Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

студента 4 курса 431 группы

факультета компьютерных наук и информационных технологий

*Кузнецова Егора Дмитриевича*

фамилия, имя, отчество

Научный руководитель

Ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ И.И. Слеповичев

подпись, дата

Саратов 2024

ЛАБОРАТОРНАЯ РАБОТА

**Задание.**

1. Сгенерировать последовательность из 10000 случайных чисел из диапазона [0,1]. Исходной программой для генерации ПСЧ может быть программа, созданная в рамках практической работы по данному курсу.
2. Протестировать статистические свойства последовательности псевдослучайных чисел:
   1. Вычислить математическое ожидание последовательности;
   2. Вычислить среднеквадратичное отклонение последовательности;
   3. Сравните полученные оценки с заданными в пп. 1 параметрами. Постройте графики зависимостей оценок от объема выборки. Оцените относительные погрешности для какой-либо одной выборки.
   4. Вычислить значение и дать ответ на вопрос удовлетворяет ли ППСЧ
      1. Критерию хи-квадрат;
      2. Критерию серий;
      3. Критерию интервалов;
      4. Критерию разбиений;
      5. Критерию перестановок;
      6. Критерию монотонности;
      7. Критерию конфликтов.

# **1 Критерий хи-квадрат**

Критерий хи-квадрат широко применяется в статистических приложениях для проверки гипотезы о том, что некоторая выборка чисел подчиняется некоторому закону распределения . Критерий имеет другое название – критерий согласия Пирсона. Он является базовым для многих других критериев проверки статистических гипотез.

**Гипотеза** : случайная величина подчиняется закону распределения .

Для проверки гипотезы рассмотрим выборку, состоящую из *n* независимых наблюдений над случайной величиной *X*:

, .

По выборке построим эмпирическое распределение    случайной величины *X*. Сравнение эмпирического    и теоретического распределения    (предполагаемого в гипотезе) производится с помощью специально подобранной функции — критерия согласия. Рассмотрим критерий согласия Пирсона (критерий ):

**Гипотеза**:  порождается функцией .

Разделим [*a, b*] на *k* непересекающихся интервалов ;

Пусть  - количество наблюдений в *j*-м интервале:

– вероятность попадания наблюдения в *j*-ый интервал при выполнении гипотезы ;

*-* ожидаемое число попаданий в *j*-й интервал;

**Статистика:**

называется распределением хи-квадрат с *k*-1 степенью свободы.

В зависимости от значения критерия  , гипотеза  может приниматься либо отвергаться:

, гипотеза выполняется.

– попадает в левый «хвост» распределения, означает что теоретические и практические значения очень близки, и гипотеза принимается.

– попадает в правый «хвост» распределения, - гипотеза отвергается.

В итоге, проверка критерия для некоторой последовательности чисел (или наблюдений величины *Х*) будет состоять из следующих шагов:

1. Выполняем достаточное число независимых наблюдений.
2. Подсчитываем число наблюдений попавших в каждый из интервалов
3. Подсчитываем статистику
4. Определяем, находится ли вычисленная в доверительном интервале.

**Исходный текст программы.**

def xi\_square(seq, num\_list=None, k=None, e=None, a=0.05):  
 if num\_list is None:  
 num\_list = list(count(seq).values())  
 if k is None:  
 k = len(num\_list)  
 if e is None:  
 e = [len(seq) / k] \* k  
 xi = sum(map(lambda x: ((x[0] - x[1]) \*\* 2) / x[1], zip(num\_list, e)))  
 squa = chi2.ppf(1 - a, k - 1)  
 return xi <= squa

# **2 Критерий серий**

Критерий серий позволяет убедиться в том, что пары последовательных чисел равномерно распределены независимым образом. Проверка критерия проводится по аналогии с предыдущим случаем, однако, считать будем количество совпадений

Хи-квадрат критерий применяем к полученному набору с параметрами

**Исходный текст программы.**

def series(seq):  
 d = 64  
 a = 0.05  
 res = [0] \* (d \*\* 2)  
 for j in range(len(seq) // 2):  
 q = int(seq[2 \* j] \* d)  
 r = int(seq[2 \* j + 1] \* d)  
 res[q \* d + r] += 1  
 e = d \*\* 2 \* [len(seq) / (2 \* d \*\* 2)]  
 return xi\_square(seq=seq, num\_list=res, k=d \* d, e=e, a=a)

# **3 Критерий интервалов**

Пусть и – два действительных числа таких, что . Рассмотрим длины подпоследовательностей , в которых . Такую последовательность будем называть интервалом длины r.

Сначала, нам нужно подсчитать число интервалов длиной .

Шаги алгоритма подсчета числа интервалов:

* 1. Инициализация. Присвоить , .
  2. . Если , то переход на шаг 5.
  3. . Переход к шагу 3.
  4. Если , то , иначе – .
  5. *.* Если то переход на шаг 2.

После этого мы можем применить хи-квадрат критерий для к значениям с параметрами

Здесь – вероятность того, что . Значения *n* и *t* выбираются так, чтобы ожидаемое значение было больше 5. Критерий часто применяют для . В этом случае на шаге 3 алгоритма можно обойтись без сравнения.

**Исходный текст программы.**

def interval(seq):  
 a = 0.05  
 t = 10  
 f = 0.5  
 b = 1.0  
 j = -1  
 s = 0  
 c = t \* [0]  
 n = len(seq)  
 intervals = n / 10  
 while s < intervals and j < n:  
 r, j = 0, j + 1  
 while f <= seq[j] <= b and j < n:  
 r, j = r + 1, j + 1  
 c[min(r, t) - 1] += 1  
 s += 1  
 sub = b - f  
 e = [intervals \* sub \* (1 - sub) \*\* z for z in range(t)]  
 return xi\_square(seq=seq, num\_list=c, e=e, k=t + 1, a=a)

# **4 Критерий разбиений**

В общем случае критерия разбиений рассматриваются *n* групп *k* последовательных чисел, и подсчитывается число групп из *k* чисел с *r* различными числами. Затем применяется хи-квадрат критерий, в котором используются вероятности того, что в группе *r* различных чисел

Здесь – числа Стирлинга, задающие число способов разбиения множества из *n* элементов на *k* непересекающихся подмножеств, которые можно вычислить по формуле:

Так как вероятности очень малы, когда или 2, следует, перед применением критерия хи-квадрат, объединить несколько категорий, имеющих малые вероятности в одну.

Чтобы получить формулу для , следует подсчитать, сколько групп из *k* чисел, расположенных между 0 и , имеют точно r различных элементов, и разделить это число на .

**Исходный текст программы.**

def stirling(r, k):  
 if r <= 0 or r != 0 and r == k:  
 return 1  
 elif k <= 0 or r < k:  
 return 0  
 elif r == 0 and k == 0:  
 return - 1  
 else:  
 return k \* (stirling(r - 1, k)) + stirling(r - 1, k - 1)  
  
  
def partition(seq):  
 d = 64  
 k = 5  
 n = len(seq)  
 num\_list = [0] \* k  
 for i in range(n // k):  
 unique = list(set([int(u \* d) for u in seq[i \* k: i \* k + k]]))  
 num\_list[len(unique) - 1] += 1  
 e = [0] \* k  
 for r in range(1, k + 1):  
 p = 1.0  
 for i in range(r):  
 p \*= d - i  
 e[r - 1] = (n / k) \* (p / d \*\* k) \* stirling(k, r)  
 return xi\_square(seq=seq, num\_list=num\_list, k=k, e=e)

# **5 Критерий перестановок**

Последовательность разбивается на *n* групп по *t* элементов в каждой:

Элементы в каждой группе можно упорядочивать *t*! различными способами. Подсчитывается число групп с любым возможным порядком и применяется хи-квадрат критерий с возможными категориями и вероятностью 1/*t*! для каждой категории. Например, для , существует две категории: или . Для таких категорий будет уже шесть: или , , или , и т.д., …, или . В этом критерии предполагается, что не могут быть равны между собой.

