

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

Яхина Шамиля Илдусовича

Преподаватель

Ассистент

Н.А. Артемова

подпись, дата

Саратов 2023

1 Тестирование статистических свойств последовательности псевдослучайных чисел

Описание задания:

Протестируйте 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
 - a) 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 величин, сгенерированные в первой части практической работы следующими алгоритмами:

1. Линейный конгруэнтный метод;
2. Аддитивный метод;
3. Пятипараметрический метод;
4. Регистр сдвига с обратной связью (РСЛОС);
5. Нелинейная комбинация РСЛОС;
6. Вихрь Мерсенна;
7. RC4;
8. ГПСЧ на основе RSA;
9. 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 $\in [0, 0.999]$.

1.1 Мат. ожидание, среднекв. отклонение, сравнение с теоретическими оценками и построение графиков

Для того, чтобы найти **математическое ожидание** необходимо воспользоваться следующей формулой:

$$M = \frac{\sum_{i=1}^k x_i}{k},$$

где x_i - числа последовательности, k - количество таких чисел.

Для того, чтобы найти **среднеквадратичное отклонение** необходимо воспользоваться следующей формулой:

$$\sigma = \sqrt{\frac{\sum_{i=1}^k |x_i - M|^2}{k}},$$

где x_i - числа последовательности, k - количество таких чисел, M - 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 выборки. Для этого необходимо сравнить полученные результаты с эталонными и вычислить погрешности согласно следующим формулам:

$$\Delta M = \frac{|M - 0.5|}{M}, \Delta Q = \frac{|Q - \sqrt{\frac{1}{12}}|}{M},$$

где M – мат. ожидание, а Q – среднекв. отклонение.

Посчитаем данные показатели для рассматриваемых последовательностей:

Генератор	Мат. ожидание	Среднекв. отклонение	Погрешность	
			Мат. ожидания	Среднекв. отклонения
lc	0.497	0.287	0.006	0.005
add	0.496	0.287	0.009	0.007
5p	0.499	0.289	0.002	0.001
lfsr	0.496	0.286	0.008	0.008
nfsr	0.503	0.288	0.007	0.002
mt	0.501	0.287	0.002	0.005
rc4	0.508	0.284	0.015	0.017
rsa	0.508	0.288	0.015	0.002
bbs	0.505	0.289	0.009	0.002

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutputLc.dat
Мат. ожидание: 0.497
Среднекв. отклонение: 0.287
Погрешность мат. ожидания: 0.006
Погрешность среднекв. отклонения: 0.005
```

Рисунок 1 – Вычисление показателей для генератора lc

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutputAdd.dat
Мат. ожидание: 0.496
Среднекв. отклонение: 0.287
Погрешность мат. ожидания: 0.009
Погрешность среднекв. отклонения: 0.007
```

Рисунок 2 – Вычисление показателей для генератора add

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutput5p.dat
Мат. ожидание: 0.499
Среднекв. отклонение: 0.289
Погрешность мат. ожидания: 0.002
Погрешность среднекв. отклонения: 0.001
```

Рисунок 3 – Вычисление показателей для генератора 5p

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutputLfsr.dat
Мат. ожидание: 0.496
Среднекв. отклонение: 0.286
Погрешность мат. ожидания: 0.008
Погрешность среднекв. отклонения: 0.008
```

Рисунок 4 – Вычисление показателей для генератора lfsr

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutputNfsr.dat
Мат. ожидание: 0.503
Среднекв. отклонение: 0.288
Погрешность мат. ожидания: 0.007
Погрешность среднекв. отклонения: 0.002
```

Рисунок 5 – Вычисление показателей для генератора nfsr

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutputMt.dat
Мат. ожидание: 0.501
Среднекв. отклонение: 0.287
Погрешность мат. ожидания: 0.002
Погрешность среднекв. отклонения: 0.005
```

Рисунок 6 – Вычисление показателей для генератора mt

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutputRc4.dat
Мат. ожидание: 0.508
Среднекв. отклонение: 0.284
Погрешность мат. ожидания: 0.015
Погрешность среднекв. отклонения: 0.017
```

Рисунок 7 – Вычисление показателей для генератора rc4

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutputRsa.dat
Мат. ожидание: 0.508
Среднекв. отклонение: 0.288
Погрешность мат. ожидания: 0.015
Погрешность среднекв. отклонения: 0.002
```

Рисунок 8 – Вычисление показателей для генератора rsa

```
D:\cpp_projects\rnt\Debug>rnt.exe /f:stOutputBbs.dat
Мат. ожидание: 0.505
Среднекв. отклонение: 0.289
Погрешность мат. ожидания: 0.009
Погрешность среднекв. отклонения: 0.002
```

