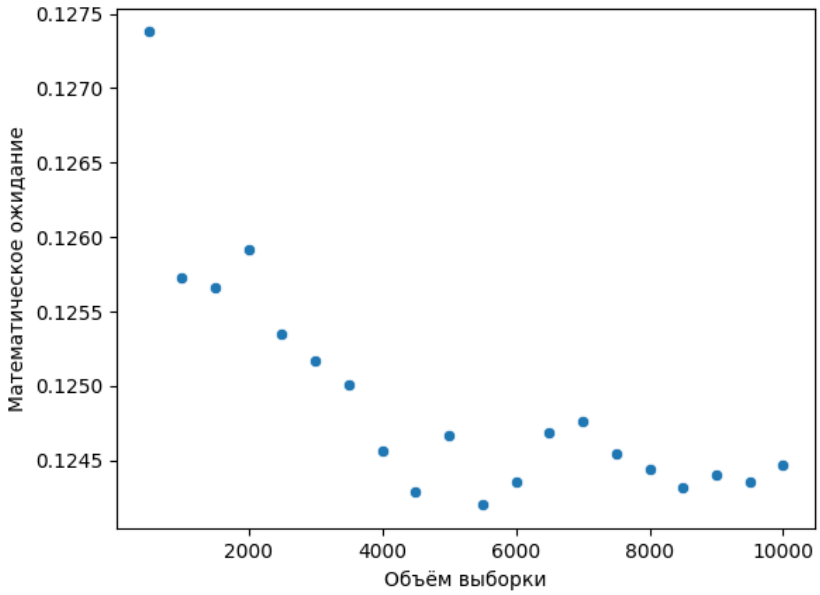

Рисунок 13 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 0.163%.

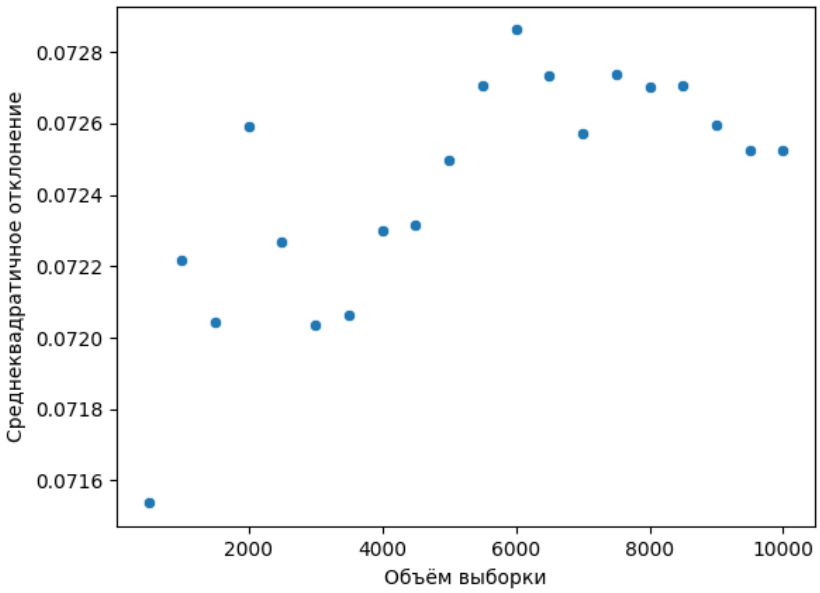

Рисунок 14 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 0.038%.

