## МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

# ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ

## ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

студента 4 курса 431 группы

## специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

## Cеребрякова Алексея Владимировича

Преподаватель Ассистент

подпись, дата

## Саратов 2023

Н.А. Артемова

# 1 Тестирование статистических свойств последовательности псевдослучайных чисел

## Описание задания:

Протестировать статистические свойства последовательности псевдослу- чайных чисел:

## Вычислить математическое ожидание последовательности;

1. Вычислить среднеквадратичное отклонение последовательности;

## Сравните полученные оценки с заданными в пп. 1 параметрами. Постройте графики зависимостей оценок от объема выборки. Оцените относитель- ные погрешности для какой-либо одной выборки.

1. Вычислить значение и дать ответ на вопрос удовлетворяет ли ППСЧ

## *а*) Критерию хи-квадрат;

*б*) Критерию серий;

## *в*) Критерию интервалов;

*г*) Критерию разбиений;

## *д*) Критерию перестановок; *е*) Критерию монотонности; *ж*) Критерию конфликтов.

Описание используемых ПСЧ:

## В работе рассматриваются последовательности псевдослучайных вели- чин, сгенерированные в первой части практической работы следующими алго- ритмами:

1. Линейный конгруэнтный метод;

## Аддитивный метод;

1. Пятипараметрический метод;

## Регистр сдвига с обратной связью (РСЛОС);

1. Нелинейная комбинация РСЛОС;

## Вихрь Мерсенна;

1. RC4;

## ГПСЧ на основе RSA;

1. Алгоритм Блюма-Блюма-Шуба.

## После генерации данные последовательности с помощью программы из второй части практической работы были приведены к стандартному равномер-

ному распределению с входными параметрами 0 и 0,999, т.е. каждое число *c* из последовательности .

# Мат. ожидание, среднекв. отклонение, сравнение с теоретически- ми оценками и построение графиков

Для того, чтобы найти **математическое ожидание** необходимо восполь- зоваться следующей формулой:

## где - числа последовательности, - количество таких чисел.

Для того, чтобы найти **среднеквадратичное отклонение** необходимо вос- пользоваться следующей формулой:

## где - числа последовательности, - количество таких чисел, - математическое ожидание.

Теоретическая оценка при равномерном распределении для мат. ожидания равна 0.5, для среднеквадратичного отклонения примерно равна 0.2887.

## Далее необходимо посчитать относительную погрешность для одной вы- борки. Для этого необходимо сравнить полученные результаты с эталонными и вычислить погрешности согласно следующим формулам:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Генератор | Мат.Ожидание | Среднекв.отклонение | Погрешность | |
| Мат.Ожидания | Среднекв. отклонения |
| lc | 0.4922 | 0.2894 | 0.0158 | 0.0024 |
| add | 0.4921 | 0.2887 | 0.0161 | 0.0010 |
| 5p | 0.4950 | 0.2895 | 0.0101 | 0.0028 |
| lfsr | 0.4948 | 0.2902 | 0.0105 | 0.0052 |
| nfsr | 0.4988 | 0.2884 | 0.0024 | 0.0010 |
| mt | 0.5013 | 0.2878 | 0.0026 | 0.0031 |
| rc4 | 0.5028 | 0.2903 | 0.0056 | 0.0055 |
| rsa | 0.4990 | 0.2876 | 0.0020 | 0.0038 |
| bbs | 0.5049 | 0.2891 | 0.0097 | 0.0014 |



Рисунок 1 – Вычисление показателей для генератора lc



Рисунок 2 – Вычисление показателей для генератора add



Рисунок 3 – Вычисление показателей для генератора 5p



Рисунок 4 – Вычисление показателей для генератора lfsr



Рисунок 5 – Вычисление показателей для генератора nfsr



Рисунок 6 – Вычисление показателей для генератора mt



Рисунок 7 – Вычисление показателей для генератора rc4



Рисунок 8 – Вычисление показателей для генератора rsa



Рисунок 9 – Вычисление показателей для генератора bbs

## Далее построим графики зависимости мат. ожидания от объема выборки и среднеквадратичного отклонения от объема выборки. Шаг: 50.