**Исходный текст программы.**

def permutation(seq):  
 t = 10  
 a = 0.05  
 n = len(seq)  
 k = factorial(t)  
 tmp = {}  
 for i in range(0, n, t):  
 group = tuple(sorted(seq[i:i + t]))  
 tmp[group] = tmp.get(group, 0) + 1  
 num\_list = list(tmp.values())  
 e = [n / k] \* len(num\_list)  
 return xi\_square(seq=seq, num\_list=num\_list, k=k, e=e, a=a)

# **6 Критерий монотонности**

Суть метода в том, чтобы проверить длины всех восходящих (нисходящих) серий в последовательности и подсчитать для них следующую статистику

где *n* – длина последовательности, а матрицы коэффициентов следующие:

Далее, к полученной статистике применяем критерий хи-квадрат с шестью степенями свободы, когда *n* – большое (например, больше 4000).

**Исходный текст программы.**

def monoton(seq):  
 a = 0.05  
 matrix = [[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]]  
 vector = [1.0 / 6.0, 5.0 / 24.0, 11.0 / 120.0, 19.0 / 720.0, 29.0 / 5040.0, 1.0 / 840.0]  
 n = len(seq)  
 tmplst = []  
 i = 0  
 while i < n:  
 s = 1  
 while i + s < n and seq[i + s - 1] <= seq[i + s]:  
 s += 1  
 tmplst.append(s)  
 i += s  
 m = 0  
 group = {}  
 for length in tmplst:  
 m = max(m, length)  
 group[length] = group.get(length, 0) + 1  
 e = []  
 tmp = 0  
 for c in tmplst:  
 m = 1 / 6  
 min\_value = min(c, 6)  
 for i in range(min\_value):  
 for j in range(min\_value):  
 m += (seq[i + tmp] - n \* vector[i]) \* (seq[j + tmp] - n \* vector[j]) \* matrix[i][j]  
 tmp += c  
 e.append(m)  
 return xi\_square(seq=seq, e=e, a=a)

# **7 Критерий конфликтов**

Предположим, что у нас *m* урн и *n* шаров, причем *m* значительно больше *n*. Если разместить шары в урнах наугад, то некоторые урны останутся пустыми, а в некоторых будет более одного шара. Когда в одну урну попадает больше одного шара, то говорят, что произошел «конфликт». Критерий конфликтов состоит в подсчете и оценке количества конфликтов.

Рассмотрим пример, когда , а . В среднем, число урн, приходящихся на один шар – 64. Вероятность того, что в конкретную урну попадет ровно *k* шаров, равна

отсюда, среднее число конфликтов в урне вычисляется по формуле

Так как – маленькое число, получим, что общее среднее число конфликтов во всех *m* урнах намного меньше

Этот критерий хорош, когда ГПСЧ выдает строки большой размерности.

**Исходный текст программы.**

def check\_conflicts(seq):  
 n = len(seq)  
 m = pow(2, 10)  
 s = n / m  
 p0 = 1 - n / m + factorial(n) / (2 \* factorial(n - 2) \* pow(m, 2))  
 conf = n / m - 1 + p0  
 return conf < s + 10 or conf > s - 10

**Исследование последовательностей.**

Запуск программы осуществляется следующей командой:

randomness\_research.exe -file *имя\_файла\_с\_последовательностью*

* 1. Линейный конгруэнтный метод;

Генерация исследуемой последовательности:

prng.exe -g lc -i 7875 211 1663 347 -f lc.txt

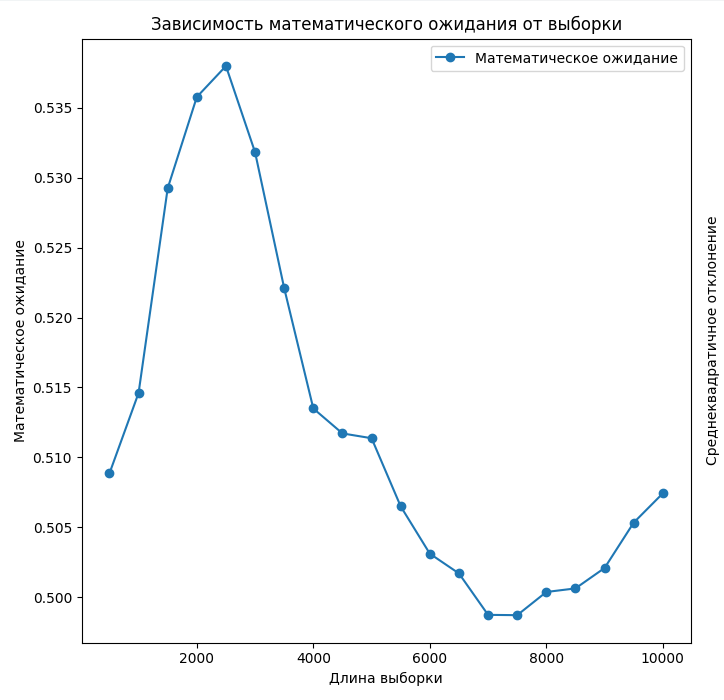
Исследование сгенерированной последовательности:

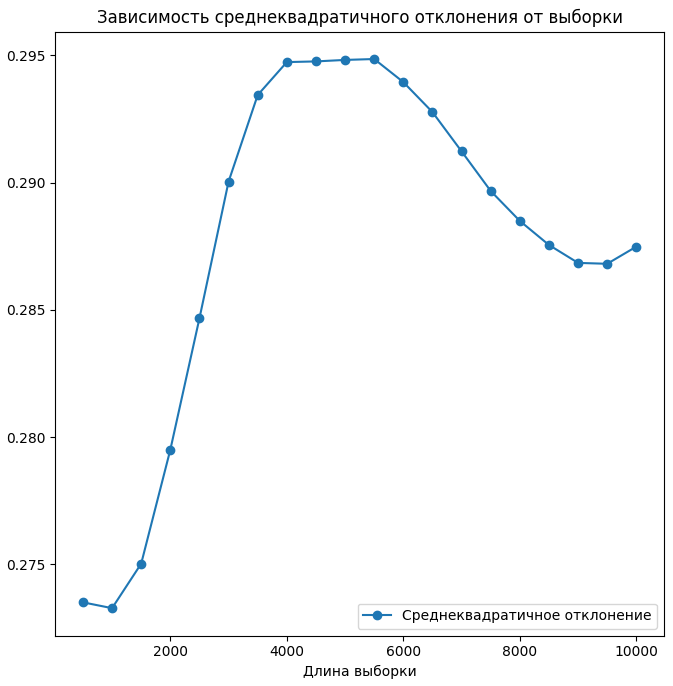
*Мат. ожидание*: 0.5074624347089102

*Cр-кв. отклонение*: 0.2874838819795578

*Относительное мат. отклонение*: 0.005664843133232737

*Относительное ср-кв. отклонение*: 0.002032818024295784





* 1. Аддитивный метод;