Рисунок 9 – Вычисление показателей для генератора bbs

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

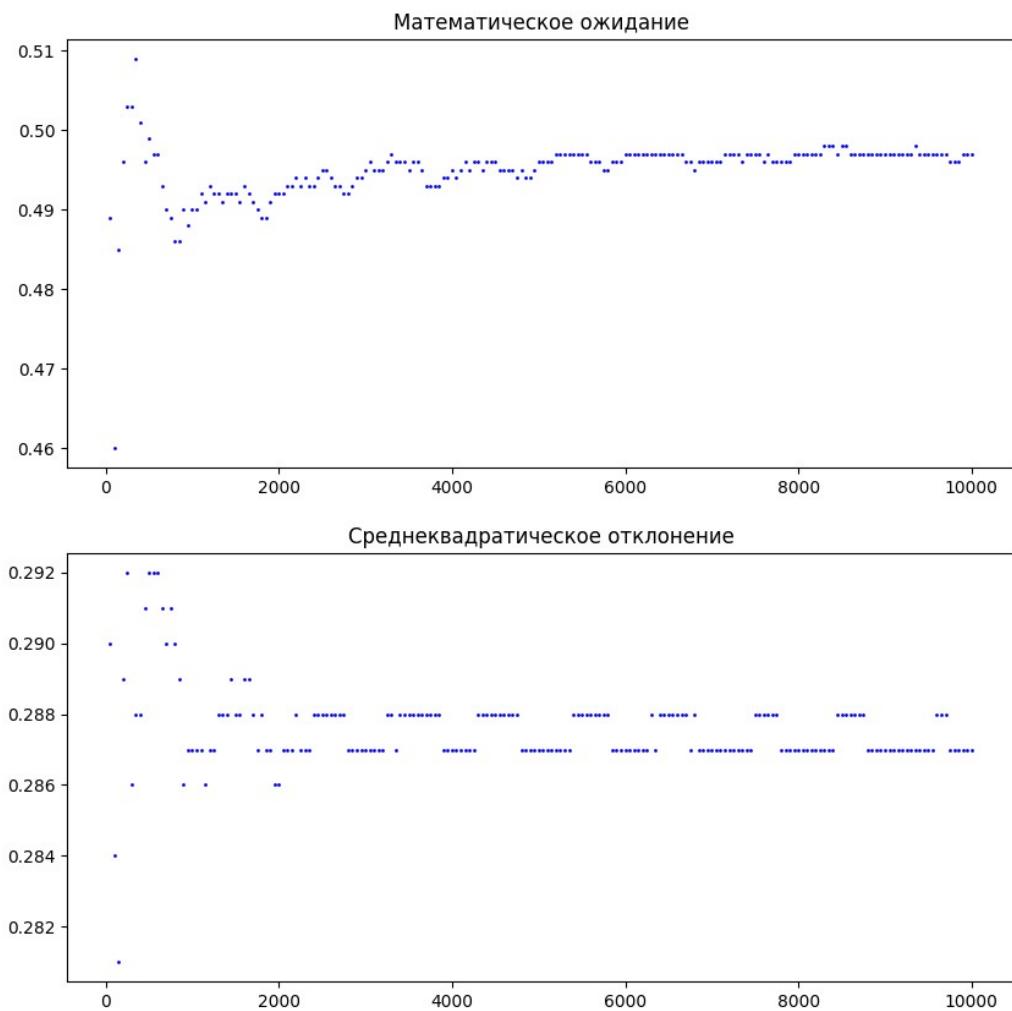


Рисунок 10 – Графики зависимости для генератора lc

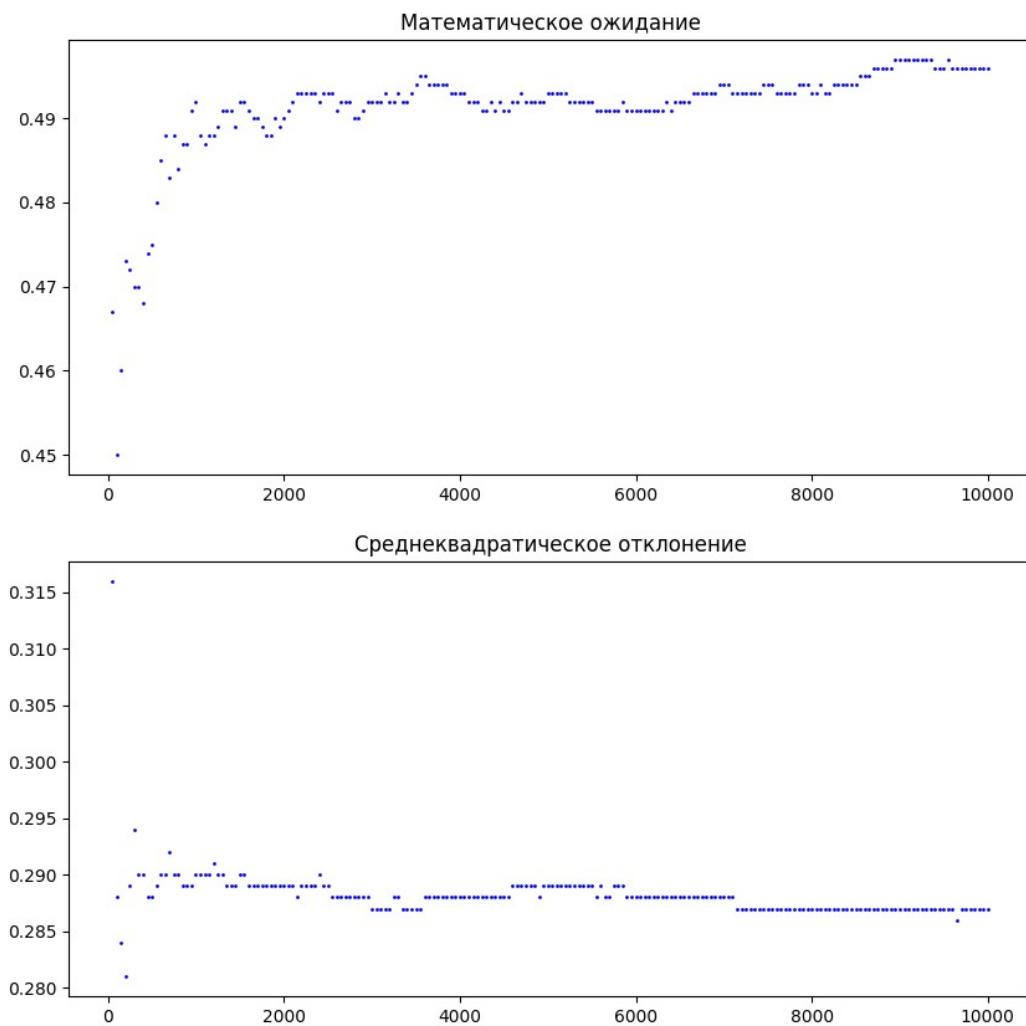