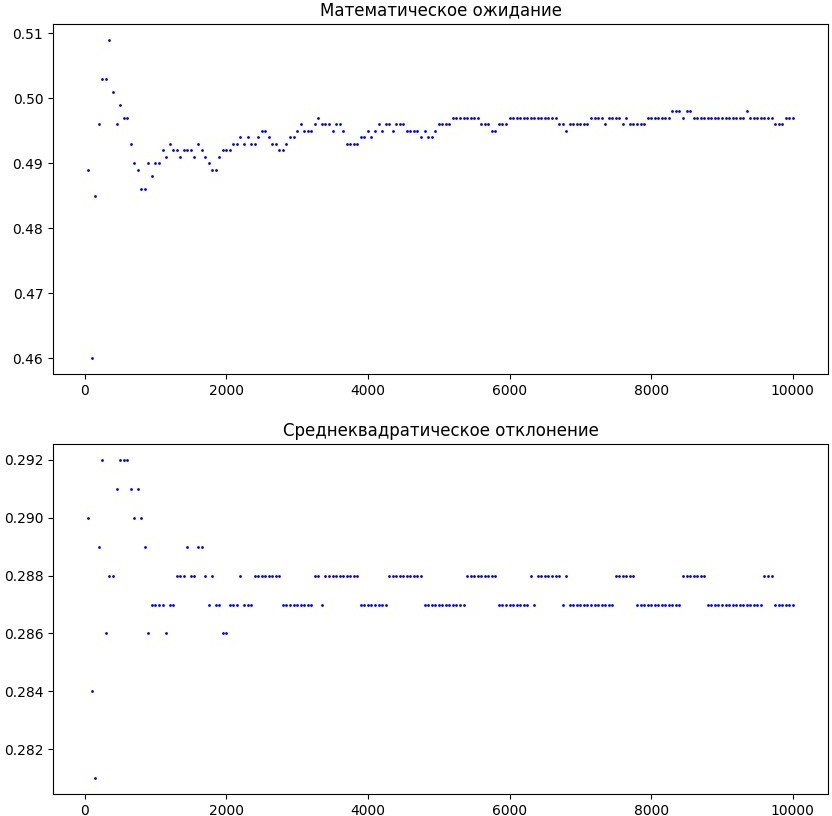


Рисунок 10 – Графики зависимости для генератора lc

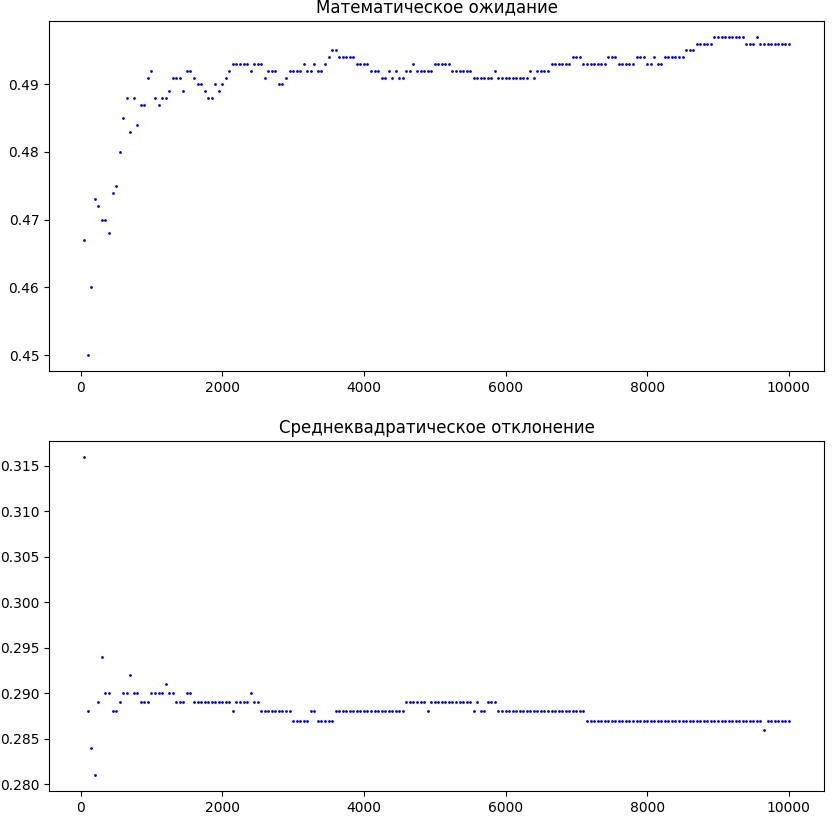


Рисунок 11 – Графики зависимости для генератора add

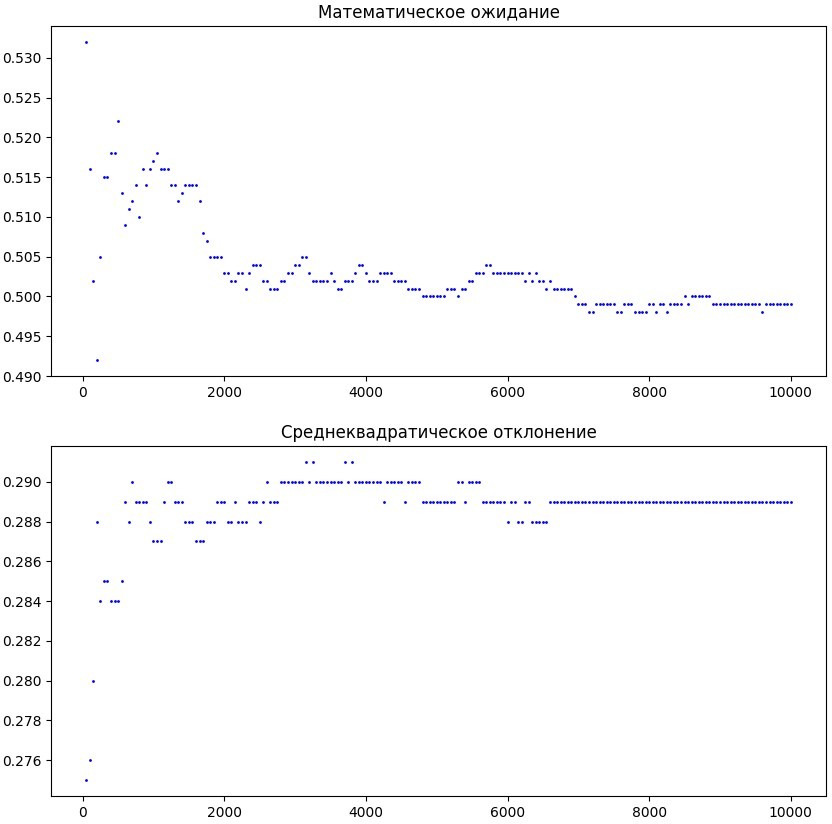


Рисунок 12 – Графики зависимости для генератора 5p

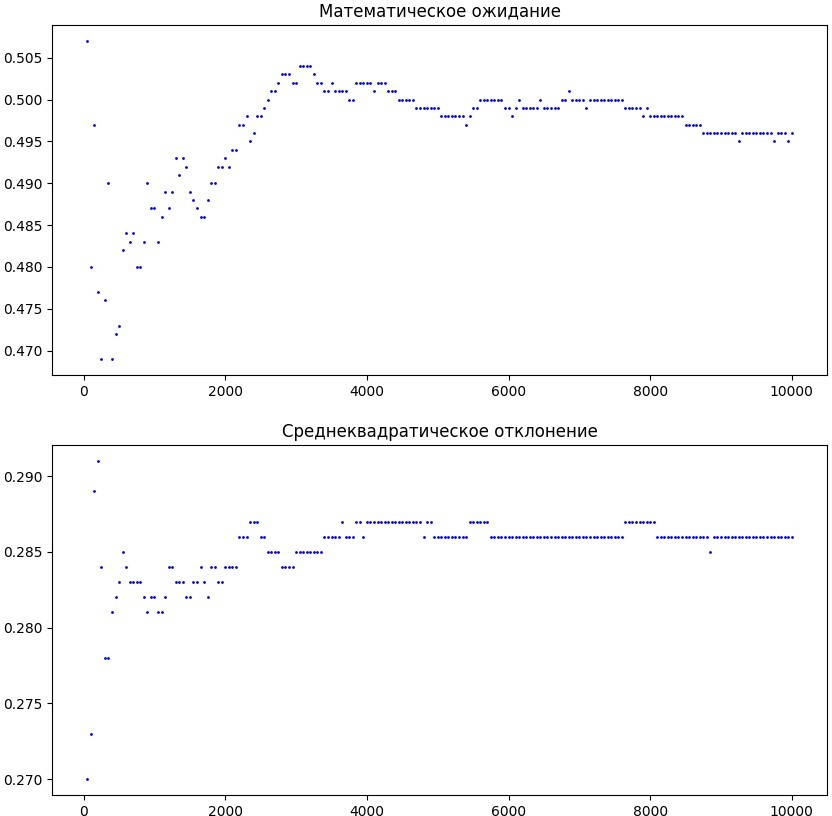


Рисунок 13 – Графики зависимости для генератора lfsr

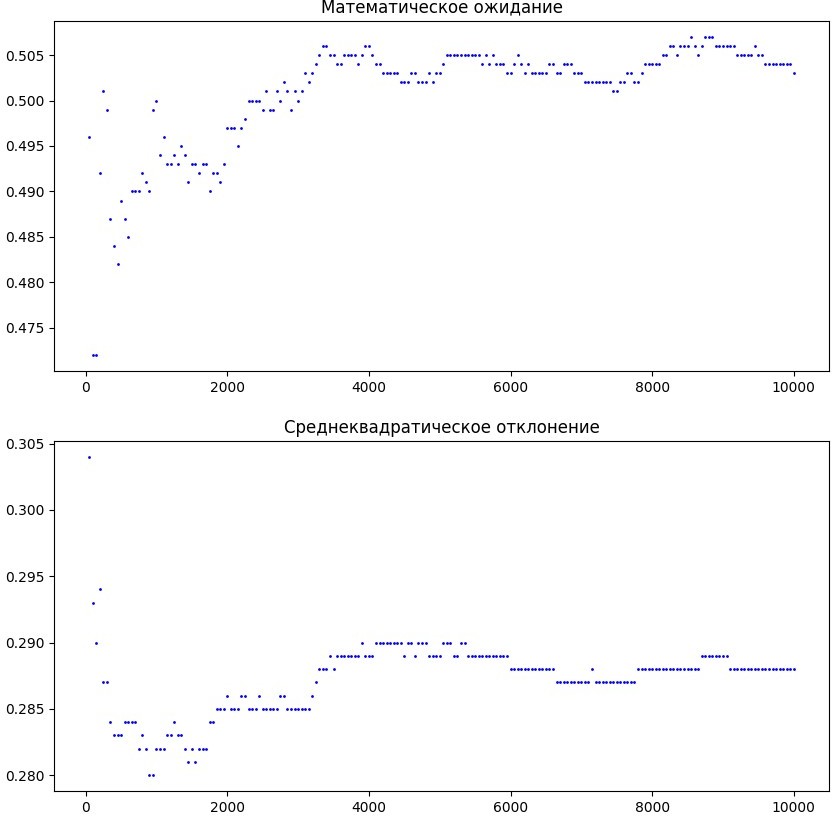


Рисунок 14 – Графики зависимости для генератора nfsr

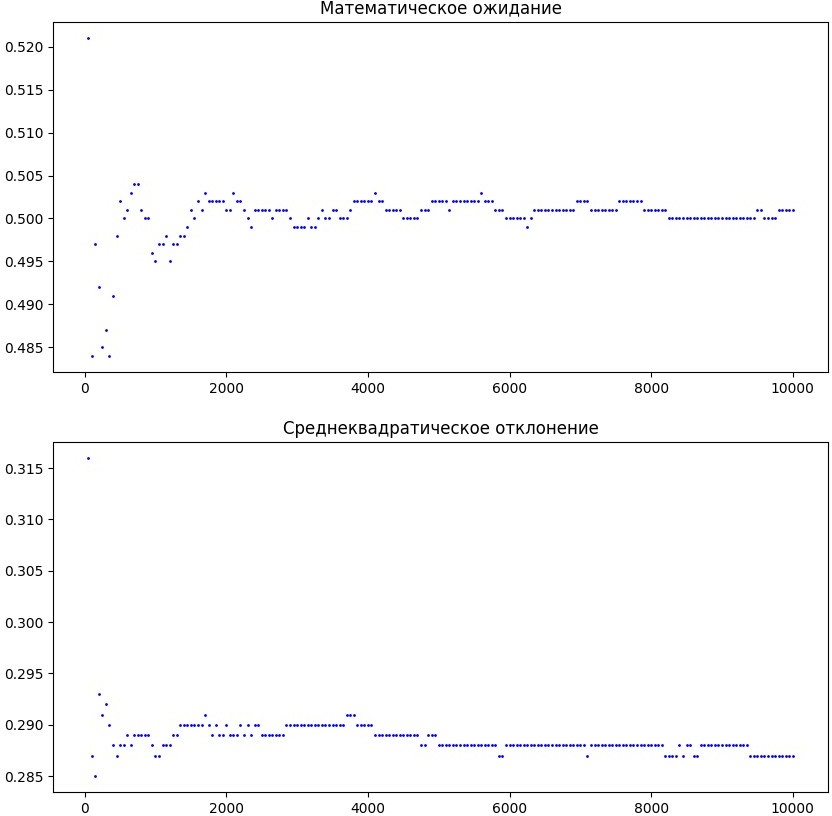


Рисунок 15 – Графики зависимости для генератора mt

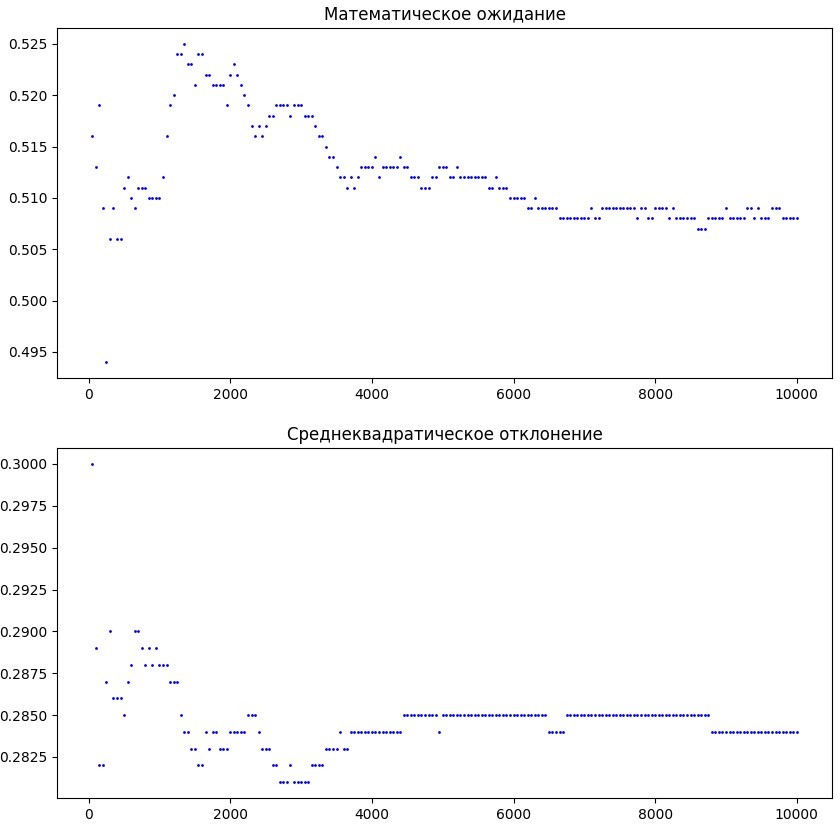


Рисунок 16 – Графики зависимости для генератора rc4

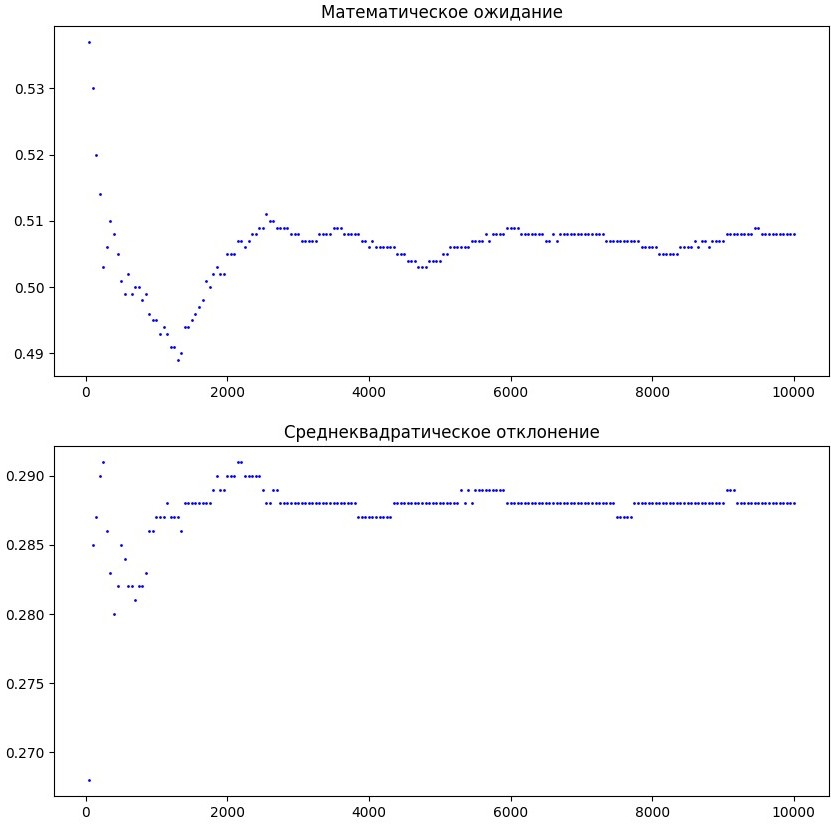


Рисунок 17 – Графики зависимости для генератора rsa

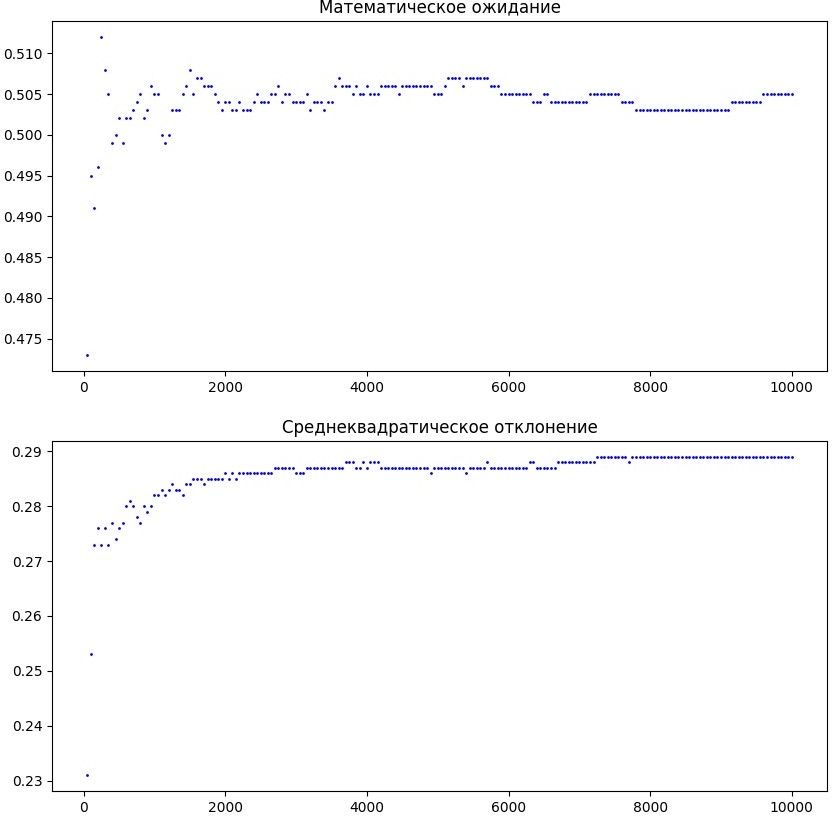


Рисунок 18 – Графики зависимости для генератора bbs

# Сравнение полученных оценок с теоретическими и построение графиков

* 1. **Проверка критериев**

## Критерий хи-квадрат Описание критерия:

Проверка критерия *χ*2 для некоторой последовательности чисел (или на- блюдений величины Х) будет состоять из следующих шагов:

## Выполняем достаточное число независимых наблюдений.

1. Подсчитываем число наблюдений попавших в каждый из интервалов

.