Генерация исследуемой последовательности:

prng.exe -g add -i 1234 19 23 4554 8697 9253 4731 6588 2864 4025 9800 8350 7638 5579 5403 9173 6399 8018 1443 9191 8998 3132 1185 3089 1356 9178 8828 8883 5223 3437 5030 1228 7052 3614 1972 6955 7056 8346 7357 8601 8179 5438 8824 5298 -f add.txt

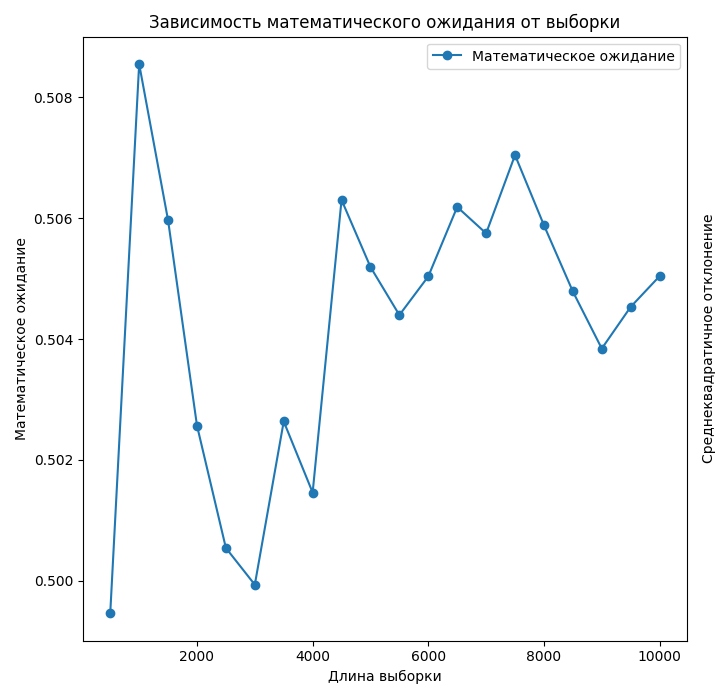
Исследование сгенерированной последовательности:

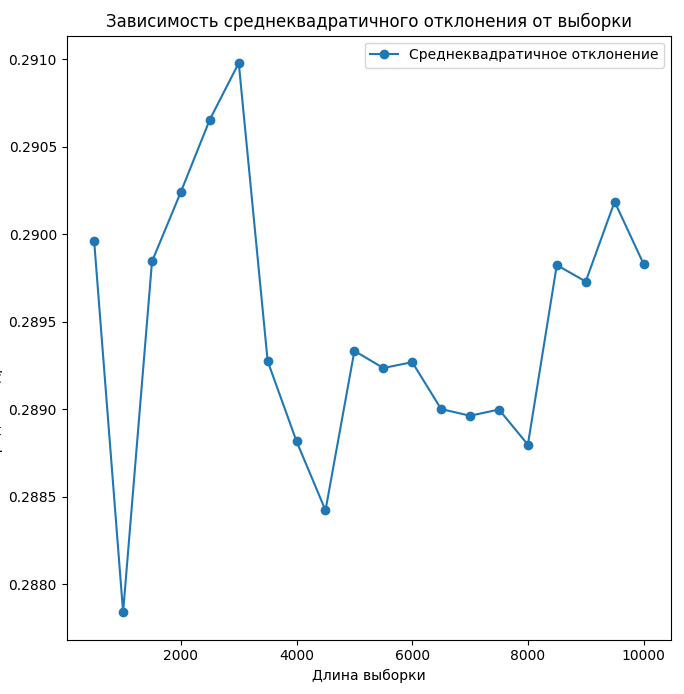
*Мат. ожидание*: 0.5050407617504051

*Cр-кв. отклонение*: 0.2898312169462395

*Относительное мат. отклонение*: 0.005738768806337976

*Относительное ср-кв. отклонение*: 0.0020495186675177557





* 1. Пятипараметрический метод;

Генерация исследуемой последовательности:

prng.exe -g 5p -i 107 31 57 82 17 10100101000011011001100001 -f 5p.txt

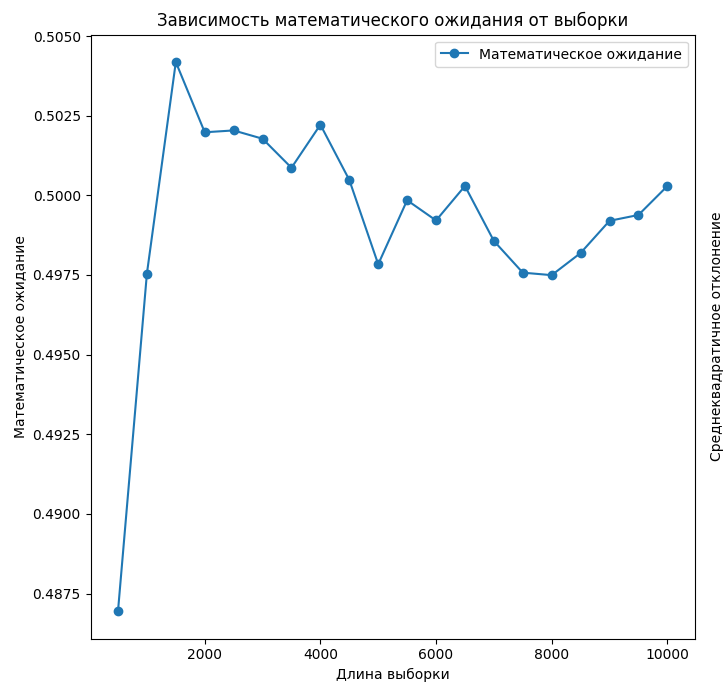
Исследование сгенерированной последовательности:

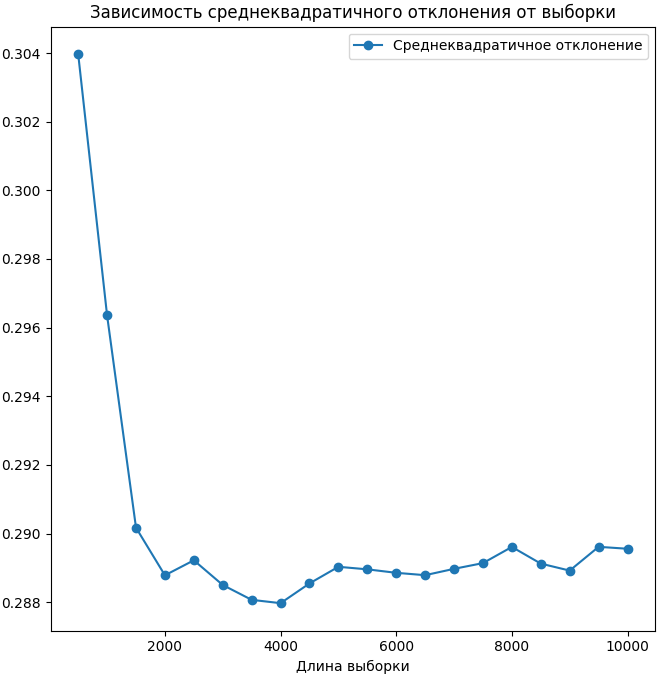
*Мат. ожидание*: 0.5002903060913086

*Cр-кв. отклонение*: 0.2895556092102124

*Относительное мат. отклонение*: 0.005787751745031118

*Относительное ср-кв. отклонение*: 0.002047569729077475





* 1. Регистр сдвига с обратной связью (РСЛОС);