Рисунок 11 – Графики зависимости для генератора add

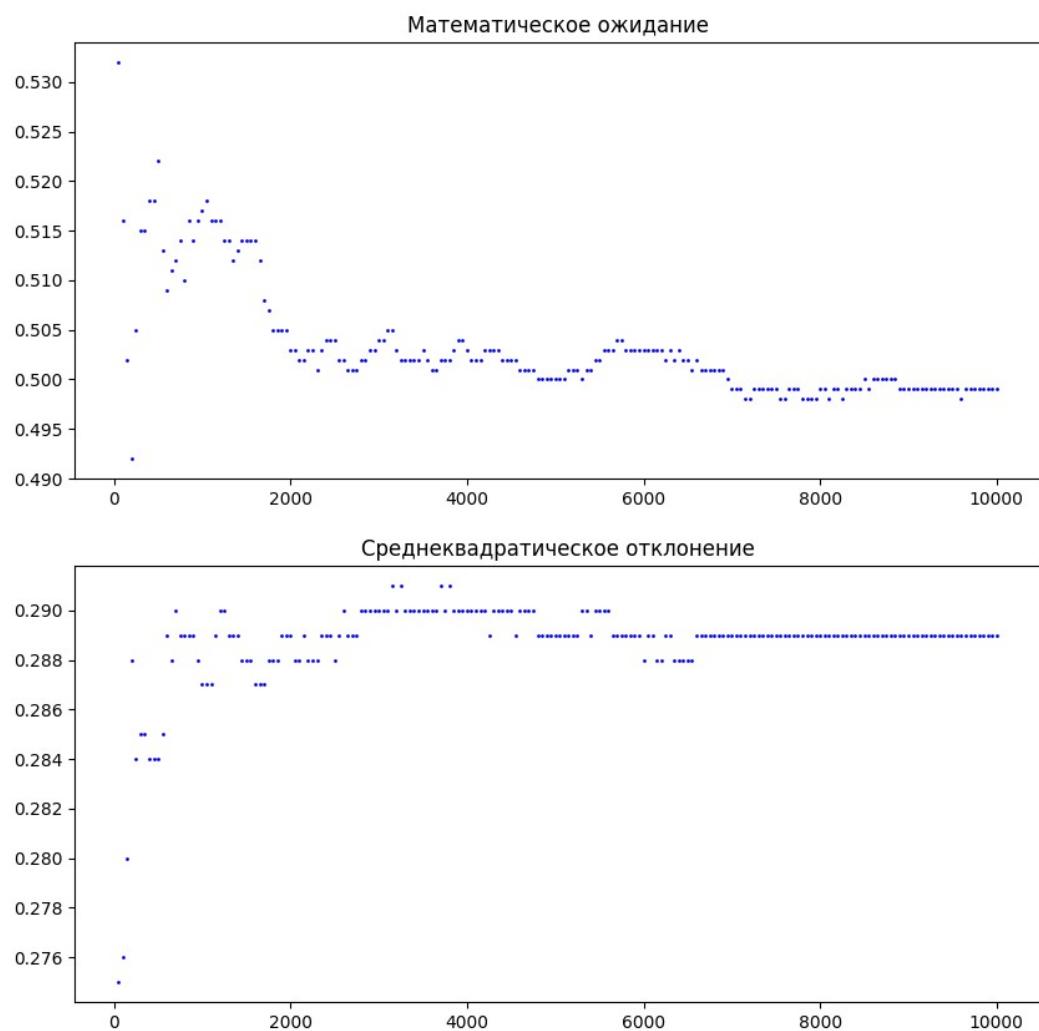


Рисунок 12 – Графики зависимости для генератора 5р

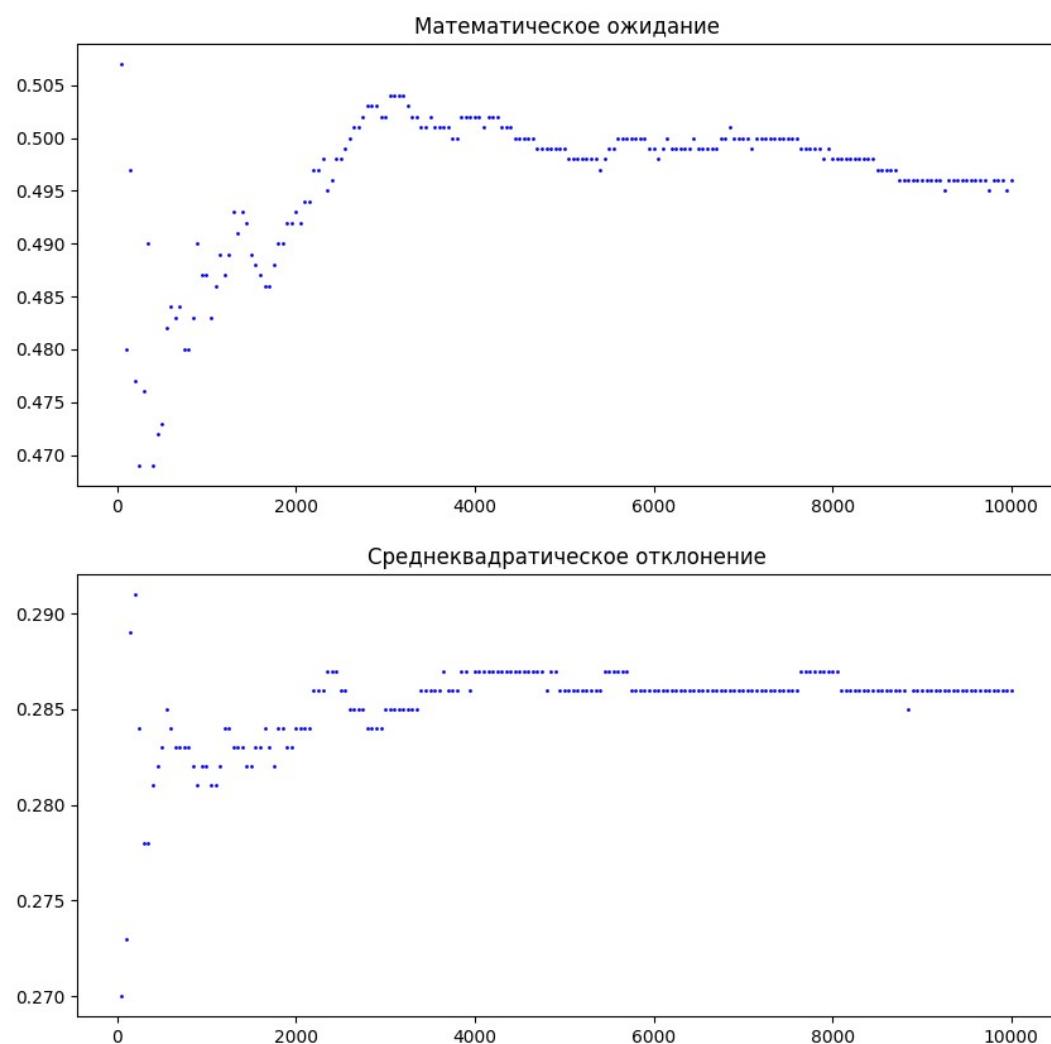


Рисунок 13 – Графики зависимости для генератора lfsr

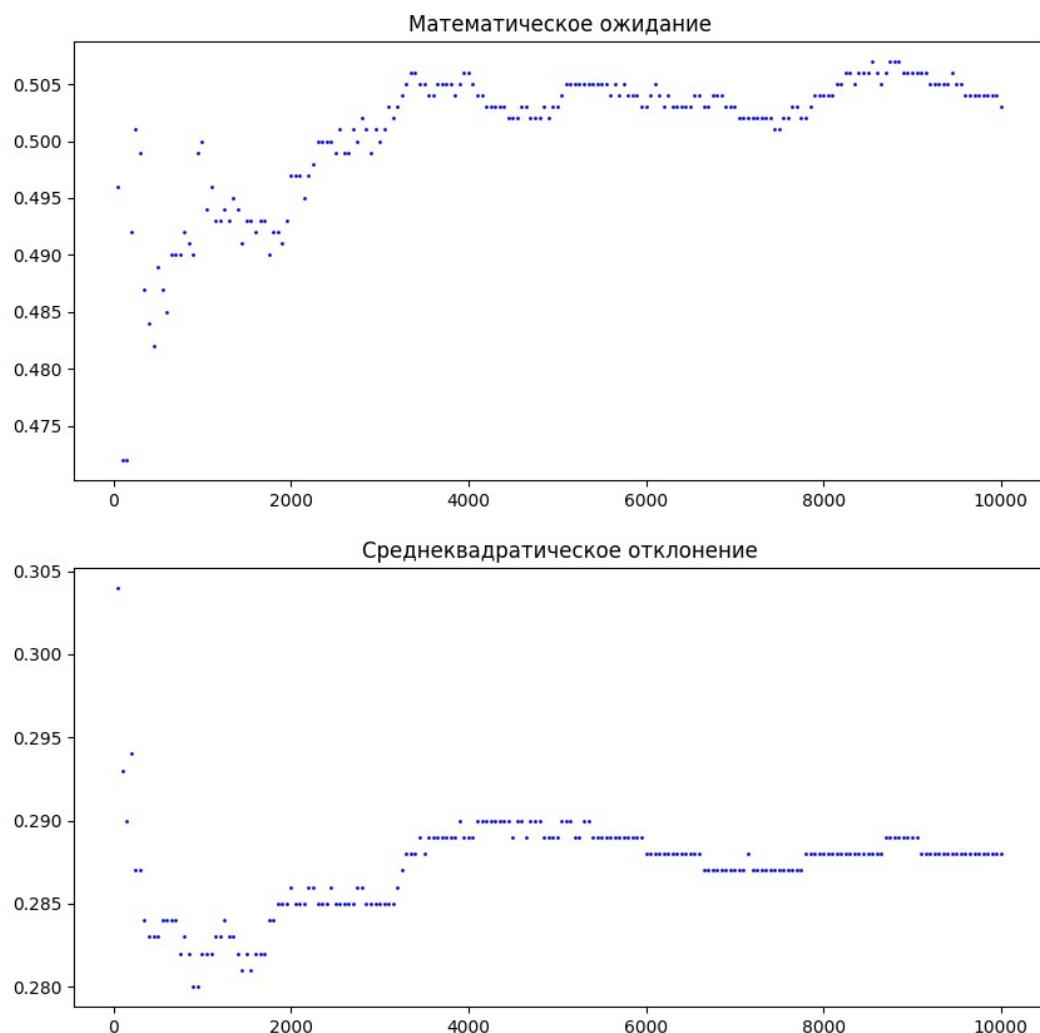