## Подсчитываем статистику

## Определяем, находится ли вычисленная в доверительном интервале.

Приведем результаты выполнения программы для каждого из генерато-

## ров:

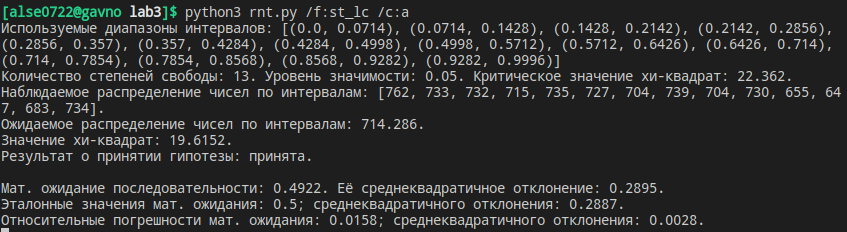


Рисунок 19 – Результаты выполнения программы для генератора lc

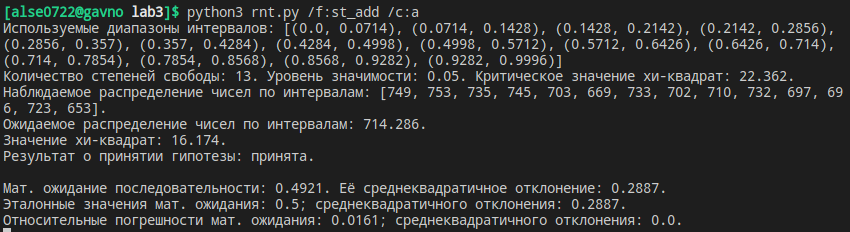


Рисунок 20 – Результаты выполнения программы для генератора add

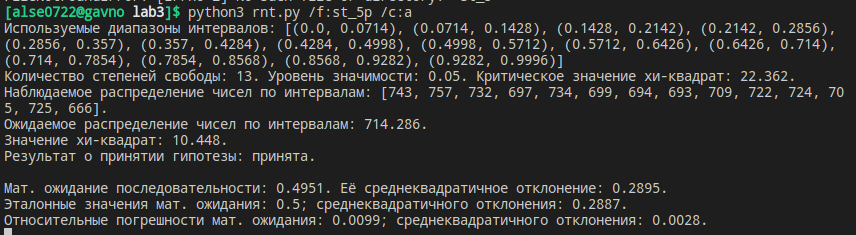


Рисунок 21 – Результаты выполнения программы для генератора 5p

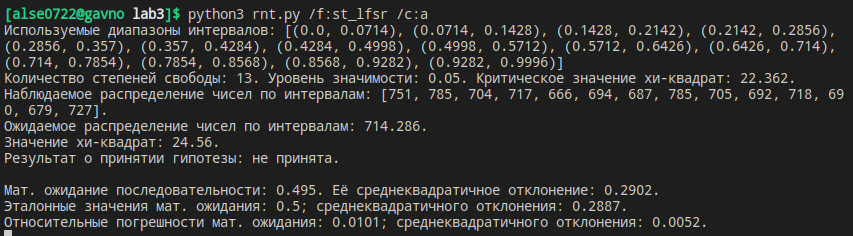


Рисунок 22 – Результаты выполнения программы для генератора lfsr

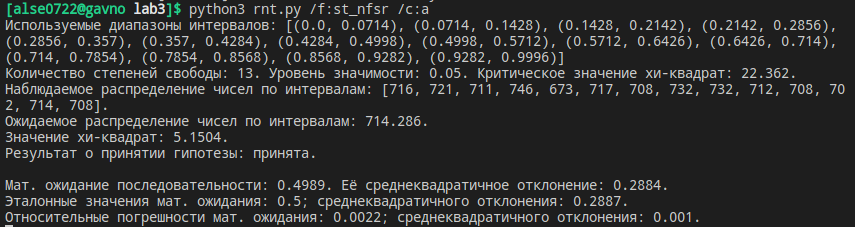


Рисунок 23 – Результаты выполнения программы для генератора nfsr

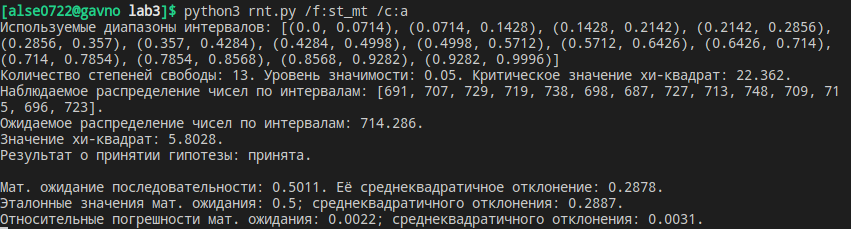


Рисунок 24 – Результаты выполнения программы для генератора mt

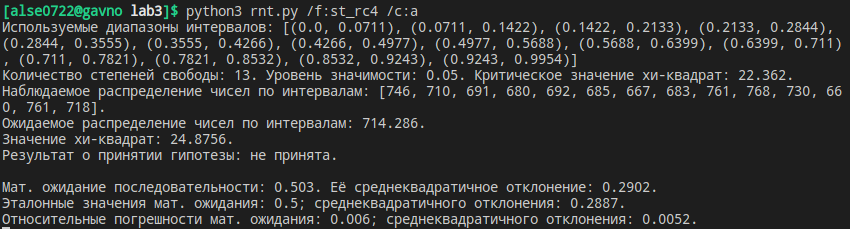


Рисунок 25 – Результаты выполнения программы для генератора rc4

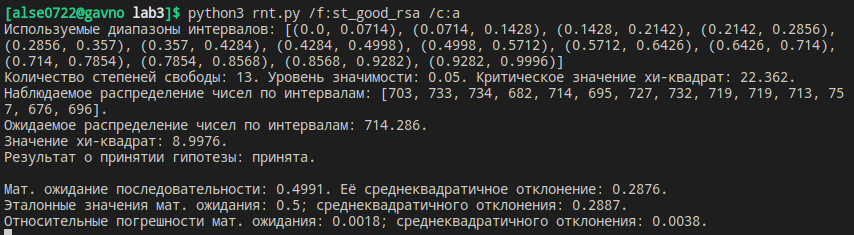


Рисунок 26 – Результаты выполнения программы для генератора rsa

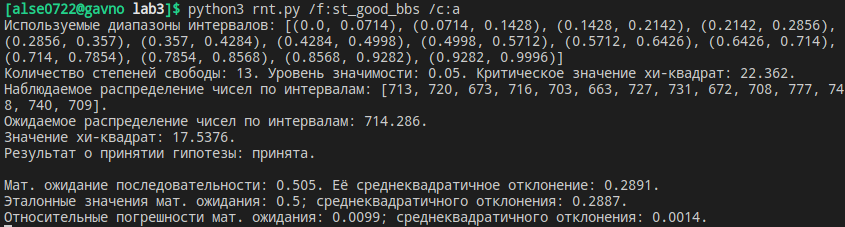


Рисунок 27 – Результаты выполнения программы для генератора bbs

## Критерий серий Описание критерия:

Критерий серий позволяет убедиться в том, что пары последовательных чисел равномерно распределены независимым образом. Проверка критерия про- водится следующим образом:

1. Воспользуемся критерием Хи-квадрат. Для этого преобразуем последова- тельность в с некоторым (в реализации ).

## Подсчитываем количество совпадений

## Применяем Хи-квадрат критерий к полученному набору с параметрами:

## Приведем результаты выполнения программы для каждого из генерато-

ров:

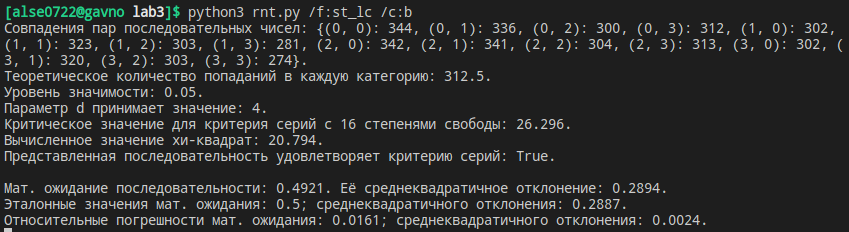


Рисунок 28 – Результаты выполнения программы для генератора lc

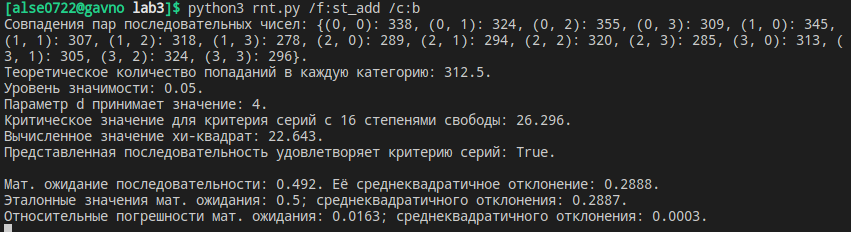


Рисунок 29 – Результаты выполнения программы для генератора add

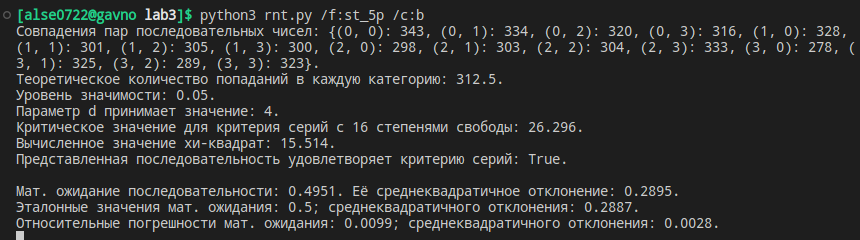


Рисунок 30 – Результаты выполнения программы для генератора 5p

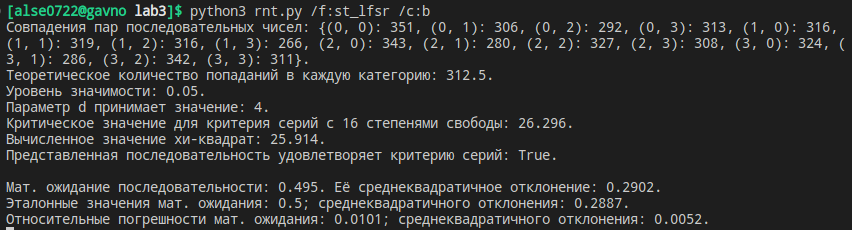


Рисунок 31 – Результаты выполнения программы для генератора lfsr

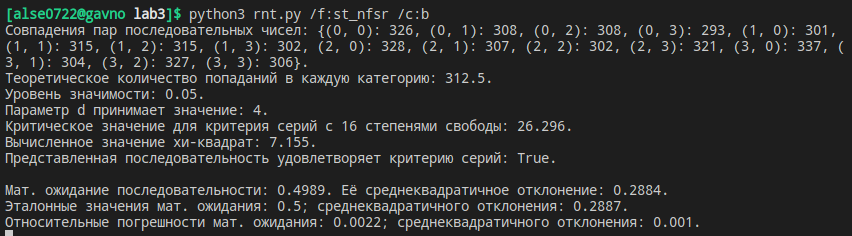


Рисунок 32 – Результаты выполнения программы для генератора nfsr

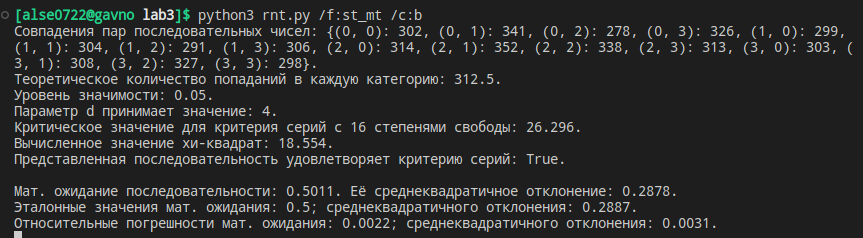


Рисунок 33 – Результаты выполнения программы для генератора mt

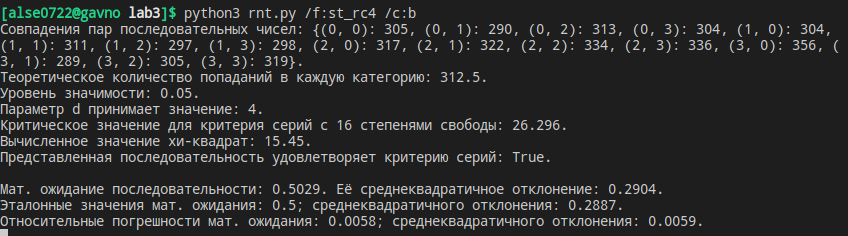


Рисунок 34 – Результаты выполнения программы для генератора rc4

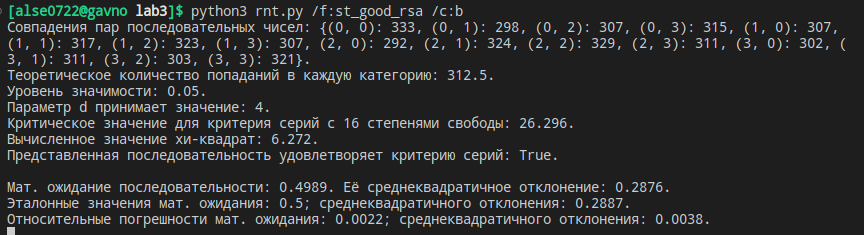


Рисунок 35 – Результаты выполнения программы для генератора rsa

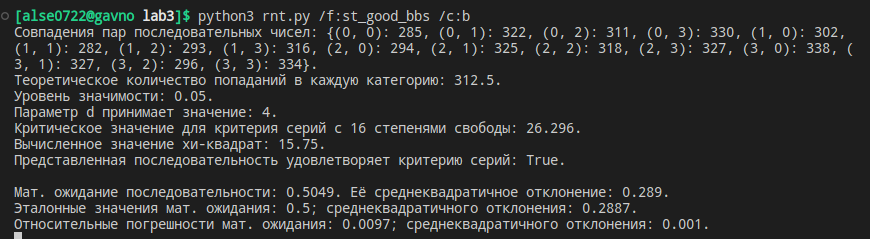


Рисунок 36 – Результаты выполнения программы для генератора bbs

## Критерий интервалов Описание критерия:

Пусть *a* и *b* – два действительных числа таких, что . Рассмот-

рим длины подпоследовательностей *xj, xj*+1*, . . . , xj*+*r*, в которых *xj, xj*+1*, . . . ,*

*xj*+*r*−1 ∈*/* [*a, b*]*, xj*+*r* ∈ [*a, b*]. Такую последовательность будем называть интервалом длины r.

Сначала, нам нужно подсчитать число интервалов длиной 0*,* 1*, . . . , n*. Шаги алгоритма подсчета числа интервалов:

## 1. Инициализация. Присвоить *j* = -1, *s* = 0, *cr* = 0, 0 ≤ *r* ≤ *t*.

2. *r* = 0.

1. *j* = *j* + 1. Если *a* ≤ *xj* ≤ *b*, то переход на шаг 5.

## *r* = *r* + 1. Переход к шагу 3.

1. Если *r* ≤ *t*, то *ct* = *ct* + 1, иначе – *cr* = *cr* + 1.
2. *s* = *s* + 1. Если *s < n* то переход на шаг 2.