Генерация исследуемой последовательности:

prng.exe -g lfsr -i 10000000011011 10010010011001 -f lfsr.txt

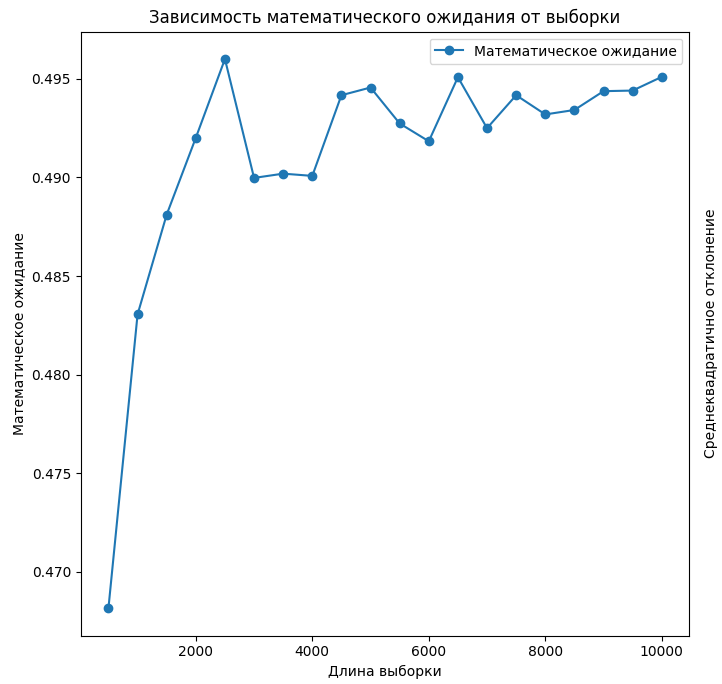
Исследование сгенерированной последовательности:

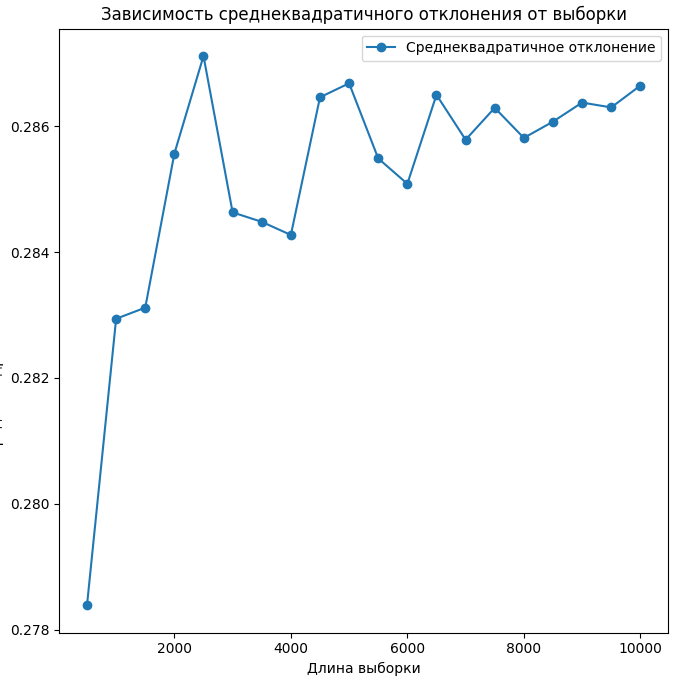
*Мат. ожидание*: 0.4951033052297165

*Cр-кв. отклонение*: 0.28664684591836226

*Относительное мат. отклонение*: 0.005789637089685045

*Относительное ср-кв. отклонение*: 0.002027000638111877





* 1. Нелинейная комбинация РСЛОС;

Генерация исследуемой последовательности:

prng.exe -g nfsr -i 10000000011011 100000000101011 1000000000010000000 23 5342 8912 23446 -f nfsr.txt

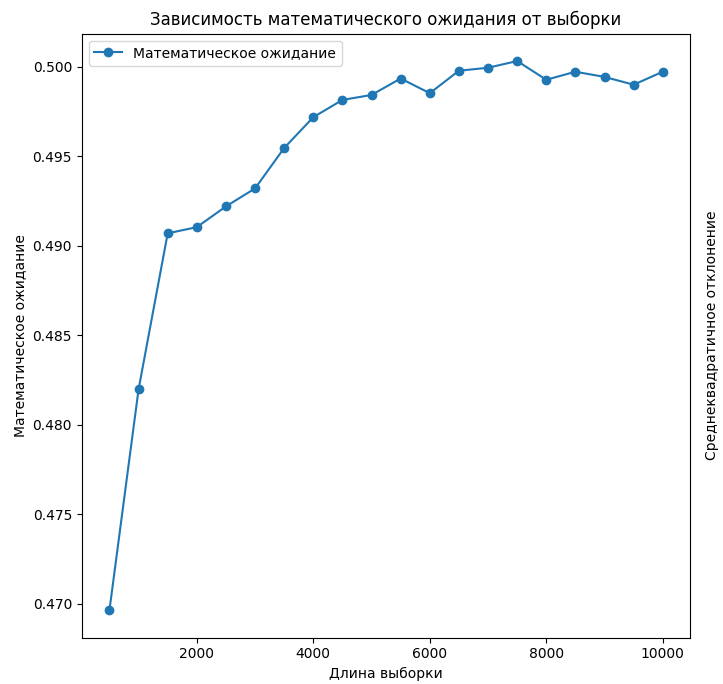
Исследование сгенерированной последовательности:

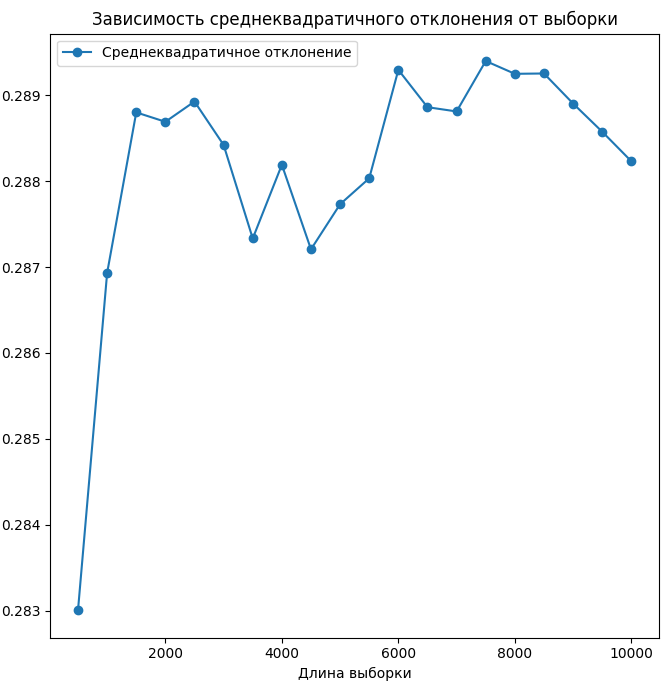
*Мат. ожидание*: 0.49972526366942766

*Cр-кв. отклонение*: 0.28823110379046296

*Относительное мат. отклонение*: 0.005767791319454488

*Относительное ср-кв. отклонение*: 0.0020382035931187374





* 1. Вихрь Мерсенна;