Рисунок 14 – Графики зависимости для генератора nfsr

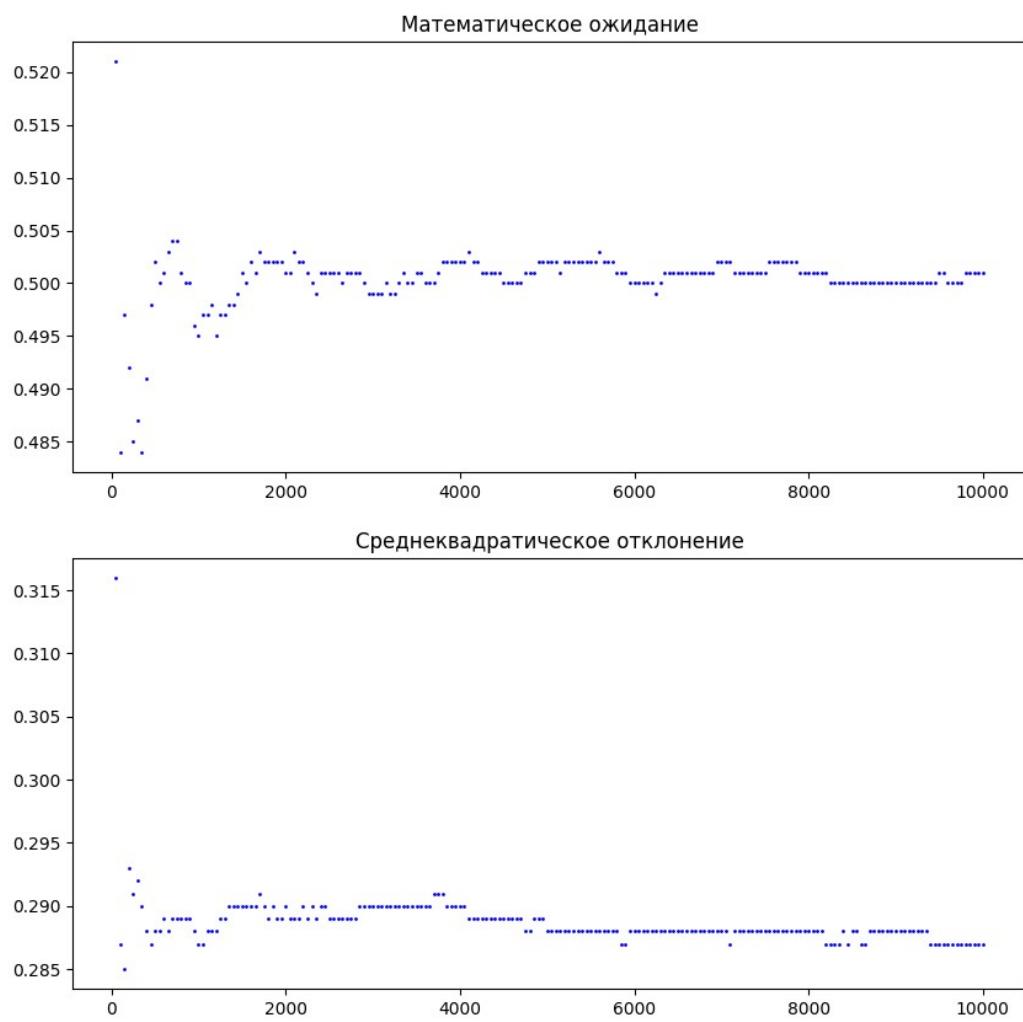


Рисунок 15 – Графики зависимости для генератора mt

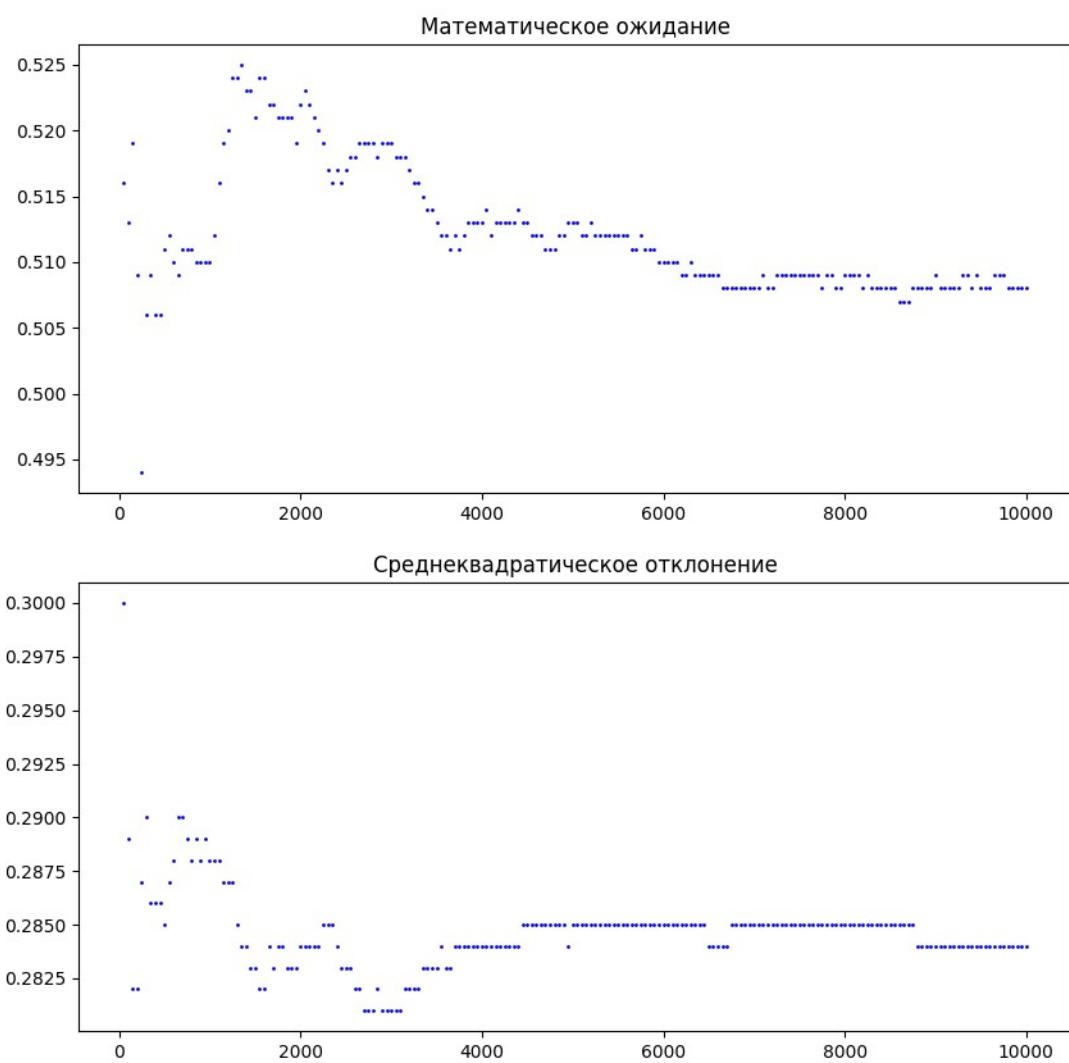


Рисунок 16 – Графики зависимости для генератора rc4