## После этого мы можем применить хи-квадрат критерий для *k* = *t* + 1 к значениям *ci*, *i* = 0*,* 1*, . . . , t* с параметрами

*pr* = *p*(1 − *p*)*r* для 0 ≤ *r* ≤ *t* − 1; *pt* = (1 − *p*)*t*; *p* = (*a* − *b*).

Приведем результаты выполнения программы для каждого из генерато-

## ров:

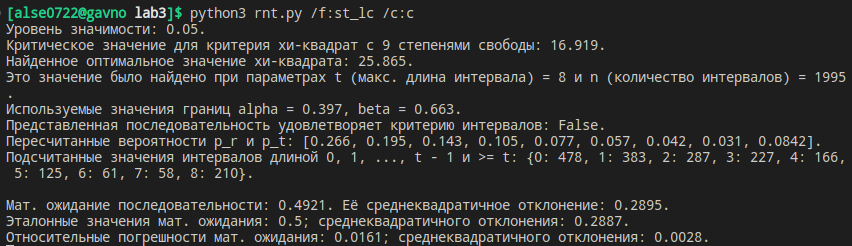


Рисунок 37 – Результаты выполнения программы для генератора lc

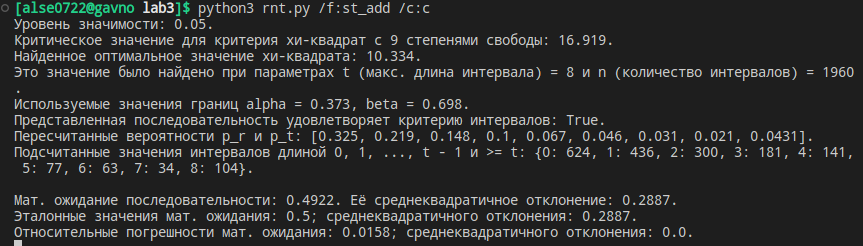


Рисунок 38 – Результаты выполнения программы для генератора add

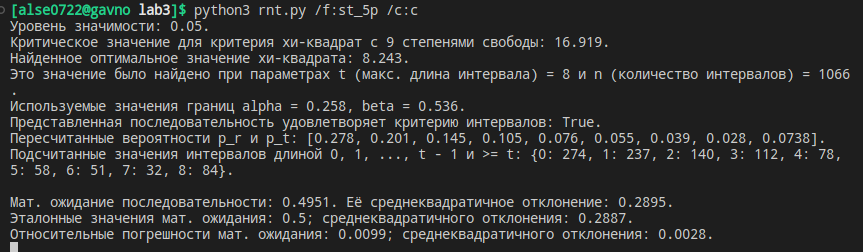


Рисунок 39 – Результаты выполнения программы для генератора 5p

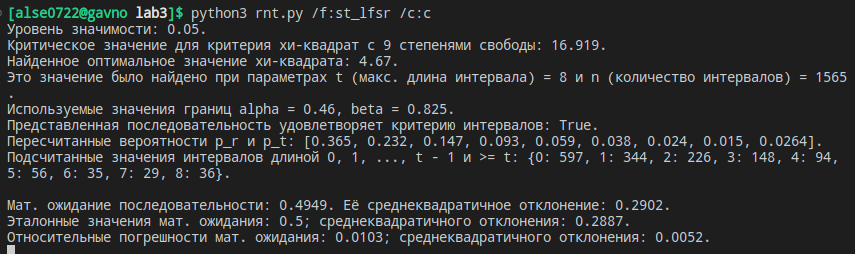


Рисунок 40 – Результаты выполнения программы для генератора lfsr

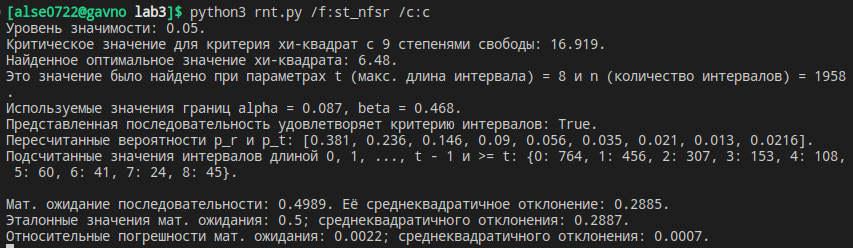


Рисунок 41 – Результаты выполнения программы для генератора nfsr

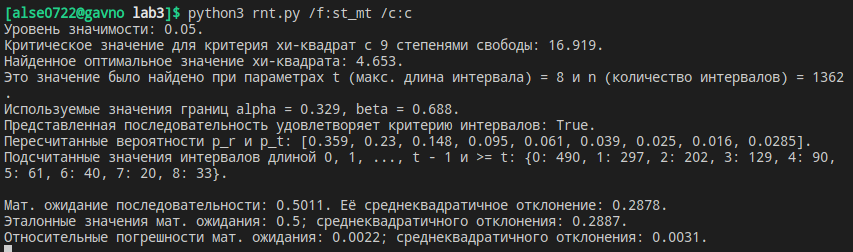


Рисунок 42 – Результаты выполнения программы для генератора mt

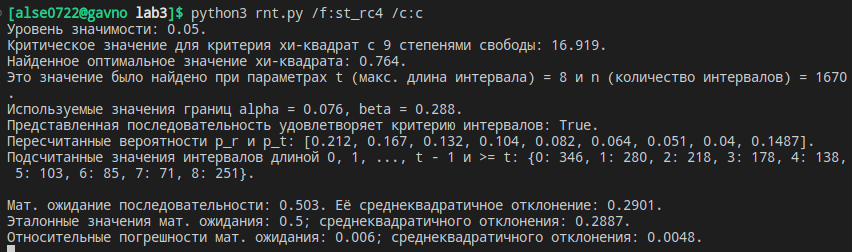


Рисунок 43 – Результаты выполнения программы для генератора rc4

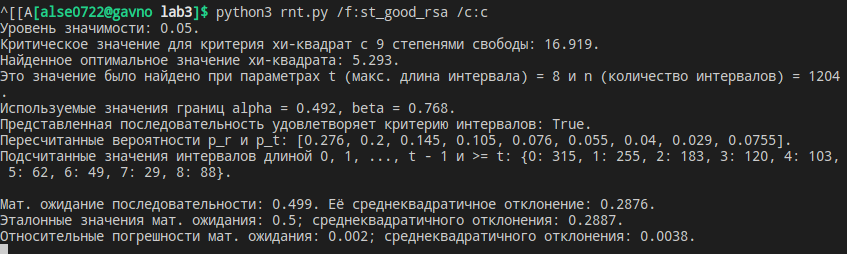


Рисунок 44 – Результаты выполнения программы для генератора rsa

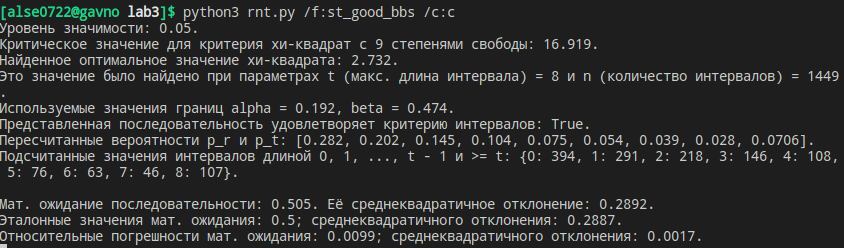


Рисунок 45 – Результаты выполнения программы для генератора bbs

## Критерий разбиений Описание критерия:

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

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

## Так как вероятности очень малы, когда *r* = 1 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 чисел, расположенных между и , имеют точно *r* различных элементов, и разделить это число на .

Приведем результаты выполнения программы для каждого из генерато-

## ров:

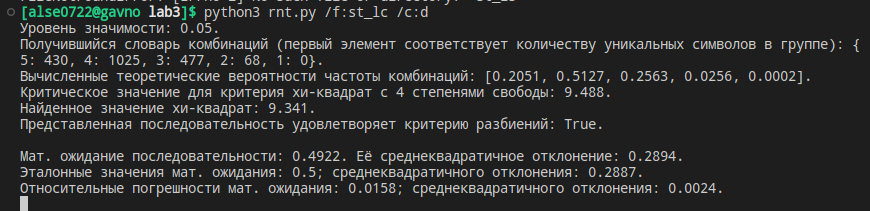


Рисунок 46 – Результаты выполнения программы для генератора lc

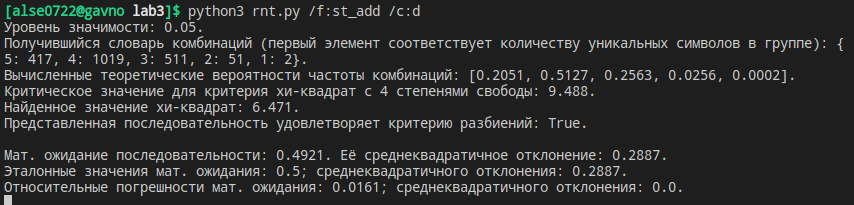


Рисунок 47 – Результаты выполнения программы для генератора add

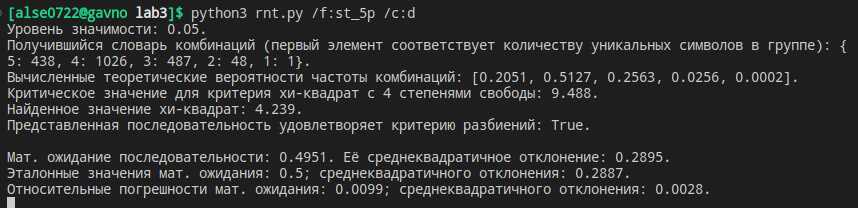


Рисунок 48 – Результаты выполнения программы для генератора 5p

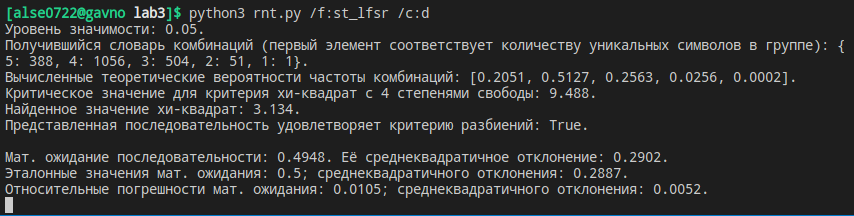


Рисунок 49 – Результаты выполнения программы для генератора lfsr

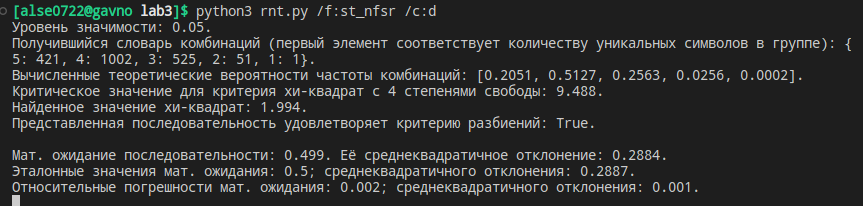


Рисунок 50 – Результаты выполнения программы для генератора nfsr

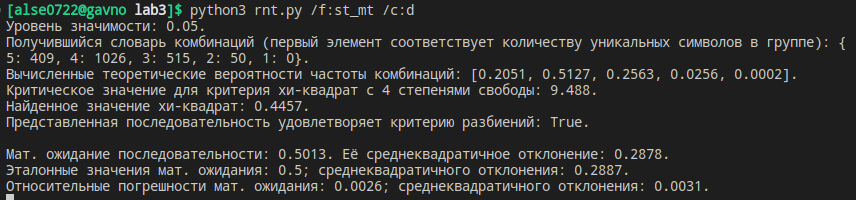


Рисунок 51 – Результаты выполнения программы для генератора mt

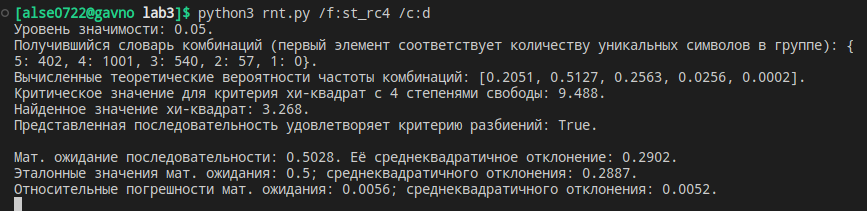


Рисунок 52 – Результаты выполнения программы для генератора rc4

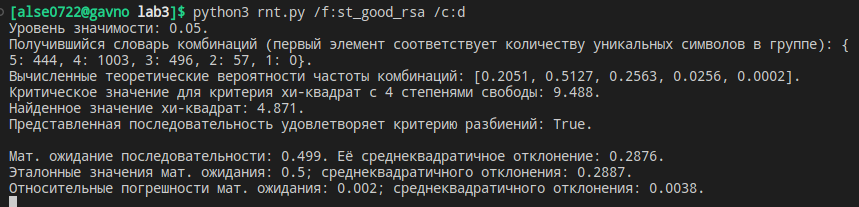


Рисунок 53 – Результаты выполнения программы для генератора rsa

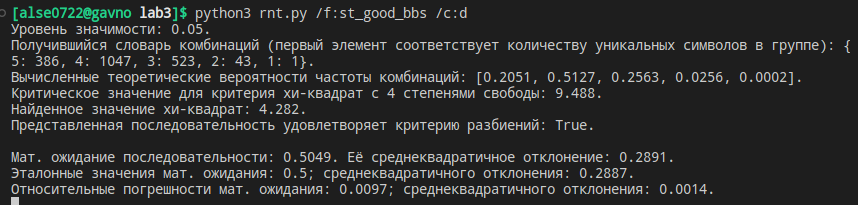


Рисунок 54 – Результаты выполнения программы для генератора bbs

## Критерий перестановок Описание критерия:

Последовательность разбивается на групп по

## элементов в каждой:

## Элементы в каждой группе можно упорядочивать различными способа- ми. Подсчитывается число групп с любым возможным порядком и применяется

хи-квадрат критерий с возможными категориями и вероятностью для

## каждой категории. В этом критерии предполагается, что не могут быть равны

между собой. В реализации берется .