2.1.8 RSA

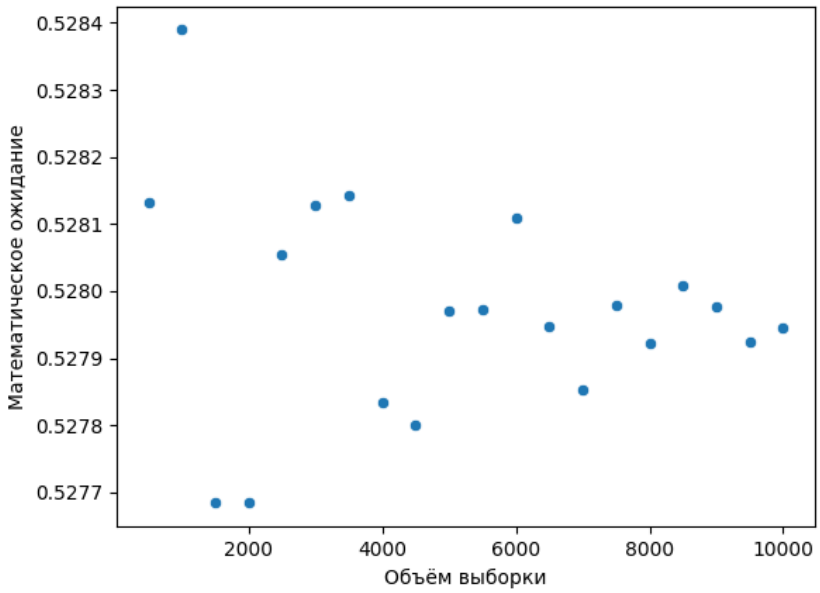

Рисунок 15 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 0.005%.

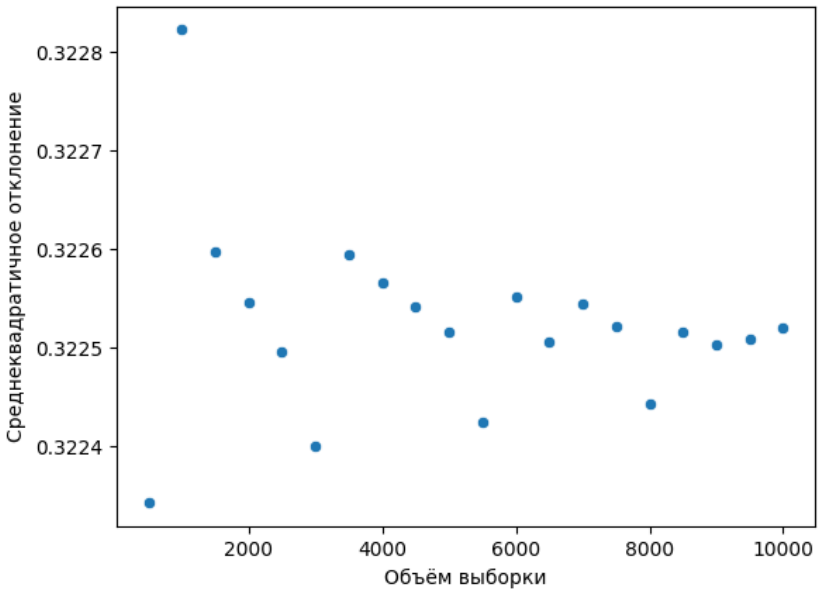

Рисунок 16 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 0.001%.

2.1.9 Блюм-Блюма-Шуба

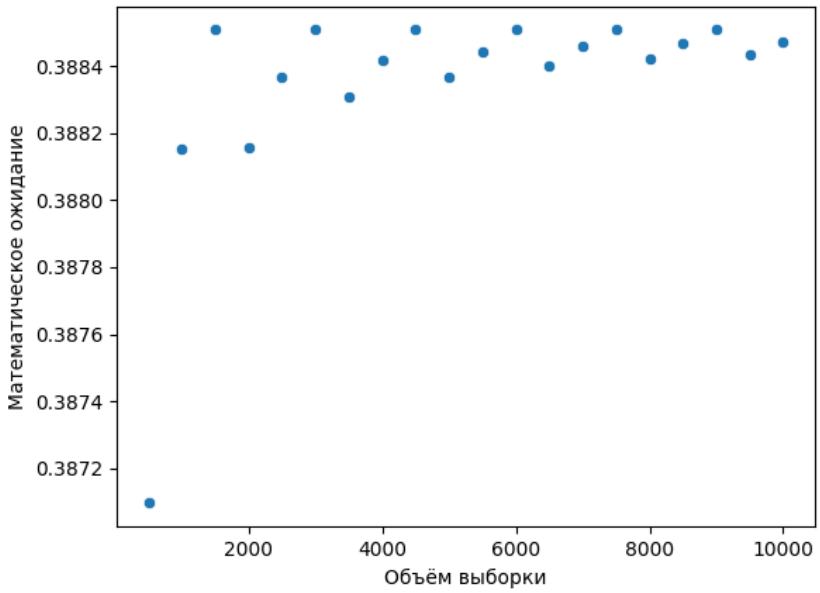

Рисунок 17 – Зависимость математического ожидания от объёма выборки

Относительная погрешность измерения: 0.027%.