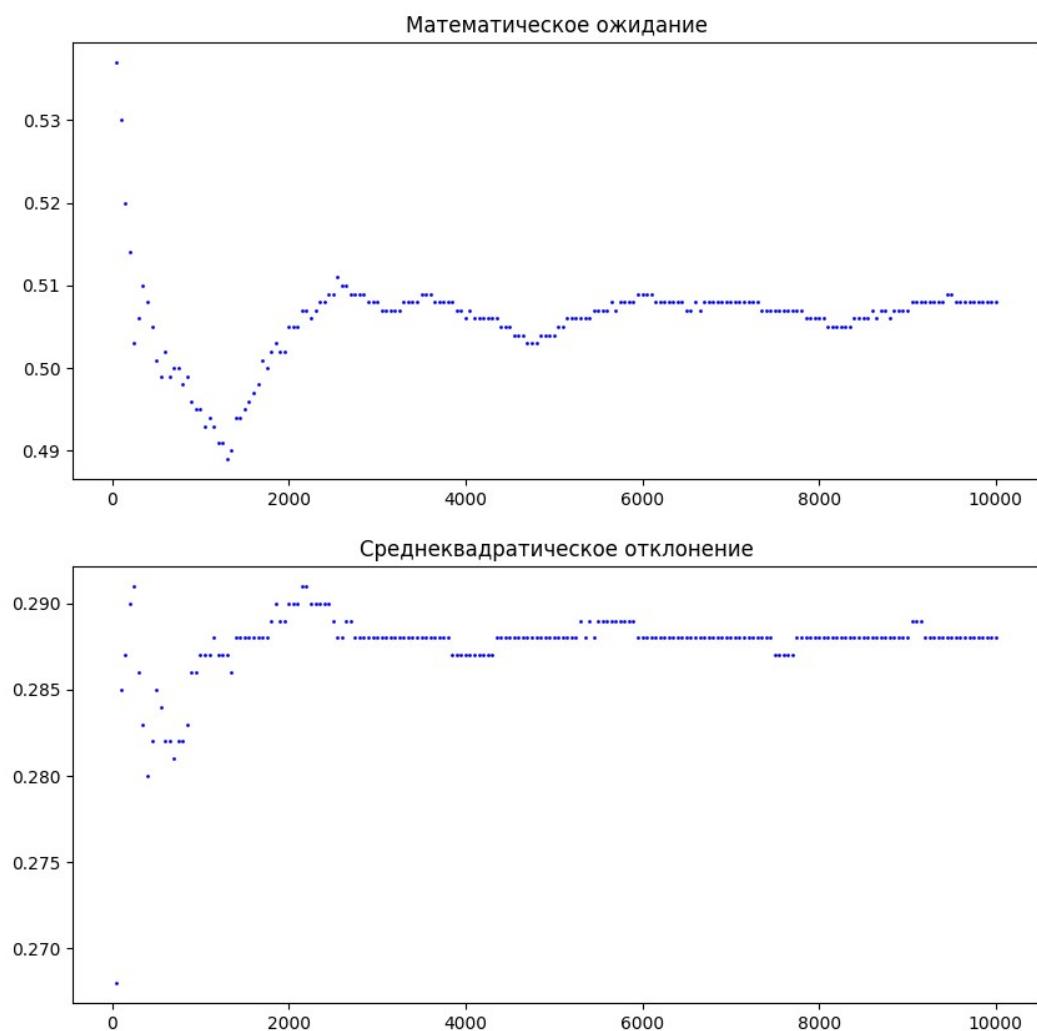


Рисунок 17 – Графики зависимости для генератора rsa

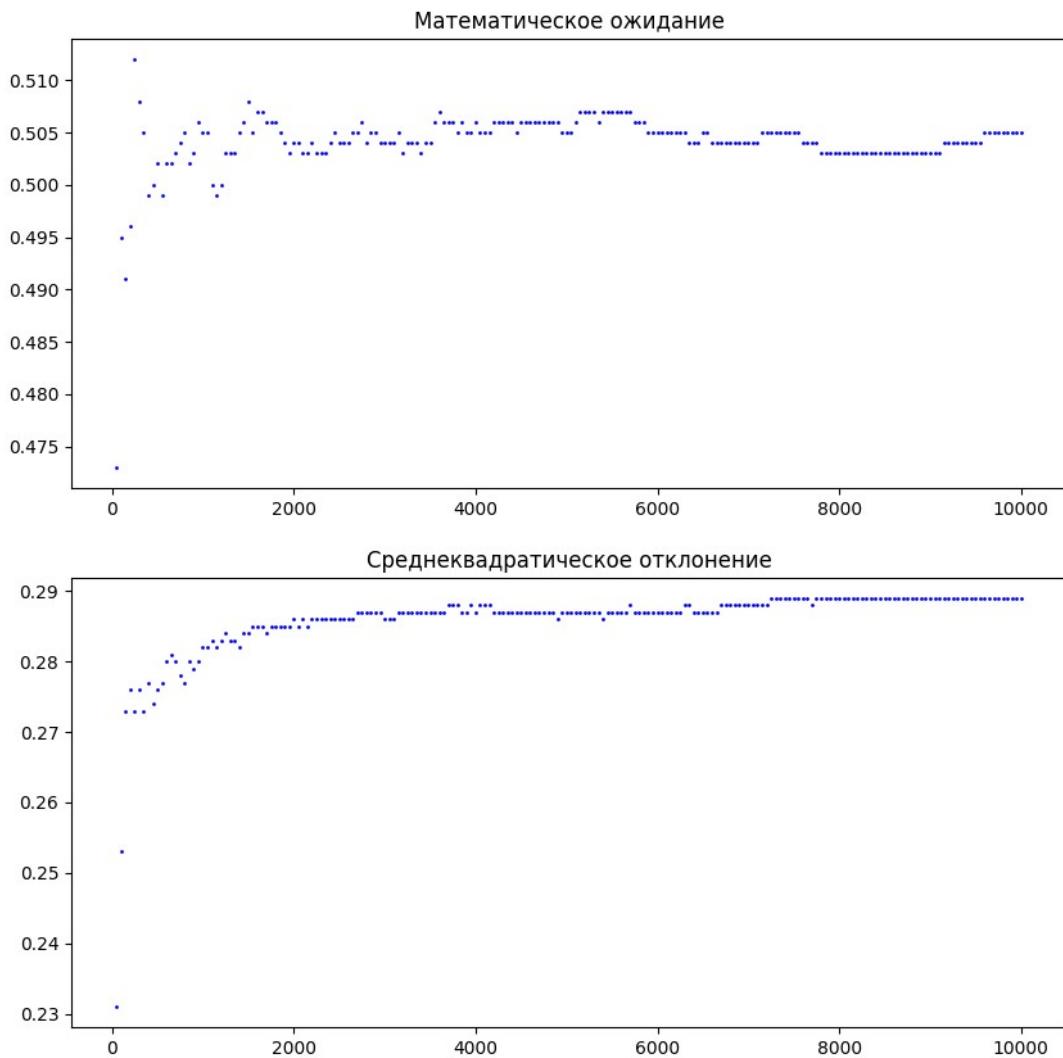


Рисунок 18 – Графики зависимости для генератора bbs

1.2 Сравнение полученных оценок с теоретическими и построение графиков

1.3 Проверка критериев

1.3.1 Критерий хи-квадрат

Описание критерия:

Проверка критерия χ^2 для некоторой последовательности чисел (или наблюдений величины X) будет состоять из следующих шагов:

1. Выполняем достаточное число независимых наблюдений.
2. Подсчитываем число n_i наблюдений попавших в каждый из интервалов $(a_i, b_i]$, $i = 1, \dots, k$.
3. Подсчитываем статистику

$$\chi^2 = \sum_{j=1}^k \frac{(n_j - E_j)^2}{E_j};$$

4. Определяем, находится ли вычисленная в доверительном интервале.