## Приведем результаты выполнения программы для каждого из генерато-

ров:

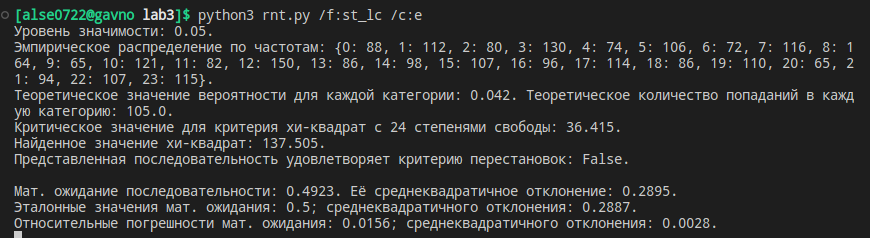


Рисунок 55 – Результаты выполнения программы для генератора lc

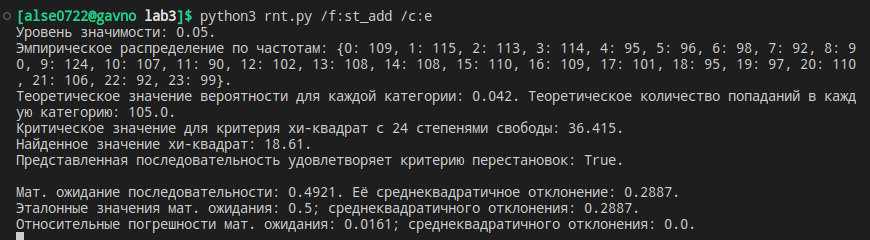


Рисунок 56 – Результаты выполнения программы для генератора add

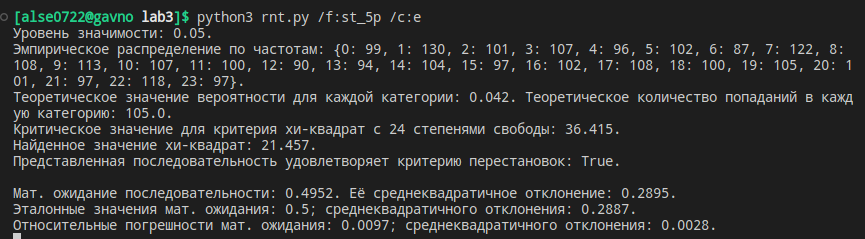


Рисунок 57 – Результаты выполнения программы для генератора 5p

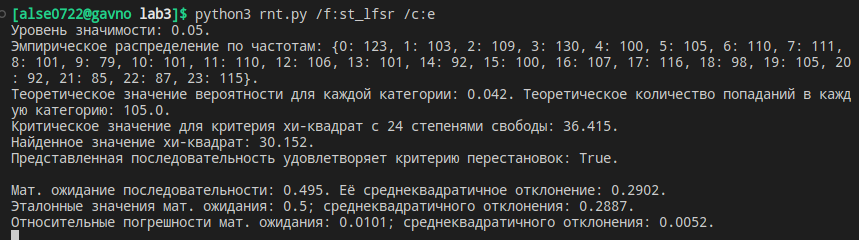


Рисунок 58 – Результаты выполнения программы для генератора lfsr

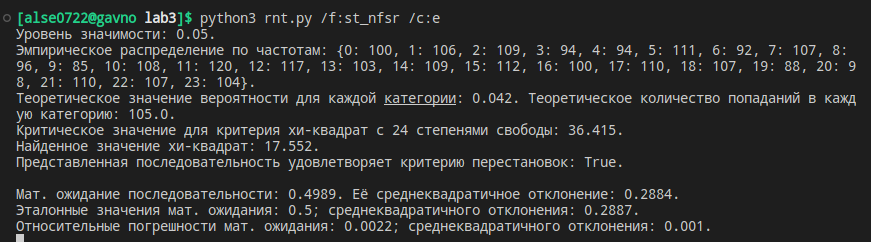


Рисунок 59 – Результаты выполнения программы для генератора nfsr

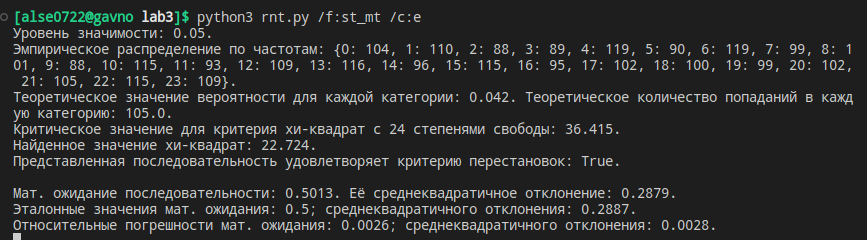


Рисунок 60 – Результаты выполнения программы для генератора mt

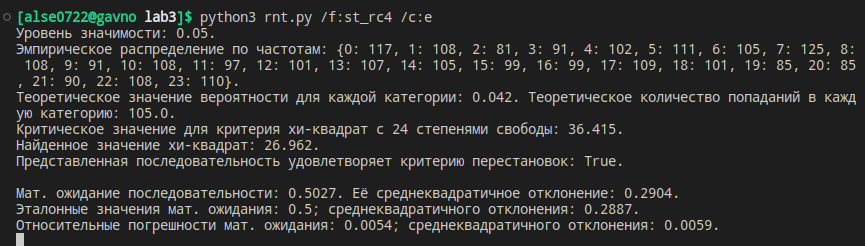


Рисунок 61 – Результаты выполнения программы для генератора rc4

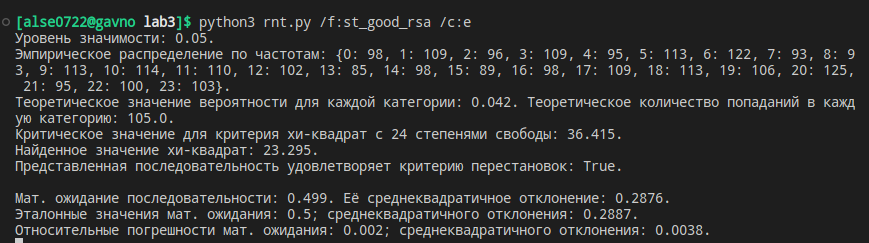


Рисунок 62 – Результаты выполнения программы для генератора rsa

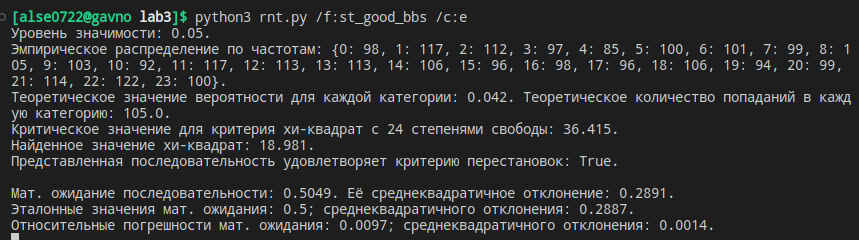


Рисунок 63 – Результаты выполнения программы для генератора bbs

## Критерий монотонности Описание критерия:

Последовательность можно проверить на предмет равномерности распре- деления монотонных серий чисел.

## Суть метода в том, чтобы проверить длины всех восходящих серий *ci* в последовательности и подсчитать для них статистику.

Для решения проблемы чередования длинных серий с короткими сериями можно сделать следующее:

## «Выбрасываем» элемент последовательности, который следует непосред- ственно за серией.

1. Если *xj* больше *xj*+1, то начнем следующую серию с *xj*+2.

## Мы получаем серии, длины которых независимы и, поэтому, можно ис- пользовать критерий хи-квадрат.

Приведем результаты выполнения программы для каждого из генерато-

## ров:

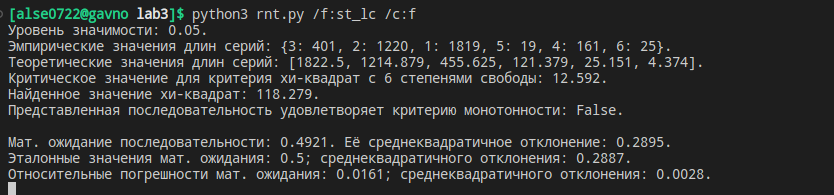


Рисунок 64 – Результаты выполнения программы для генератора lc

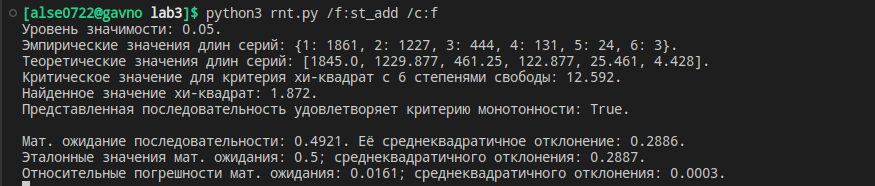


Рисунок 65 – Результаты выполнения программы для генератора add

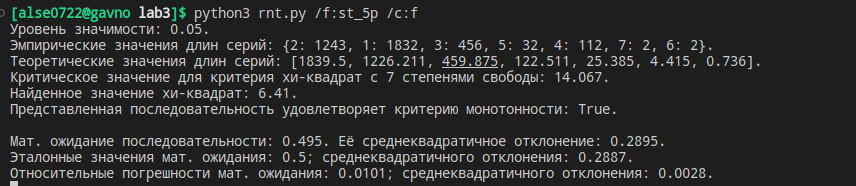


Рисунок 66 – Результаты выполнения программы для генератора 5p

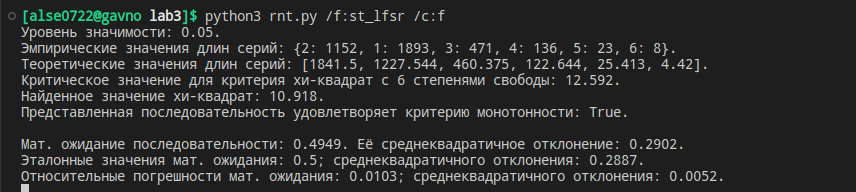


Рисунок 67 – Результаты выполнения программы для генератора lfsr

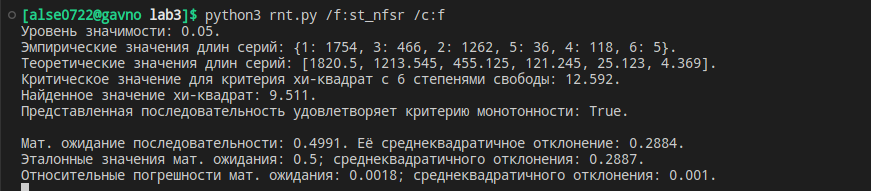


Рисунок 68 – Результаты выполнения программы для генератора nfsr

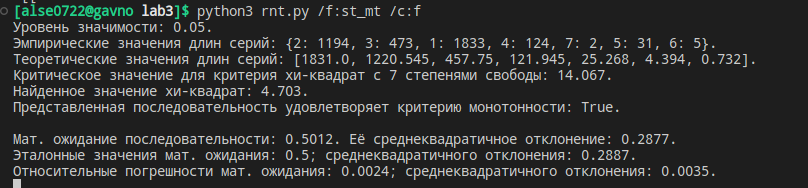


Рисунок 69 – Результаты выполнения программы для генератора mt

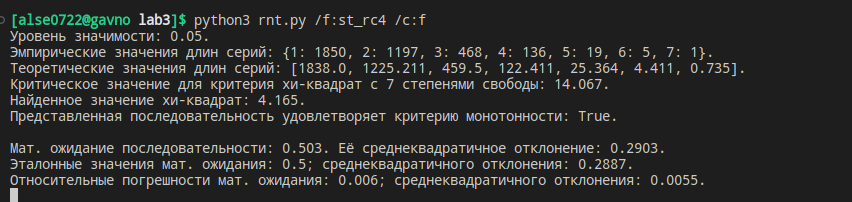


Рисунок 70 – Результаты выполнения программы для генератора rc4

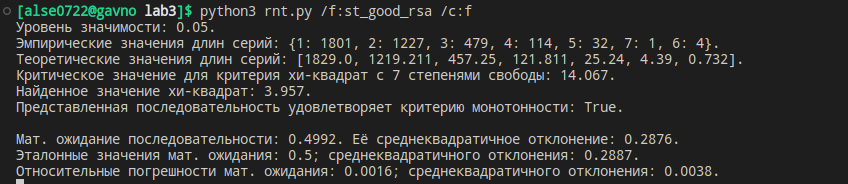


Рисунок 71 – Результаты выполнения программы для генератора rsa

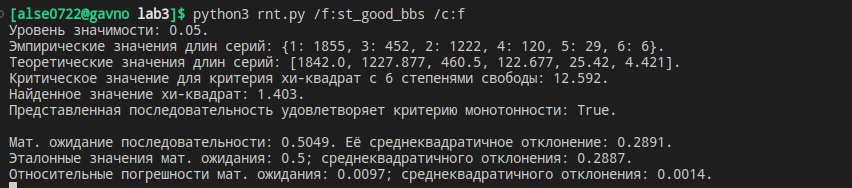


Рисунок 72 – Результаты выполнения программы для генератора bbs

## Критерий конфликтов Описание критерия:

Предположим, нам нужно оценить последовательность случайных чисел,

## в которой число величин в последовательности намного меньше числа катего- рий. В этом случае критерий хи-квадрат не применим, но можно использовать критерий конфликтов.