Генерация исследуемой последовательности:

prng.exe -g mt -i 1739 27541 -f mt.txt

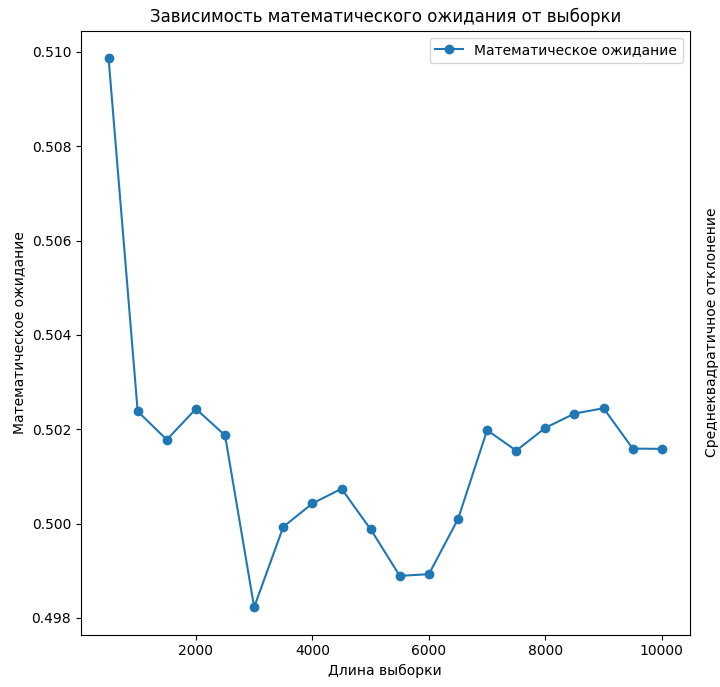
Исследование сгенерированной последовательности:

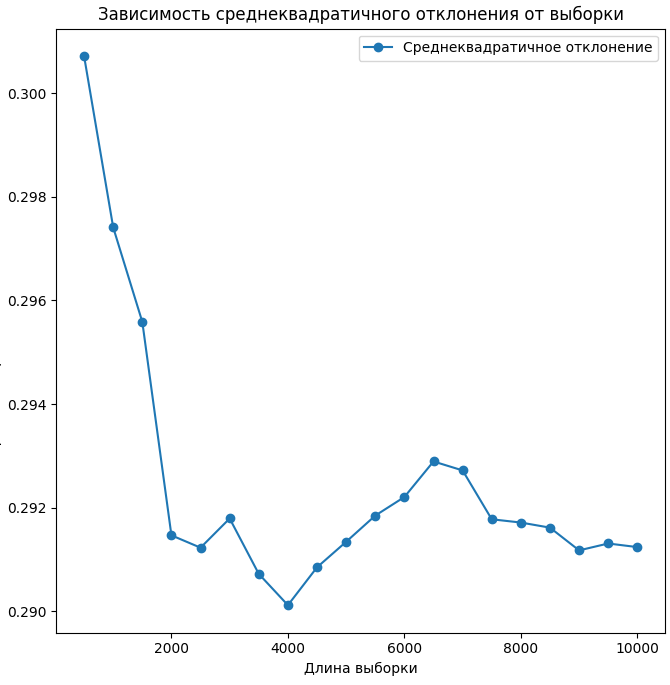
*Мат. ожидание*: 0.5015837263831029

*Cр-кв. отклонение*: 0.29123668484731147

*Относительное мат. отклонение*: 0.005806342381707751

*Относительное ср-кв. отклонение*: 0.002059457323298849





* 1. RC4;

Генерация исследуемой последовательности:

prng.exe -g rc4 -i 6351 3704 4923 1474 5645 6822 3883 176 9085 4332 6685 9105 6465 7498 7135 1332 5292 3628 652 5450 4067 5662 2147 8960 1277 172 7806 4761 4329 6177 4456 7641 4436 3240 7494 6443 5762 1058 649 5529 5519 9977 9273 1496 7921 166 8305 236 285 8714 2090 9705 8574 2402 6201 3155 5350 6151 4046 1400 6338 3189 4204 3150 4201 2413 8530 7264 6297 8383 1479 2664 1398 7844 782 9460 229 2626 2791 8092 5799 5724 7058 9785 7339 3185 1528 5291 5027 7166 5071 9610 2926 3471 1463 597 3393 1762 3512 7344 6515 1459 5912 6564 6319 6014 8143 1554 522 702 1601 2426 6722 155 938 9488 5847 9555 7628 1155 9722 4224 9394 8444 5295 1425 3543 4528 4817 1242 161 165 279 6401 7959 6240 6887 9219 5568 965 2301 9204 5893 7397 8363 2030 8487 1312 1612 9566 3730 1485 2770 5892 8043 2885 9362 9904 6786 8677 6101 1278 2943 2310 3788 1289 4660 2020 3704 3064 3774 3324 7969 7801 7260 763 1942 3162 500 5842 6723 5967 2212 2083 9458 9729 2421 1909 8299 8313 9682 4887 940 8109 1542 1550 8522 3148 1341 5363 9064 3916 7378 4343 3816 6892 5182 9718 3644 7796 6816 2675 4105 9727 8948 4167 4998 8069 2337 3061 2442 8765 193 2444 3068 4511 7936 3041 5816 6859 7561 8078 7538 9498 4181 8674 4740 4663 6282 615 1275 3779 3937 8286 4726 9381 3979 9259 1348 5102 3434 2104 2880 2195 3237 7727 -f rc4.txt

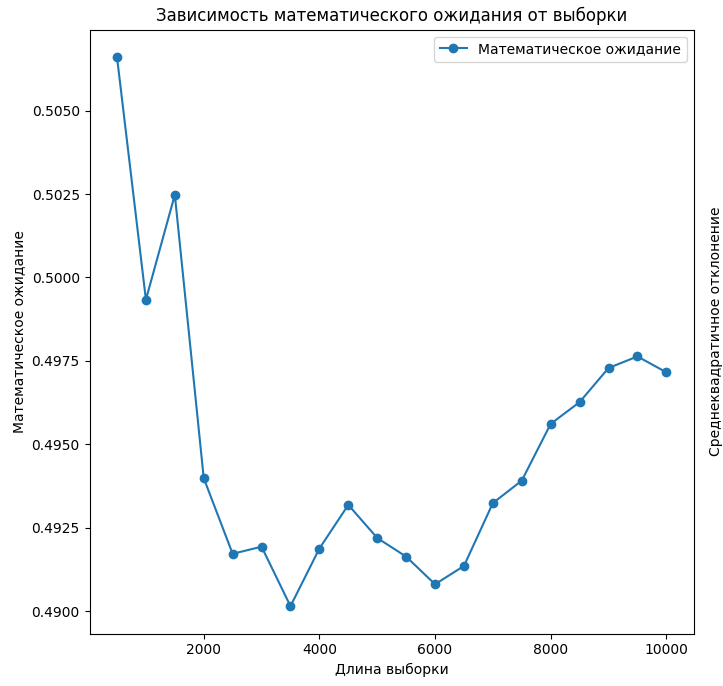
Исследование сгенерированной последовательности:

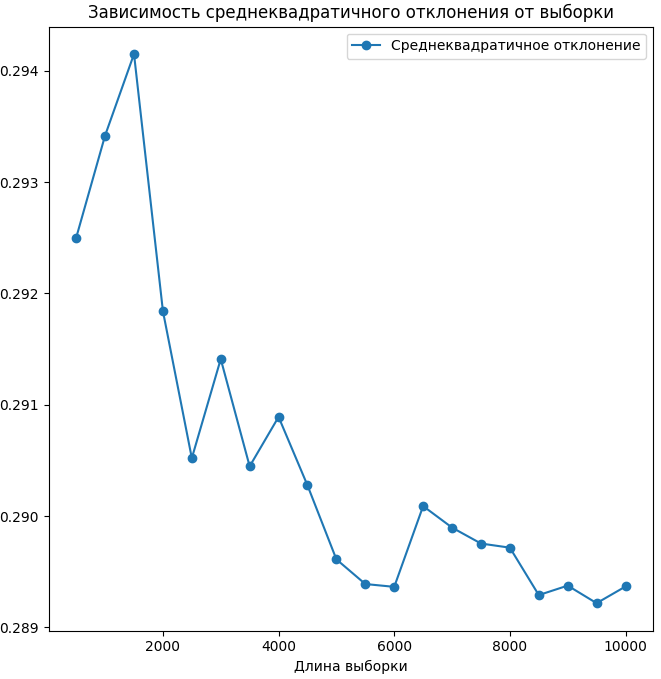
*Мат. ожидание*: 0.4971567626953125

*Cр-кв. отклонение*: 0.28936764024258466

*Относительное мат. отклонение*: 0.005820450649686254

*Относительное ср-кв. отклонение*: 0.0020462405212988027





* 1. ГПСЧ на основе RSA;

Генерация исследуемой последовательности:

prng.exe -g rsa -i 183292009640395590214888670943599889254407673336479468817820022395501420474923891839974832526263583707218142346311235634273205098061509171660517990143484111795063123224329671654802381997553865015352836802354736706323154667318615994555262815691223037591248696879026124585947695957972205849026957552289 721740113940525878464693165169605984115746921202274439947798437099874098831534693267276910373674126915339893910906955771122520186649471327469569713468941776292589145842712981890451759924566220243844660846376820048287571598677174279462665931090123124139379039091295993702208396683165163952858982396937 9 738757207292049099681186263047264483621186264658013413340565144230403446314908771216666695670040613851377887665768240265717648919534745092259554784688980071746560036939207049777629182048114446081198394504024887869085513953906106436793914226972714932196077148543042413224732428045682566262016086615551 -f rsa.txt

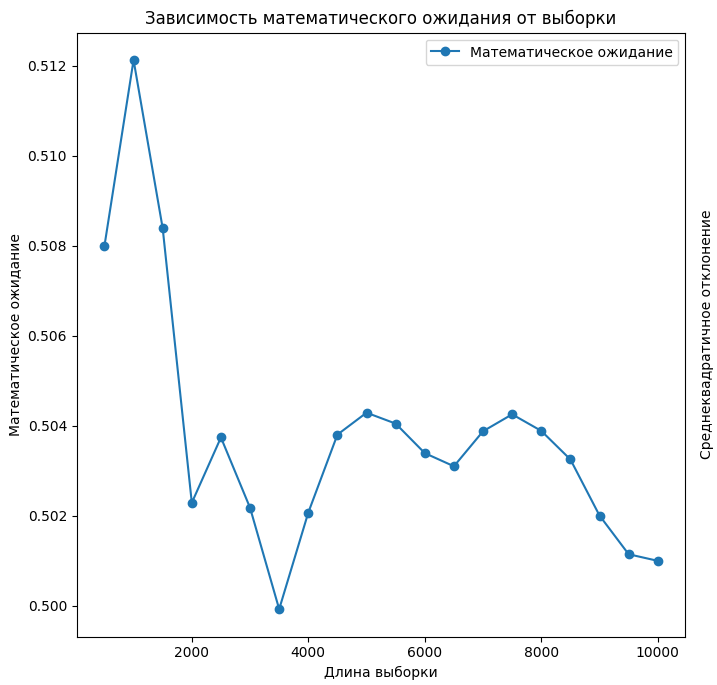
Исследование сгенерированной последовательности:

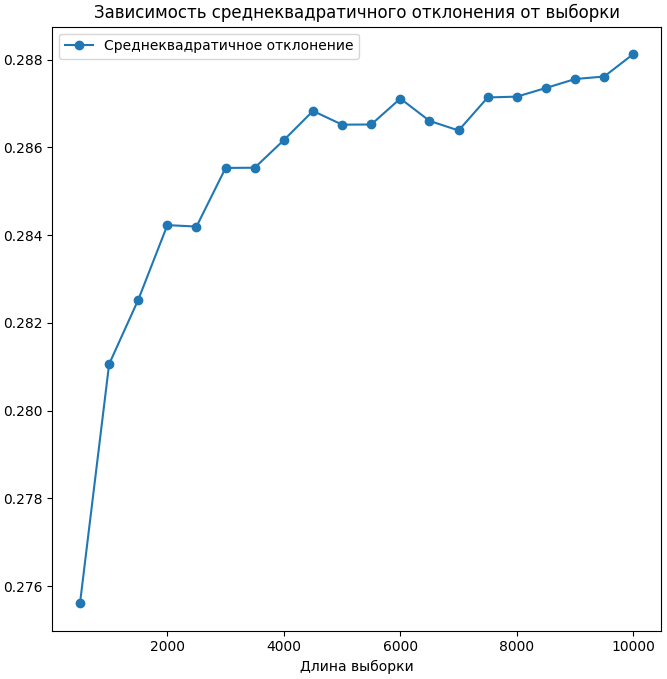
*Мат. ожидание*: 0.50099375

*Cр-кв. отклонение*: 0.2881256920931884

*Относительное мат. отклонение*: 0.005751083563281745

*Относительное ср-кв. отклонение*: 0.0020374581825876865





* 1. Алгоритм Блюма-Блюма-Шуба;

Генерация исследуемой последовательности:

prng.exe -g bbs -i 24 13 -f bbs.txt

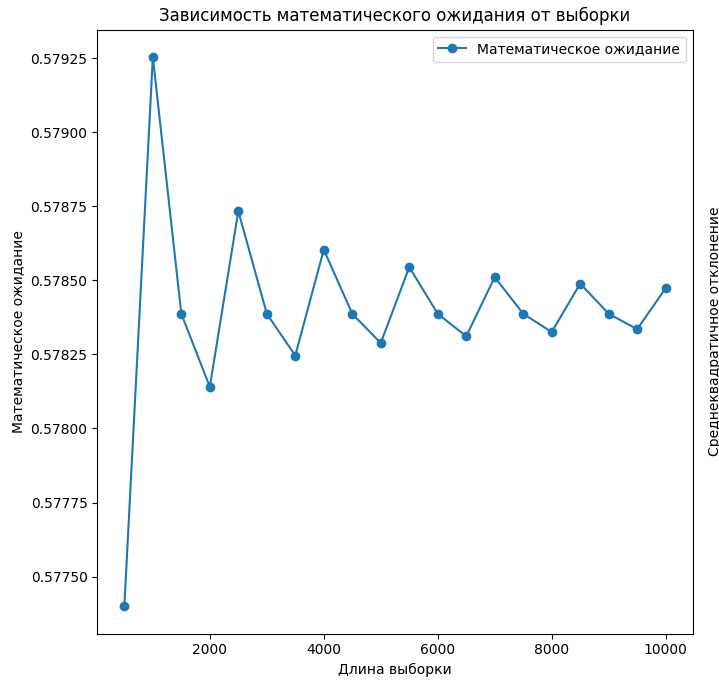
Исследование сгенерированной последовательности:

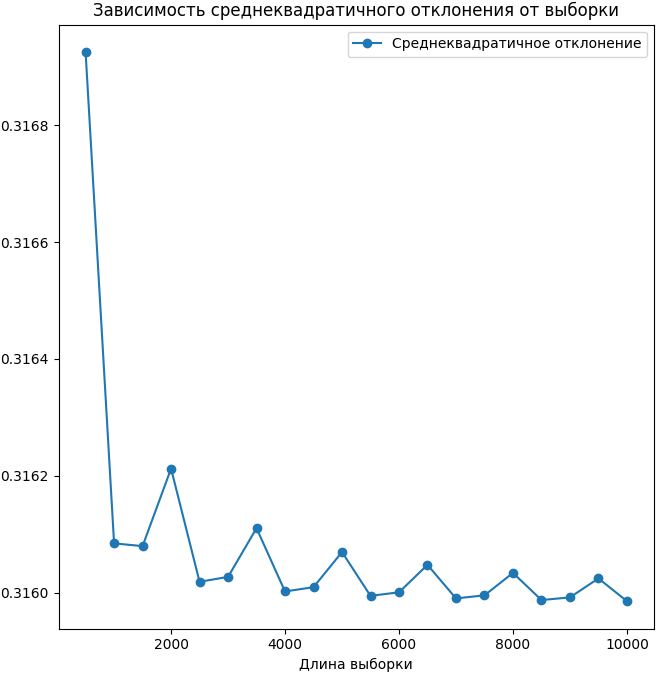
*Мат. ожидание*: 0.5784738844393593

*Cр-кв. отклонение*: 0.31598525819349954

*Относительное мат. отклонение*: 0.005462394529698494

*Относительное ср-кв. отклонение*: 0.0022344649142749906





## **Проверка ПСП критериями**

Таблица 1. Результаты проверки ПСП различными критериями

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | lc | add | 5p | lfsr | nfsr | mt | rc4 | rsa | bbs |
| Хи-квадрат | + | + | + | + | + | + | + | + | + |
| серий | + | + | + | - | + | + | + | + | - |
| интервалов | - | - | - | - | - | - | - | - | - |
| разбиений | - | + | + | - | - | + | + | + | - |
| перестановок | + | + | + | + | + | + | + | + | - |
| монотонности | - | - | - | - | - | - | - | - | - |
| конфликтов | + | + | + | + | + | + | + | + | + |

ПРИЛОЖЕНИЕ А

Листинг программы randomness\_research.exe

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.stats import chi2  
from math import factorial, floor  
import argparse  
  
  
def mean(seq):  
 return np.mean(seq)  
  
  
def relative\_error\_mean(seq):  
 mean\_val = mean(seq)  
 std\_dev = np.std(seq, ddof=1)  
 relative\_error = std\_dev / np.sqrt(len(seq)) / mean\_val  
 return relative\_error  
  
  
def std\_dev(seq):  
 return np.std(seq, ddof=1)  
  
  
def relative\_error\_std\_dev(seq):  
 std\_dev\_val = std\_dev(seq)  
 n = len(seq)  
 relative\_error = std\_dev\_val / np.sqrt(2 \* (n - 1))  
 return relative\_error  
  
  
def normalize(seq):  
 min\_val = np.min(seq)  
 max\_val = np.max(seq)  
 return (seq - min\_val) / (max\_val - min\_val + 1)  
  
  
def plot\_statistics(seq, step=500):  
 normalized\_seq = normalize(abs(seq)) if np.max(seq) > 1 else seq  
 sample\_sizes = list(range(step, len(normalized\_seq) + 1, step))  
 means = []  
 std\_devs = []  
  
 for size in sample\_sizes:  
 sample = normalized\_seq[:size]  
 means.append(mean(sample))  
 std\_devs.append(std\_dev(sample))  
  
 plt.figure(figsize=(14, 7))  
  
 plt.subplot(1, 2, 1)  
 plt.plot(sample\_sizes, means, label='Математическое ожидание', marker='o')  
 plt.xlabel('Длина выборки')  
 plt.ylabel('Математическое ожидание')  
 plt.title('Зависимость математического ожидания от выборки')  
 plt.legend()  
  
 plt.subplot(1, 2, 2)  
 plt.plot(sample\_sizes, std\_devs, label='Среднеквадратичное отклонение', marker='o')  
 plt.xlabel('Длина выборки')  
 plt.ylabel('Среднеквадратичное отклонение')  
 plt.title('Зависимость среднеквадратичного отклонения от выборки')  
 plt.legend()  
  
 plt.tight\_layout()  
 plt.show()  
  
  
def count(seq):  
 lst = {}  
 for x in seq:  
 if x in lst:  
 lst[x] += 1  
 else:  
 lst[x] = 1  
 return lst  
  
  
def xi\_square(seq, num\_list=None, k=None, e=None, a=0.05):  
 if num\_list is None:  
 num\_list = list(count(seq).values())  
 if k is None:  
 k = len(num\_list)  
 if e is None:  
 e = [len(seq) / k] \* k  
 xi = sum(map(lambda x: ((x[0] - x[1]) \*\* 2) / x[1], zip(num\_list, e)))  
 squa = chi2.ppf(1 - a, k - 1)  
 return xi <= squa  
  
  
def series(seq):  
 d = 64  
 a = 0.05  
 res = [0] \* (d \*\* 2)  
 for j in range(len(seq) // 2):  
 q = int(seq[2 \* j] \* d)  
 r = int(seq[2 \* j + 1] \* d)  
 res[q \* d + r] += 1  
 e = d \*\* 2 \* [len(seq) / (2 \* d \*\* 2)]  
 return xi\_square(seq=seq, num\_list=res, k=d \* d, e=e, a=a)  
  
  
def interval(seq):  
 a = 0.05  
 t = 10  
 f = 0.5  
 b = 1.0  
 j = -1  
 s = 0  
 c = t \* [0]  
 n = len(seq)  
 intervals = n / 10  
 while s < intervals and j < n:  
 r, j = 0, j + 1  
 while f <= seq[j] <= b and j < n:  
 r, j = r + 1, j + 1  
 c[min(r, t) - 1] += 1  
 s += 1  
 sub = b - f  
 e = [intervals \* sub \* (1 - sub) \*\* z for z in range(t)]  
 return xi\_square(seq=seq, num\_list=c, e=e, k=t + 1, a=a)  
  
  
def stirling(r, k):  
 if r <= 0 or r != 0 and r == k:  
 return 1  
 elif k <= 0 or r < k:  
 return 0  
 elif r == 0 and k == 0:  
 return - 1  
 else:  
 return k \* (stirling(r - 1, k)) + stirling(r - 1, k - 1)  
  