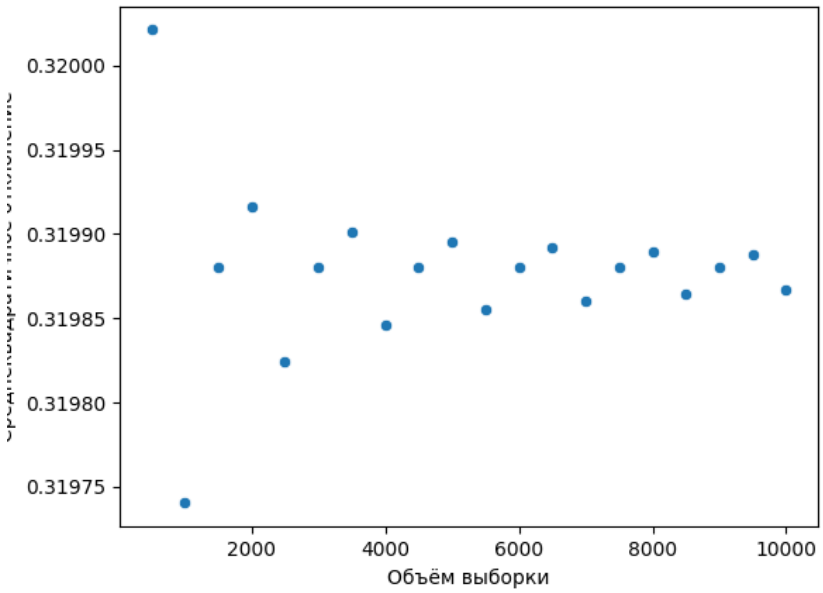

Рисунок 18 – Зависимость среднеквадратичного отклонения от объёма выборки

Относительная погрешность измерения: 0.009%.

2.2 Результаты проверок критериев ПСЧ

	Хи-квадрат	Серий	Интервалов	Разбиений	Перестановок	Монотонности
lc	+	-	+	+	-	+
add	+	+	-	+	+	+
5p	+	+	+	+	+	-
lfsr	+	-	+	-	-	-
nfsr	-	-	-	-	+	+
mt	+	+	+	+	+	+
rc4	+	+	+	+	+	+
rsa	-	-	-	-	-	-
bbs	+	-	-	-	-	-