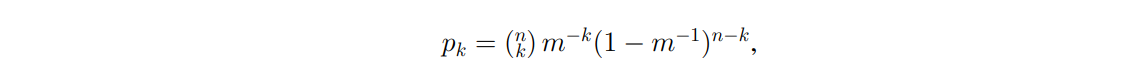
Предположим, что у нас m урн и n шаров, причем m значительно больше *n*. Если разместить шары в урнах наугад, то некоторые урны останутся пустыми, а в некоторых будет более одного шара. Когда в одну урну попадает больше

## одного шара, то говорят, что произошел «конфликт». Критерий конфликтов состоит в подсчете и оценке количества конфликтов.

Рассмотрим пример, когда *m* = 220, а *n* = 214. 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



## Приведем результаты выполнения программы для каждого из генерато-

ров:

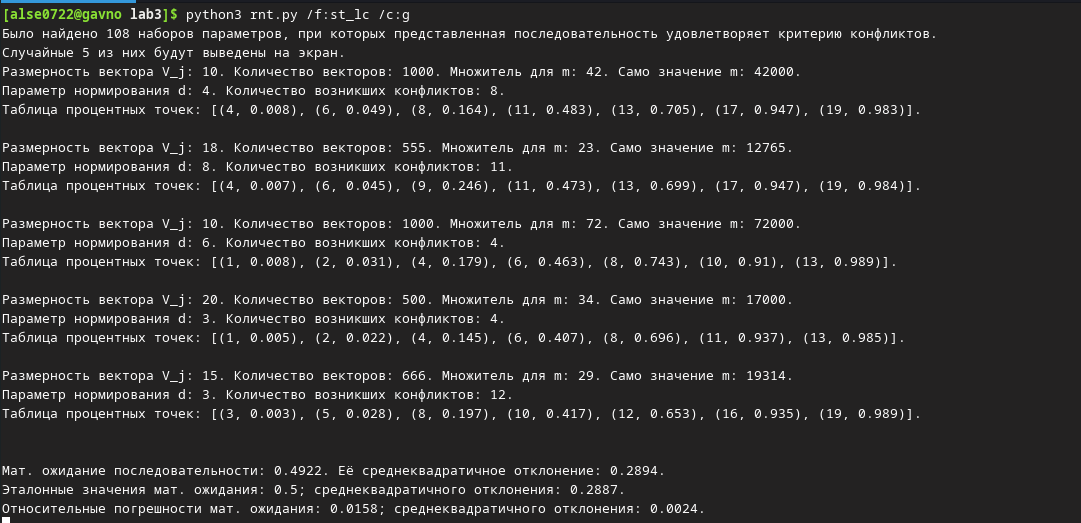


Рисунок 73 – Результаты выполнения программы для генератора lc

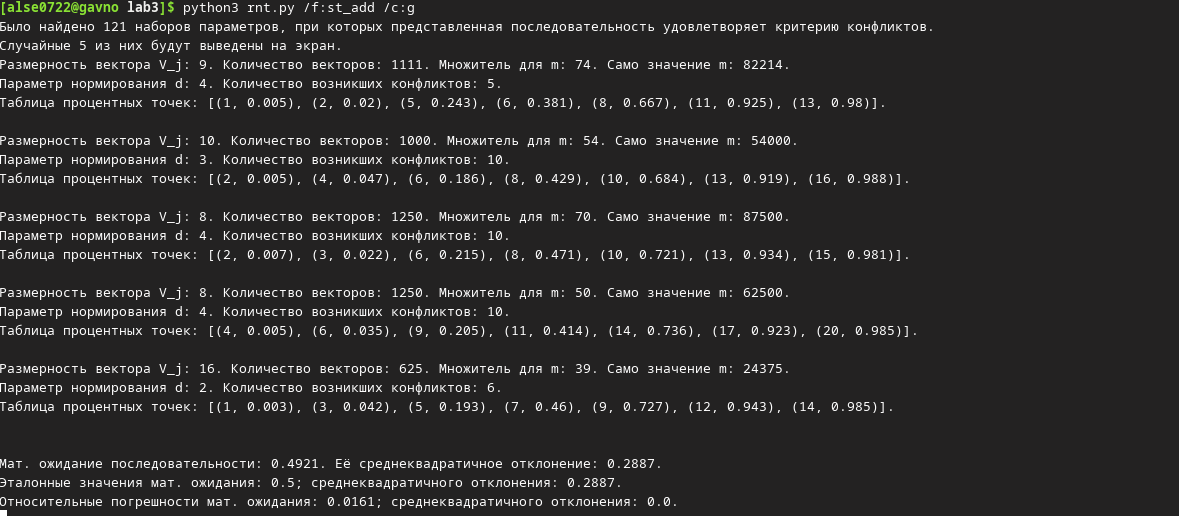


Рисунок 74 – Результаты выполнения программы для генератора add

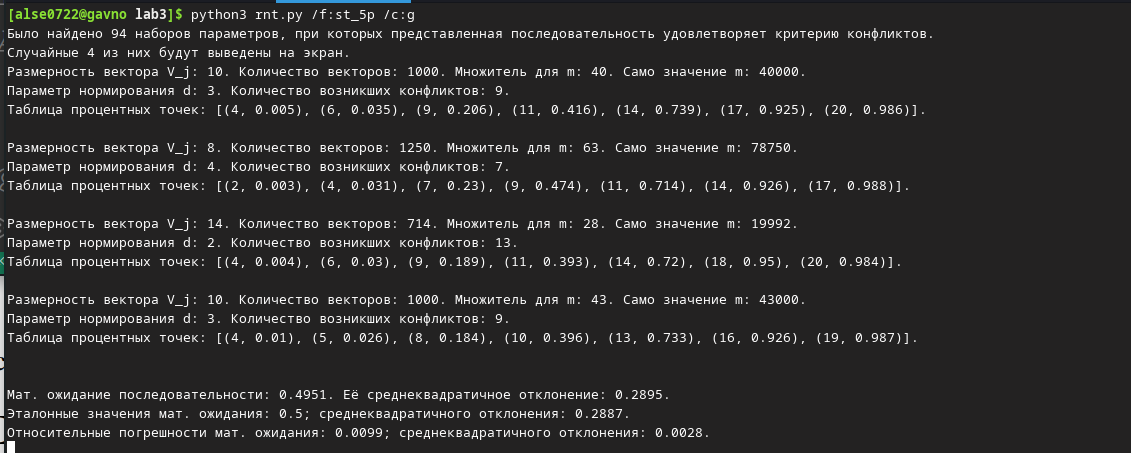


Рисунок 75 – Результаты выполнения программы для генератора 5p

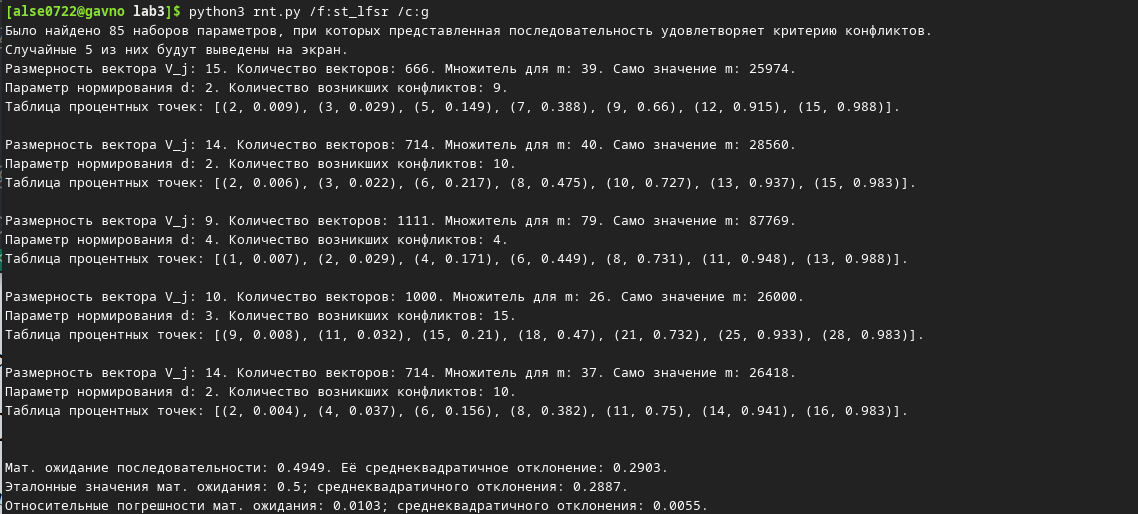


Рисунок 76 – Результаты выполнения программы для генератора lfsr

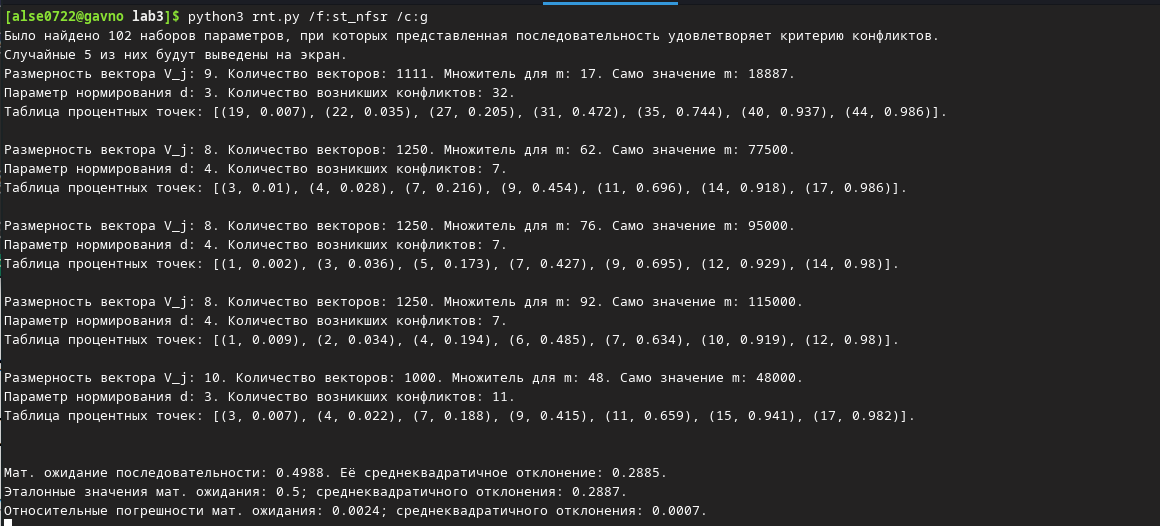


Рисунок 77 – Результаты выполнения программы для генератора nfsr

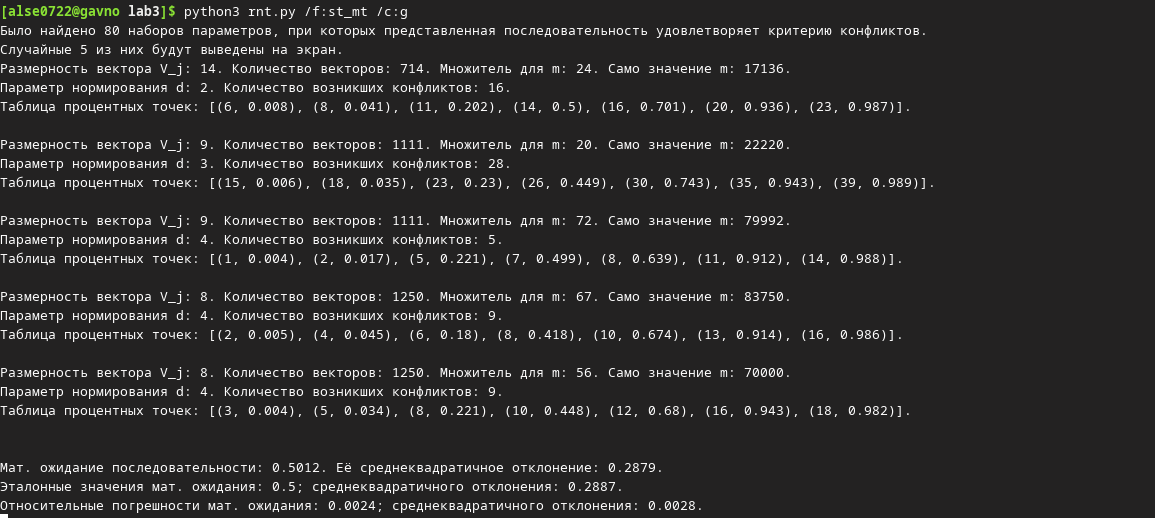


Рисунок 78 – Результаты выполнения программы для генератора mt

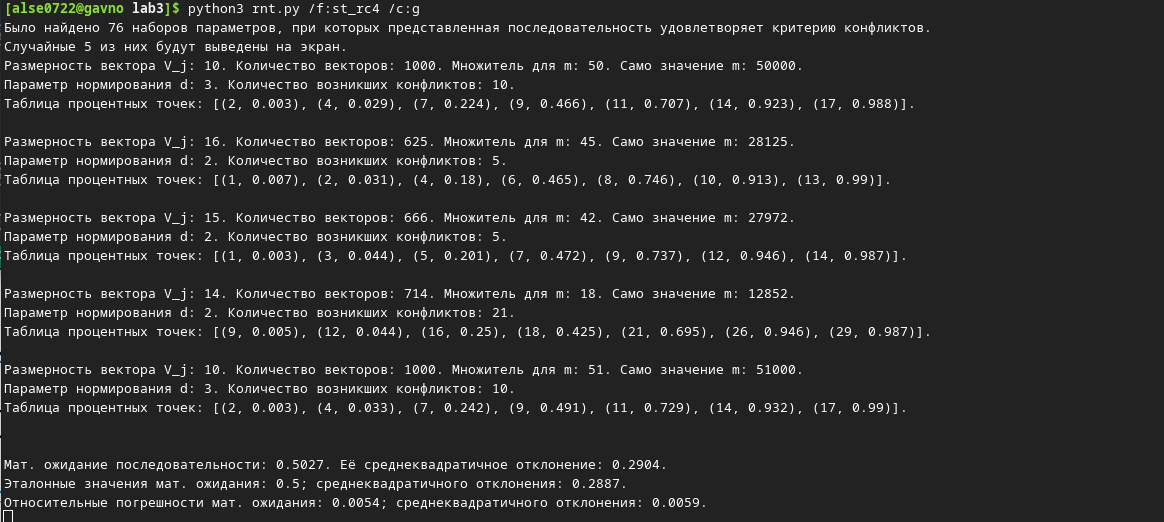


Рисунок 79 – Результаты выполнения программы для генератора rc4

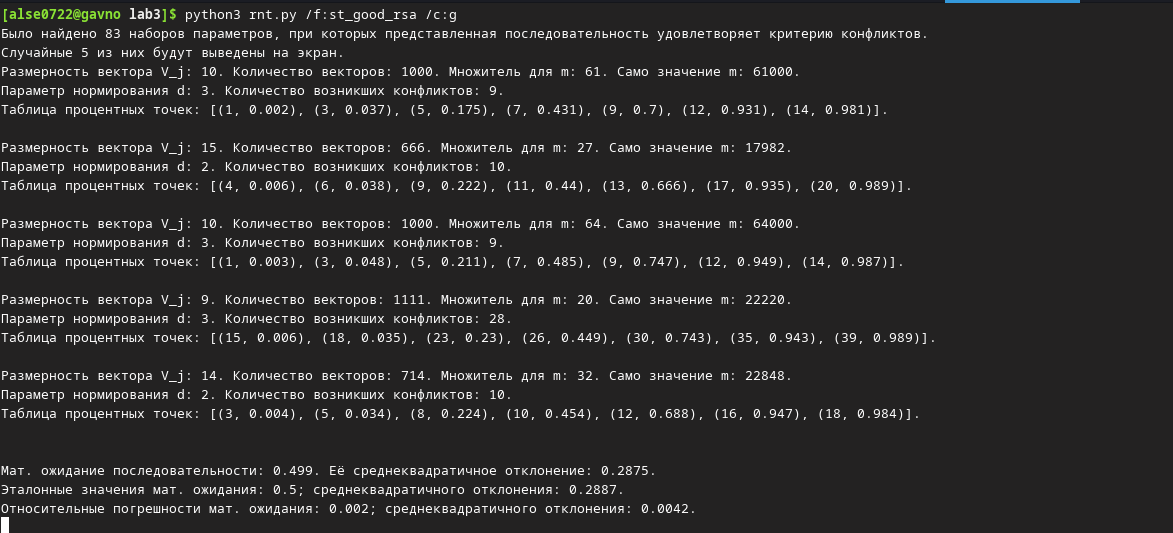


Рисунок 80 – Результаты выполнения программы для генератора rsa

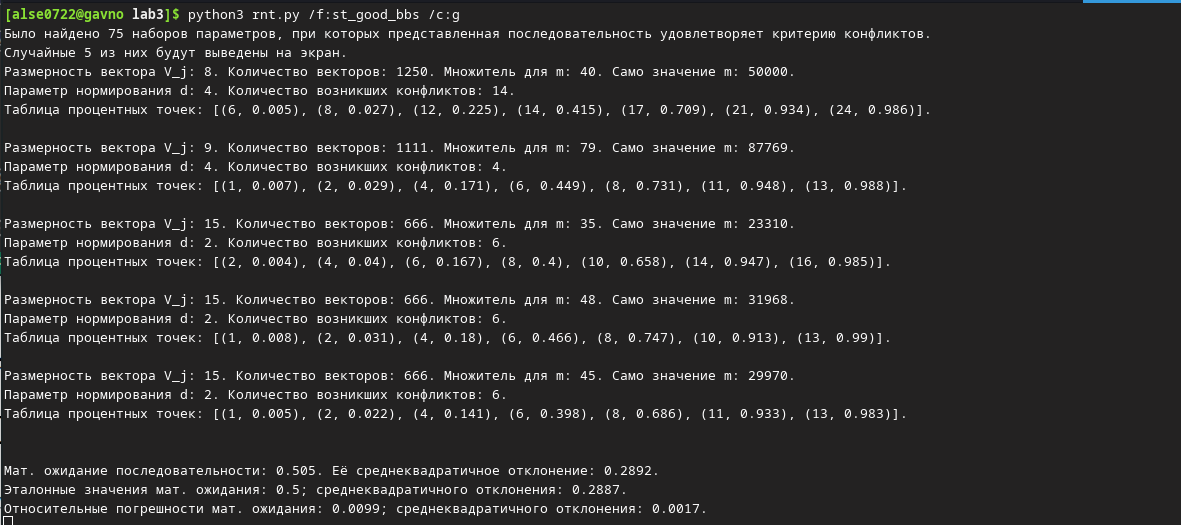


Рисунок 81 – Результаты выполнения программы для генератора bbs

## Результаты проверки всех критериев для последовательностей, сге- нерированных каждым генератором

В следующей таблице представлены результаты проверки для всех сгене- рированных последовательностей по каждому из рассмотренных выше крите- риев. Зеленый цвет - критерий пройден, красный - не пройден.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | *χ*2 | серий | интервал. | разбиений | перест. | монот. | конфл. |
| lc | 3.2297 | 3.353 | 13.313 | 2.037 | 119.287 | 46.48 | 148 |
| add | 24.7563 | 15.019 | 9.321 | 9.402 | 18.99 | 2.279 | 106 |
| 5p | 9.5378 | 65.929 | 12.985 | 3.94 | 131.638 | 3991940 | 378 |
| lfsr | 12.902 | 2097.11 | 30.225 | 52.143 | 481.057 | 371.465 | 211 |
| nfsr | 7.7395 | 3947.99 | 358.885 | 677.402 | 1584.28 | 43547900 | 333 |
| mt | 13.916 | 12.808 | 5.571 | 2.233 | 20.038 | 6.745 | 115 |
| rc4 | 822.272 | 135.25 | 30.929 | 4.392 | 26.81 | 10.65 | 130 |
| rsa | 164.639 | 57.526 | 29.857 | 3.496 | 11.238 | 18.575 | 87 |
| bbs | 17.7563 | 16.006 | 6.135 | 4.856 | 18.029 | 1.422 | 99 |

# ПРИЛОЖЕНИЕ А

**Исходный код программы проверки критериев**

**import** scipy

**import** sys

**import** math

**import** random

**import** numpy **as** np

**import** scipy.stats

**import** scipy.special

**from** sympy.functions.combinatorial.numbers **import** stirling

**import** matplotlib.pyplot **as** plt

sqrt\_two = math.sqrt(2)

crit\_unique\_values = 49

eps = 0.001

**def** **compute\_sd**(seq, l\_seq, expected=False):

**if** type(expected) == type(True):

expected = compute\_expected(seq, l\_seq)

**return** math.sqrt(sum([(seq[j] - expected) \*\* 2 **for** j **in** range(l\_seq)]) / l\_seq)

**def** **compute\_expected**(seq, l\_seq):

**return** sum(seq) / l\_seq

**def** **uniform**(seq, l\_seq, d=64, targeted=True):

seq = list(map(**lambda** elem : math.floor(elem \* d), seq))

alpha = .05

critical\_value = round(scipy.stats.chi2.ppf(1 - alpha, d - 1), 3)

est, chi = l\_seq / d, 0

**if** targeted:

print(f"Уровень значимости: {alpha}.")