  
def partition(seq):  
 d = 64  
 k = 5  
 n = len(seq)  
 num\_list = [0] \* k  
 for i in range(n // k):  
 unique = list(set([int(u \* d) for u in seq[i \* k: i \* k + k]]))  
 num\_list[len(unique) - 1] += 1  
 e = [0] \* k  
 for r in range(1, k + 1):  
 p = 1.0  
 for i in range(r):  
 p \*= d - i  
 e[r - 1] = (n / k) \* (p / d \*\* k) \* stirling(k, r)  
 return xi\_square(seq=seq, num\_list=num\_list, k=k, e=e)  
  
  
def permutation(seq):  
 t = 10  
 a = 0.05  
 n = len(seq)  
 k = factorial(t)  
 tmp = {}  
 for i in range(0, n, t):  
 group = tuple(sorted(seq[i:i + t]))  
 tmp[group] = tmp.get(group, 0) + 1  
 num\_list = list(tmp.values())  
 e = [n / k] \* len(num\_list)  
 return xi\_square(seq=seq, num\_list=num\_list, k=k, e=e, a=a)  
  
  
def monoton(seq):  
 a = 0.05  
 matrix = [[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]]  
 vector = [1.0 / 6.0, 5.0 / 24.0, 11.0 / 120.0, 19.0 / 720.0, 29.0 / 5040.0, 1.0 / 840.0]  
 n = len(seq)  
 tmplst = []  
 i = 0  
 while i < n:  
 s = 1  
 while i + s < n and seq[i + s - 1] <= seq[i + s]:  
 s += 1  
 tmplst.append(s)  
 i += s  
 m = 0  
 group = {}  
 for length in tmplst:  
 m = max(m, length)  
 group[length] = group.get(length, 0) + 1  
 e = []  
 tmp = 0  
 for c in tmplst:  
 m = 1 / 6  
 min\_value = min(c, 6)  
 for i in range(min\_value):  
 for j in range(min\_value):  
 m += (seq[i + tmp] - n \* vector[i]) \* (seq[j + tmp] - n \* vector[j]) \* matrix[i][j]  
 tmp += c  
 e.append(m)  
 return xi\_square(seq=seq, e=e, a=a)  
  
  
def check\_conflicts(seq):  
 n = len(seq)  
 m = pow(2, 10)  
 s = n / m  
 p0 = 1 - n / m + factorial(n) / (2 \* factorial(n - 2) \* pow(m, 2))  
 conf = n / m - 1 + p0  
 return conf < s + 10 or conf > s - 10  
  
  
def get\_data\_from\_file(file\_path):  
 with open(file\_path, "r") as file:  
 data\_line = file.readline()  
 num\_lst = list(map(int, data\_line.strip().split(",")))  
 return normalize(num\_lst)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser(  
 prog="check.py",  
 description="Проверка последовательности чисел по заданному критерию"  
 )  
 help1 = "Имя файла с входной последовательностью"  
 help2 = """ Критерии для проверки заданной последовательности на равномерное распределение:  
 • chi - Критерий хи-квадрат,  
 • series - Критерий серий,  
 • intervals - Критерий интервалов,  
 • partition - Критерий разбиений,  
 • permutation - Критерий перестановок,  
 • monoton - Критерий монотонности,  
 • conflict - Критерий конфликтов  
 """  
 types\_crit = ["chi", "series", "intervals", "partition", "permutation", "monoton", "conflict"]  
 parser.add\_argument("-type", nargs=1, choices=types\_crit, required=False, default=[None], help=help2)  
 parser.add\_argument("-file", nargs=1, required=False, default=[None], help=help1)  
 args = parser.parse\_args()  
 type\_criteries = args.type[0]  
 file\_path = args.file[0]  
 try:  
 seq = get\_data\_from\_file(file\_path)  
 num1 = mean(seq)  
 num2 = std\_dev(seq)  
 num3 = relative\_error\_mean(seq)  
 num4 = relative\_error\_std\_dev(seq)  
 print(f"Мат. ожидание: {num1}")  
 print(f"Cр-кв. отклонение: {num2}")  
 print(f"Относительное мат. отклонение: {num3}")  
 print(f"Относительное ср-кв. отклонение: {num4}")  
 if type\_criteries == None:  
 print("Критерий хи-квадрат:")  
 if (xi\_square(seq)):  
 print("• Удовлетворяет критерию хи-квадрат")  
 else:  
 print("• Не удовлетворяет критерию хи-квадрат")  
 print("Критерий серий:")  
 if series(seq):  
 print("• Удовлетворяет критерию серий")  
 else:  
 print("• Не удовлетворяет критерию серий")  
 print("Критерий интервалов:")  
 if interval(seq):  
 print("• Удовлетворяет критерию интервалов")  
 else:  
 print("• Не удовлетворяет критерию интервалов")  
 print("Критерий разбиений:")  
 if partition(seq):  
 print("• Удовлетворяет критерию разбиений")  
 else:  
 print("• Не удовлетворяет критерию разбиений")  
 print("Критерий перестановок:")  
 if permutation(seq):  
 print("• Удовлетворяет критерию перестановок")  
 else:  
 print("• Не удовлетворяет критерию перестановок")  
 print("Критерий монотонности:")  
 if monoton(seq):  
 print("• Удовлетворяет критерию монотонности")  
 else:  
 print("• Не удовлетворяет критерию монотонности")  
 print("Критерий конфликтов:")  
 if check\_conflicts(seq):  
 print("• Удовлетворяет критерию конфликтов")  
 else:  
 print("• Не удовлетворяет критерию конфликтов")  
 else:  
 match type\_criteries:  
 case 'chi':  
 print("Критерий хи-квадрат:")  
 if (xi\_square(seq)):  
 print("• Удовлетворяет критерию хи-квадрат")  
 else:  
 print("• Не удовлетворяет критерию хи-квадрат")  
 case 'series':  
 print("Критерий серий:")  
 if series(seq):  
 print("• Удовлетворяет критерию серий")  
 else:  
 print("• Не удовлетворяет критерию серий")  
 case 'intervals':  
 print("Критерий интервалов:")  
 if interval(seq):  
 print("• Удовлетворяет критерию интервалов")  
 else:  
 print("• Не удовлетворяет критерию интервалов")  
 case 'partition':  
 print("Критерий разбиений:")  
 if partition(seq):  
 print("• Удовлетворяет критерию разбиений")  
 else:  
 print("• Не удовлетворяет критерию разбиений")  
 case 'permutation':  
 print("Критерий перестановок:")  
 if permutation(seq):  
 print("• Удовлетворяет критерию перестановок")  
 else:  
 print("• Не удовлетворяет критерию перестановок")  
 case 'monoton':  
 print("Критерий монотонности:")  
 if monoton(seq):  
 print("• Удовлетворяет критерию монотонности")  
 else:  
 print("• Не удовлетворяет критерию монотонности")  
 case 'conflict':  
 print("Критерий конфликтов:")  
 if check\_conflicts(seq):  
 print("• Удовлетворяет критерию конфликтов")  
 else:  
 print("• Не удовлетворяет критерию конфликтов")  
 except Exception as err:  
 print("В процессе проверки произошла ошибка!")  
 print(" • " + str(err))  
 plot\_statistics(seq)