ПРИЛОЖЕНИЕ А

Код `main.py`

```
1  from functools import reduce
2
3  import numpy as np
4  import seaborn as sns
5  import pandas as pd
6  import matplotlib.pyplot as plt
7  import math
8  from scipy.stats import chi2
9
10
11 def calc_probs(u):
12     probs = dict()
13     values, counts = np.unique(u, return_counts=True)
14
15     for num, count in zip(values, counts):
16         probability = count / len(u)
17         probs[num] = probability
18
19     return probs
20
21
22 def count_elems(u):
23     element_count = {}
24     for element in u:
25         if element in element_count:
26             element_count[element] += 1
27         else:
28             element_count[element] = 1
29
30     return element_count
31
32
33 def math_exp(probs):
34     me = 0
35     for x in probs:
36         me += x * probs[x]
37     return me
38
39
40 def st_deviation(u, n):
41     average = 0
42     for x in u:
43         average += x
44     average /= n
45
```

```
46      sigma = 0
47      for x in u:
48          sigma += (x - average) * (x - average)
49      sigma = np.sqrt(sigma / n)
50
51      return sigma
52
53
54  def graph_me(u):
55      xs = [i * 500 for i in range(1, 21)]
56      ys = []
57      for x in xs:
58          ys.append(math_exp(calc_probs(u[0:x])))
59
60      absolute = abs(ys[9] - ys[-1])
61      relative = absolute / ys[9] * 100
62      print('Относительная погрешность измерения математического ожидания для выборки из {}
        ↪  элементов: '
63            '{}%'.format(xs[9], round(relative, 3)))
64
65      df = pd.DataFrame({'Объём выборки': xs, 'Математическое ожидание': ys})
66
67      sns.scatterplot(data=df, x='Объём выборки', y='Математическое ожидание')
68      plt.show()
69
70
71  def graph_sigma(u):
72      xs = [i * 500 for i in range(1, 21)]
73      ys = []
74      for x in xs:
75          ys.append(st_deviation(u[0:x], x))
76
77      absolute = abs(ys[9] - ys[-1])
78      relative = absolute / ys[9] * 100
79      print('Относительная погрешность измерения среднеквадратичного отклонения для выборки из
        ↪  {} элементов: '
80            '{}%'.format(xs[9], round(relative, 3)))
81
82      df = pd.DataFrame({'Объём выборки': xs, 'Среднеквадратичное отклонение': ys})
83
84      sns.scatterplot(data=df, x='Объём выборки', y='Среднеквадратичное отклонение')
85      plt.show()
86
87
88  def chi_square(observed=None, expected=None, k=None, alpha=0.05):
89      if observed is None:
90          observed = np.array(list(count_elems(u).values()))
91      if k is None:
```

```
92          k = observed.shape[0]
93      if expected is None:
94          expected = np.array([n / k for _ in range(k)])
95
96      diff = observed - expected
97      squared_diff = np.square(diff)
98      chi_squared = np.sum(squared_diff / expected)
99
100     critical = chi2.ppf(1 - alpha, k - 1)
101     # print(chi_squared, critical)
102     return chi_squared <= critical
103
104
105 def series(d=8):
106     observed = np.array((d * d) * [0])
107     for i in range(n // 2):
108         q, r = math.floor(u[2 * i] * d), math.floor(u[2 * i + 1] * d)
109         observed[q * d + r] += 1
110
111     expected = np.array(d ** 2 * [n / (2 * d ** 2)])
112
113     return chi_square(observed, expected, d * d)
114
115
116 def intervals(d=16):
117     j, s, count = -1, 0, 8 * [0]
118     num_of_intervals = n / 10
119
120     while s != num_of_intervals and j != n:
121         j, r = j + 1, 0
122
123         while j != n and u[j] * d < d / 2:
124             j, r = j + 1, r + 1
125
126         count[min(r, 7)] += 1
127         s += 1
128
129     if j == n:
130         raise Exception("Последовательность исчерпана, найдено %d из %d отрезков." % (s,
        ↪  num_of_intervals))
131
132     pd = 0.5
133     expected = [num_of_intervals * pd * (1.0 - pd) ** r for r in range(7)] +
    ↪  [num_of_intervals * (1.0 - pd) ** 7]
134
135     return chi_square(np.array(count), np.array(expected), 8)
136
137
```

```
138 def stirling(n, k):
139     n1 = n
140     k1 = k
141     if n <= 0:
142         return 1
143 
144     elif k <= 0:
145         return 0
146 
147     elif n == 0 and k == 0:
148         return -1
149 
150     elif n != 0 and n == k:
151         return 1
152 
153     elif n < k:
154         return 0
155 
156     else:
157         temp1 = stirling(n1 - 1, k1)
158         temp1 = k1 * temp1
159         return (k1 * (stirling(n1 - 1, k1))) + stirling(n1 - 1, k1 - 1)
160 
161 
162 def poker(d=16):
163     observed = 5 * [0]
164     for i in range(n // 5):
165         hand = [math.floor(v * d) for v in u[i * 5:i * 5 + 5]]
166         unique = [v in hand for v in range(d)]
167         distinct = reduce(lambda a, b: a + b, unique, 0)
168         observed[distinct - 1] += 1
169 
170     k = 5
171     expected = 5 * [0]
172     for r in range(1, 6):
173         p = 1.0
174         for i in range(r):
175             p *= d - i
176         expected[r - 1] = (n / 5) * (p / d ** k) * stirling(k, r)
177 
178     return chi_square(np.array(observed), np.array(expected), k)
179 
180 
181 def permutation(t=4, d=1024):
182     t_fact = math.factorial(t)
183     observed = t_fact * [0]
184     for i in range(n // t):
185         v = [v * d for v in u[t * i:t * i + t]]
```

```
        c = t * [0]
        r = t

        while r > 0:
            s = 0
            for j in range(r):
                if v[j] > v[s]:
                    s = j
            c[r - 1] = s
            v[r - 1], v[s] = v[s], v[r - 1]
            r -= 1

        f = 0
        for j in range(t - 1):
            f = (f + c[j]) * (j + 2)
        f += c[t - 1]
        observed[f] += 1

    expected = t_fact * [n / t / t_fact]

    return chi_square(np.array(observed), np.array(expected), t_fact)


def run(d=1024):
    last, ln = u[0] * d, 1
    observed = 7 * [0]
    for i in u[1:]:
        y = i * d
        if y > last:
            last, ln = y, ln + 1
        else:
            observed[min(ln, 6)] += 1
            last, ln = y, 1

    observed[min(ln, 6)] += 1

    a = [[4529.4, 9044.9, 13568.0, 18091.0, 22615.0, 27892.0],
         [9044.9, 18097.0, 27139.0, 36187.0, 45234.0, 55789.0],
         [13568.0, 27139.0, 40721.0, 54281.0, 67852.0, 83685.0],
         [18091.0, 36187.0, 54281.0, 72414.0, 90470.0, 111580.0],
         [22615.0, 45234.0, 67852.0, 90470.0, 113262.0, 139476.0],
         [27892.0, 55789.0, 83685.0, 111580.0, 139476.0, 172860.0]]
    b = [1.0 / 6.0, 5.0 / 24.0, 11.0 / 120.0, 19.0 / 720.0, 29.0 / 5040.0, 1.0 / 840.0]

    expected = 0.0
    for i in range(6):
        for j in range(6):
```

```
233             expected += (observed[i + 1] - n * b[i]) * (observed[j + 1] - n * b[j]) *
                ↪ a[i][j]
234     expected /= n
235
236     # К данному критерию не применяется хи-квадрат, вместо этого сравнивается статистика
        ↪ этого критерия
237     return 1.653 <= expected <= 12.59
238
239
240 def collision():
241     m = 128 * n
242     v = u.copy()
243     v.sort()
244     collisions = 0
245     for i in range(n - 1):
246         if abs(v[i] - v[i + 1]) <= 1 / m:
247             print(v[i], v[i + 1])
248             collisions += 1
249
250     print(collisions)
251
252     # a = n * [0]
253     # a[1] = 1
254     # j0, j1 = 1, 1
255     # for _ in range(n - 1):
256     #     j1 += 1
257     #     for j in range(j1, j0, -1):
258     #         a[j] = (j / m) * a[j] + ((1 + 1 / m) - (j / m)) * a[j - 1]
259     #         if a[j] < 10e-10:
260     #             a[j] = 0
261     #             if j == j1:
262     #                 j1 -= 1
263     #             elif j == j0:
264     #                 j0 += 1
265     # print(a)
266     # t_table = [0.01, 0.05, 0.25, 0.5, 0.75, 0.95, 0.99, 1.0]
267     # p, t = 0, 1
268     # j = j0 - 1
269     # while t != len(t_table) - 1 and p <= t_table[t]:
270     #     j += 1
271     #     p = p + a[j]
272     #     t += 1
273     #
274     # print(1 - p, n - j - 1)
275
276
277 if __name__ == '__main__':
278     rnd_name = input('Введите имя файла с ПСЧ ')
```

```
279     with open(rnd_name, 'r') as infile:
280         u = [int(i) for i in infile.read()[:-1].split(',')]
281         mx = max(u) + 1
282         u = [i / mx for i in u]
283         n = len(u)
284 
285     # graph_me(u)
286     # graph_sigma(u)
287 
288     print('Критерий хи-квадрат', chi_square())
289     print('Критерий серий', series())
290     print('Критерий интервалов', intervals())
291     print('Критерий разбиений', poker())
292     print('Критерий перестановок', permutation())
293     print('Критерий монотонности', run())
```