print(f"Критическое значение для критерия равномерности с {d - 1} степенями свободы: {critical\_value}.")

**for** i **in** range(d):

chi += ((seq.count(i) - est) \*\* 2) / est

chi = round(chi, 3)

print(f"Вычисленное значение хи-квадрат: {chi}.")

print(f"Представленная последовательность удовлетворяет критерию равномерности: {chi < critical\_value}.")

**return** seq

**def** **series**(seq, l\_seq):

d, matches = 4, {}

d\_sq = d \* d

l\_unique\_values = len(np.unique(seq))

**for** i **in** range(d):

**for** j **in** range(d):

matches[(i, j)] = 0

seq = uniform(seq, l\_seq, d, targeted=False)

alpha, chi = .05, 0

critical\_value, est = round(scipy.stats.chi2.ppf(1 - alpha, d\_sq), 3), round(l\_seq / (2 \* d\_sq), 3)

**for** i **in** range(l\_seq // 2):

matches[(seq[2 \* i], seq[2 \* i + 1])] += 1

print(f"Совпадения пар последовательных чисел: {matches}.")

print(f"Теоретическое количество попаданий в каждую категорию: {est}.")

print(f"Уровень значимости: {alpha}.")

print(f"Параметр d принимает значение: {d}.")

print(f"Критическое значение для критерия серий с {d\_sq} степенями свободы: {critical\_value}.")

**for** key **in** matches.keys():

chi += ((matches[key] - est) \*\* 2) / est

chi = round(chi, 3)

print(f"Вычисленное значение хи-квадрат: {chi}.")

print(f"Представленная последовательность удовлетворяет критерию серий: {chi < critical\_value}.")

**if** l\_unique\_values < crit\_unique\_values:

print(f"Из-за маленького периода ({l\_unique\_values}) предложенная последовательность не может удовлетворять критерию серий.")

**def** **intervals**(seq, l\_seq):

t\_lower, t\_upper, n\_lower, n\_upper = 8, 12, 1000, 2000

\_alpha = .05

alpha, beta = random.uniform(0, 1), random.uniform(0, 1)

**while** **not** (0.2 < abs(alpha - beta) < 0.4):

alpha, beta = random.uniform(0, 1), random.uniform(0, 1)

**if** beta < alpha: alpha, beta = beta, alpha

alpha, beta = round(alpha, 3), round(beta, 3)

p = beta - alpha

s, count, chi, best\_crit = 0, {}, math.inf, math.inf

best\_len, best\_ints = math.inf, math. inf

best\_probs, best\_count = list(), list()

**for** t **in** range(t\_lower, t\_upper):

**for** n **in** range(n\_lower, n\_upper):

s, count = 0, {}

**for** k **in** range(t + 1): count[k] = 0

r = 0

**for** j **in** range(l\_seq):

**if** alpha <= seq[j] < beta:

**if** r >= t: count[t] += 1

**else**: count[r] += 1

s += 1

**if** s < n: r = 0

**else**: **break**

**else**: r += 1

probs, chi\_acc = [round(p \* ((1 - p) \*\* \_r), 3) **for** \_r **in** range(t)], 0

probs.append(round((1 - p) \*\* t, 4))

critical\_value = round(scipy.stats.chi2.ppf(1 - \_alpha, t + 1), 3)

**for** j **in** range(len(probs)):

est = n \* probs[j]

**if** est == 0: **break**

chi\_acc += ((count[j] - est) \*\* 2) / est

**if** chi\_acc < chi:

chi, best\_crit = chi\_acc, critical\_value

best\_len, best\_ints = t, n

best\_probs, best\_count = probs, count

print(f"Уровень значимости: {\_alpha}.")

print(f"Критическое значение для критерия хи-квадрат с {best\_len + 1} степенями свободы: {best\_crit}.")

print(f"Найденное оптимальное значение хи-квадрата: {round(chi, 3)}.")

print(f"Это значение было найдено при параметрах t (макс. длина интервала) = {best\_len} и n (количество интервалов) = {best\_ints}.")

print(f"Используемые значения границ alpha = {alpha}, beta = {beta}.")

print(f"Представленная последовательность удовлетворяет критерию интервалов: {chi < best\_crit}.")

print(f"Пересчитанные вероятности p\_r и p\_t: {best\_probs}.")

print(f"Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t: {best\_count}.")

**def** **partitions**(seq, l\_seq, d=8, k=5):

seq = uniform(seq, l\_seq, d, targeted=False)

groups\_quan, alpha, chi, num\_groups = {}, .05, 0, l\_seq // k

critical\_value = round(scipy.stats.chi2.ppf(1 - alpha, k - 1), 3)

**for** i **in** range(k, 0, -1): groups\_quan[i] = 0

counter = 0

**for** i **in** range(num\_groups):

current\_group = seq[(i \* k):((i + 1) \* k)]

groups\_quan[len(list(np.unique(current\_group)))] += 1

probs = [round((math.prod([i **for** i **in** range(d, d - r, -1)]) / (d \*\* k)) \* stirling(k, r, kind=2), 4) **for** r **in** groups\_quan.keys()]

chi = sum([((value - probs[i] \* num\_groups) \*\* 2) / (probs[i] \* num\_groups) **for** i, value **in** enumerate(groups\_quan.values())])

print(f"Уровень значимости: {alpha}.")

print(f"Получившийся словарь комбинаций (первый элемент соответствует количеству уникальных символов в группе): {groups\_quan}.")

print(f"Вычисленные теоретические вероятности частоты комбинаций: {probs}.")

print(f"Критическое значение для критерия хи-квадрат с {k - 1} степенями свободы: {critical\_value}.")

print(f"Найденное значение хи-квадрат: {chi}.")

print(f"Представленная последовательность удовлетворяет критерию разбиений: {chi < critical\_value}.")

**def** **permutations**(seq, l\_seq, t=4):

df = math.prod([j **for** j **in** range(1, t + 1)])

categ\_quan = {i:0 **for** i **in** range(df)}

alpha, num\_groups = .05, l\_seq // t

critical\_value = round(scipy.stats.chi2.ppf(1 - alpha, df), 3)

**def** **perm\_analysis**(perm):

r, f = t, 0

**while** r > 1:

s = perm.index(max(perm)) + 1

f = r \* f + s - 1

perm[r - 1], perm[s - 1] = perm[s - 1], perm[r - 1]

perm, r = perm[:-1], r - 1

**return** f

**for** i **in** range(num\_groups):

current\_group = seq[(i \* t):((i + 1) \* t)]

**if** len(np.unique(current\_group)) != t: **continue**

**else**: categ\_quan[perm\_analysis(current\_group)] += 1

est, chi = round(1 / df, 3), 0

est\_cat = round(est \* num\_groups, 3)

print(f"Уровень значимости: {alpha}.")

print(f"Эмпирическое распределение по частотам: {categ\_quan}.")

print(f"Теоретическое значение вероятности для каждой категории: {est}. Теоретическое количество попаданий в каждую категорию: {est\_cat}.")

print(f"Критическое значение для критерия хи-квадрат с {df} степенями свободы: {critical\_value}.")

chi = round(sum([((categ\_quan[i] - est\_cat) \*\* 2) / est\_cat **for** i **in** range(df)]), 3)

print(f"Найденное значение хи-квадрат: {chi}.")

print(f"Представленная последовательность удовлетворяет критерию перестановок: {chi < critical\_value}.")

**def** **monotonous**(seq, l\_seq):

categ\_quan, est = {}, list()

alpha, chi, i = .05, 0, 0

current\_spree = 1

**while** i < l\_seq - 1:

**if** seq[i] < seq[i + 1]: current\_spree += 1

**else**:

**if** current\_spree **not** **in** categ\_quan:

categ\_quan[current\_spree] = 1

**else**: categ\_quan[current\_spree] += 1

current\_spree, i = 1, i + 1

i += 1

sum\_series = sum([value **for** value **in** categ\_quan.values()])

num\_categs = len(categ\_quan)

critical\_value = round(scipy.stats.chi2.ppf(1 - alpha, num\_categs), 3)

fact\_acc = 1

**for** i **in** range(1, num\_categs + 1):

fact\_acc \*= i

est.append(round(1 / fact\_acc - 1 / (fact\_acc \* (i + 1)), 4))

est = [round(sum\_series \* est[j], 3) **for** j **in** range(num\_categs)]

chi = round(sum([((categ\_quan[j + 1] - est[j]) \*\* 2) / est[j] **for** j **in** range(num\_categs)]), 3)

print(f"Уровень значимости: {alpha}.")

print(f"Эмпирические значения длин серий: {categ\_quan}.")

print(f"Теоретические значения длин серий: {est}.")

print(f"Критическое значение для критерия хи-квадрат с {num\_categs} степенями свободы: {critical\_value}.")

print(f"Найденное значение хи-квадрат: {chi}.")

print(f"Представленная последовательность удовлетворяет критерию монотонности: {chi < critical\_value}.")

**def** **conflicts**(seq, l\_seq):

s\_lower, s\_upper, d\_lower, d\_upper, n\_to\_print = 8, 20, 2, 8, 5

m\_lower, m\_upper = 16, 128

eps = 1e-20

T\_table = (.01, .05, .25, .50, .75, .95, .99, 1.)

**def** **percent\_points**(m, n):

A, conflicts\_et\_probs = [0] \* (n + 1), list()

A[1] = j\_0 = j\_1 = 1

**for** i **in** range(n - 1):

j\_1 += 1

**for** j **in** range(j\_1, j\_0 - 1, -1):

j\_by\_m = j / m

A[j] = j\_by\_m \* A[j] + (1 + 1 / m - j\_by\_m) \* A[j - 1]

**if** A[j] < eps:

A[j] = 0

**if** j == j\_1: j\_1 -= 1; **continue**

**if** j == j\_0: j\_0 += 1

p, t, j = 0, 0, j\_0 - 1

**while** t != len(T\_table) - 1:

**while** p <= T\_table[t]:

j += 1

p += A[j]

conflicts\_et\_probs.append((n - j - 1, round(1 - p, 3)))

t += 1

**return** conflicts\_et\_probs

suitables = list()

**for** vec\_size **in** range(s\_lower, s\_upper + 1):

n\_param = l\_seq // vec\_size

**for** d\_param **in** range(d\_lower, d\_upper + 1):

seq\_normed = uniform(seq, l\_seq, d=d\_param, targeted=False)

words, n\_conflicts = list(), 0

**for** j\_index **in** range(n\_param):

\_slice = seq\_normed[(j\_index \* vec\_size):((j\_index + 1) \* vec\_size)]

**if** \_slice **not** **in** words: words.append(\_slice)

**else**: n\_conflicts += 1

**for** m\_param **in** range(m\_lower, m\_upper + 1):

m\_value = n\_param \* m\_param

confs\_et\_probs = percent\_points(m\_value, n\_param)[::-1]

**if** n\_conflicts == 0 **or** confs\_et\_probs[0][0] == -1 **or** confs\_et\_probs[0][0] == 0:

**continue**

**if** confs\_et\_probs[2][0] <= n\_conflicts <= confs\_et\_probs[-3][0]:

suitables.append((confs\_et\_probs, n\_conflicts, vec\_size, n\_param, d\_param, m\_param, m\_value))

l\_suits, rand\_idxs = len(suitables), list()

**for** j **in** range(n\_to\_print):

rand\_idx = random.randint(0, l\_suits - 1)

**if** rand\_idx **not** **in** rand\_idxs: rand\_idxs.append(rand\_idx)

**else**:

**while** rand\_idx **in** rand\_idxs: rand\_idx = random.randint(0, l\_suits - 1)

print(f"Было найдено {l\_suits} наборов параметров, при которых представленная последовательность удовлетворяет критерию конфликтов.")

print(f"Случайные {len(rand\_idxs)} из них будут выведены на экран.")

**for** j **in** range(len(rand\_idxs)):

confs\_et\_probs, n\_conflicts, vec\_size, n\_param, d\_param, m\_param, m\_value = suitables[rand\_idxs[j] % l\_suits]

print(f"Размерность вектора V\_j: {vec\_size}. Количество векторов: {n\_param}. Множитель для m: {m\_param}. Само значение m: {m\_value}.")

print(f"Параметр нормирования d: {d\_param}. Количество возникших конфликтов: {n\_conflicts}.")

print(f"Таблица процентных точек: {confs\_et\_probs}.", end='\n\n')

**def** **chi\_wrapper**(seq, l\_seq, intervals):

print(f"Используемые диапазоны интервалов: {[(round(interval[0], 4), round(interval[1], 4)) **for** interval **in** intervals]}")

ints\_len = len(intervals)

**def** **printer**(est, chi\_val, chi\_crit):

print(f"Ожидаемое распределение чисел по интервалам: {est}.")

print(f"Значение хи-квадрат: {chi\_val}.")

print(f"Результат о принятии гипотезы: {'принята' **if** 0 < chi\_val < chi\_crit **else** 'не принята'}.", end='\n')

actual = [0 **for** i **in** range(ints\_len)]

**for** i **in** range(l\_seq):

**for** j **in** range(ints\_len):

**if** intervals[j][0] <= seq[i] < intervals[j][1]:

actual[j] += 1; **break**

est = round(l\_seq / ints\_len, 3)

chi\_st = round(sum([((actual[j] - est) \*\* 2) / est **for** j **in** range(ints\_len)]), 4)

chi\_crit = round(scipy.stats.chi2.ppf(1-.05, df=(ints\_len - 1)), 4)

print(f"Количество степеней свободы: {ints\_len - 1}. Уровень значимости: {0.05}.", end= ' ')

print(f"Критическое значение хи-квадрат: {chi\_crit}.")

print(f"Наблюдаемое распределение чисел по интервалам: {actual}.", end='\n')

printer(est, chi\_st, chi\_crit)

**def** **draw\_params**(seq, l\_seq):

steps = (50, 100, 200, 500)

exp\_vals\_stepped, \_sd\_vals\_stepped = list(), list()

**for** step **in** steps:

exp\_acc, exp\_vals = 0, list()

sd\_vals = list()

**for** j **in** range(l\_seq // step):

idx\_l, idx\_r = j \* step, (j + 1) \* step

\_slice\_exp = compute\_expected(seq[idx\_l:idx\_r], idx\_r)

exp\_acc = exp\_acc \* idx\_l / idx\_r + \_slice\_exp

exp\_vals.append(exp\_acc)

sd\_acc = compute\_sd(seq[:idx\_r], idx\_r, exp\_acc)

sd\_vals.append(sd\_acc)

exp\_vals\_stepped.append(exp\_vals)

\_sd\_vals\_stepped.append(sd\_vals)

exp\_fig, exp\_axs = plt.subplots(nrows=2, ncols=2)

exp\_fig.suptitle('Сходимость мат. ожидания для различных значений шага')

\_sd\_fig, \_sd\_axs = plt.subplots(nrows=2, ncols=2)

\_sd\_fig.suptitle('Сходимость среднеквадратичного отклонения для различных значений шага')

**for** j **in** range(len(steps)):

j\_bin = bin(j)[2:]

l\_j\_bin = len(j\_bin)

stepped = [steps[j] \* (k + 1) **for** k **in** range(l\_seq // steps[j])]

idx\_0 = 0 **if** l\_j\_bin == 1 **else** int(j\_bin[0])

idx\_1 = int(j\_bin[0]) **if** l\_j\_bin == 1 **else** int(j\_bin[1])

exp\_axs[idx\_0, idx\_1].plot(stepped, exp\_vals\_stepped[j])

exp\_axs[idx\_0, idx\_1].set\_title(f"Шаг: {steps[j]}")

\_sd\_axs[idx\_0, idx\_1].plot(stepped, \_sd\_vals\_stepped[j])

\_sd\_axs[idx\_0, idx\_1].set\_title(f"Шаг: {steps[j]}")

plt.show()

**def** **main**():

file = sys.argv[1][3:]

criterion = sys.argv[2][3:]

f = open(file, 'r')

seq = list(map(float, (f.read()).split(',')))

l\_seq = len(seq)

unique\_values = np.unique(seq)

**if** len(unique\_values) > crit\_unique\_values:

seq = [seq[i] **if** seq[i] != 1.0 **else** round(np.random.uniform(), 3) **for** i **in** range(l\_seq)]

l\_bord, r\_bord = min(seq), max(seq)

num\_intervals = math.ceil(1 + 1.4 \* math.log(l\_seq))

step = round((r\_bord - l\_bord) / num\_intervals, 4)

\_intervals = [(l\_bord + i \* step, l\_bord + (i + 1) \* step) **for** i **in** range(num\_intervals)]

**else**:

seq = [seq[i] **if** seq[i] != 1.0 **else** seq[i] - eps **for** i **in** range(l\_seq)]

unique\_values = np.unique(seq)

\_intervals = [(value - eps, value + eps) **for** value **in** unique\_values]

**if** criterion=='a':

chi\_wrapper(seq, l\_seq, \_intervals)

**if** criterion=='b':

series(seq, l\_seq)

**if** criterion=='c':

intervals(seq, l\_seq)

**if** criterion=='d':

partitions(seq, l\_seq)

**if** criterion=='e':

permutations(seq, l\_seq)

**if** criterion=='f':

monotonous(seq, l\_seq)

**if** criterion=='g':

conflicts(seq, l\_seq)

exp, sd = round(compute\_expected(seq, l\_seq), 4), round(compute\_sd(seq, l\_seq), 4)

exp\_th, sd\_th = .5, round(math.sqrt(1 / 12), 4)

print()

print(f"Мат. ожидание последовательности: {exp}. Её среднеквадратичное отклонение: {sd}.")

print(f"Эталонные значения мат. ожидания: {exp\_th}; среднеквадратичного отклонения: {sd\_th}.")

rel\_err\_exp, rel\_err\_sd = round(abs(exp - exp\_th) / exp, 4), round(abs(sd - sd\_th) / sd, 4)

print(f"Относительные погрешности мат. ожидания: {rel\_err\_exp}; среднеквадратичного отклонения: {rel\_err\_sd}.")

waiter = input()

draw\_params(seq, l\_seq)

**if** \_\_name\_\_ == '\_\_main\_\_':

main()