Приведем результаты выполнения программы для каждого из генераторов:

```
Критерий Хи-квадрат

> Количество интервалов: 14
> Ожидаемое число попаданий в интервалы: 714
> Число попаданий в каждый из интервалов:
  1: 727
  2: 704
  3: 712
  4: 736
  5: 712
  6: 709
  7: 726
  8: 718
  9: 710
  10: 720
  11: 727
  12: 709
  13: 709
  14: 681

> Критическое значение хи-квадрат для 13 степеней свобод: 22.362
> Значение критерия хи-квадрат с 13 степенью свободы: 3.2297

[+++] Критерий пройден
```

Рисунок 19 – Результаты выполнения программы для генератора lc

```
Критерий Хи-квадрат

> Количество интервалов: 14
> Ожидаемое число попаданий в интервалы: 714
> Число попаданий в каждый из интервалов:
  1: 696
  2: 740
  3: 719
  4: 743
  5: 687
  6: 756
  7: 762
  8: 654
  9: 694
  10: 748
  11: 710
  12: 690
  13: 750
  14: 651

> Критическое значение хи-квадрат для 13 степеней свобод: 22.362
> Значение критерия хи-квадрат с 13 степенью свободы: 24.7563

[--] Критерий не пройден
```

Рисунок 20 – Результаты выполнения программы для генератора add

```
-----  
Критерий Хи-квадрат  
  
> Количество интервалов: 14  
> Ожидаемое число попаданий в интервалы: 714  
> Число попаданий в каждый из интервалов:  
 1: 715  
 2: 698  
 3: 736  
 4: 731  
 5: 701  
 6: 680  
 7: 745  
 8: 705  
 9: 710  
10: 701  
11: 687  
12: 743  
13: 751  
14: 697  
  
> Критическое значение хи-квадрат для 13 степеней свобод: 22.362  
> Значение критерия хи-квадрат с 13 степенью свободы: 9.5378  
  
[+++] Критерий пройден  
-----
```

Рисунок 21 – Результаты выполнения программы для генератора 5р

```
-----  
Критерий Хи-квадрат  
  
> Количество интервалов: 14  
> Ожидаемое число попаданий в интервалы: 714  
> Число попаданий в каждый из интервалов:  
 1: 699  
 2: 732  
 3: 726  
 4: 738  
 5: 689  
 6: 724  
 7: 764  
 8: 686  
 9: 708  
10: 744  
11: 733  
12: 679  
13: 698  
14: 680  
  
> Критическое значение хи-квадрат для 13 степеней свобод: 22.362  
> Значение критерия хи-квадрат с 13 степенью свободы: 12.902  
  
[+++] Критерий пройден  
-----
```

Рисунок 22 – Результаты выполнения программы для генератора lfsr

```
-----  
Критерий Хи-квадрат  
  
> Количество интервалов: 14  
> Ожидаемое число попаданий в интервалы: 714  
> Число попаданий в каждый из интервалов:  
 1: 688  
 2: 713  
 3: 688  
 4: 702  
 5: 727  
 6: 704  
 7: 718  
 8: 722  
 9: 680  
10: 737  
11: 726  
12: 727  
13: 755  
14: 713  
  
> Критическое значение хи-квадрат для 13 степеней свобод: 22.362  
> Значение критерия хи-квадрат с 13 степенью свободы: 7.7395  
  
[+++] Критерий пройден  
-----
```

Рисунок 23 – Результаты выполнения программы для генератора nfsr

```
-----  
Критерий Хи-квадрат  
  
> Количество интервалов: 14  
> Ожидаемое число попаданий в интервалы: 714  
> Число попаданий в каждый из интервалов:  
 1: 707  
 2: 691  
 3: 709  
 4: 729  
 5: 723  
 6: 718  
 7: 669  
 8: 686  
 9: 743  
10: 750  
11: 710  
12: 766  
13: 721  
14: 678  
  
> Критическое значение хи-квадрат для 13 степеней свобод: 22.362  
> Значение критерия хи-квадрат с 13 степенью свободы: 13.916  
  
[+++] Критерий пройден  
-----
```

Рисунок 24 – Результаты выполнения программы для генератора mt

```
-----  
Критерий Хи-квадрат  
  
> Количество интервалов: 14  
> Ожидаемое число попаданий в интервалы: 714  
> Число попаданий в каждый из интервалов:  
 1: 798  
 2: 641  
 3: 711  
 4: 580  
 5: 306  
 6: 904  
 7: 756  
 8: 688  
 9: 1125  
10: 583  
11: 960  
12: 571  
13: 863  
14: 464  
  
> Критическое значение хи-квадрат для 13 степеней свобод: 22.362  
> Значение критерия хи-квадрат с 13 степенью свободы: 822.272  
  
[---] Критерий не пройден  
-----
```

Рисунок 25 – Результаты выполнения программы для генератора rc4

```
-----  
Критерий Хи-квадрат  
  
> Количество интервалов: 14  
> Ожидаемое число попаданий в интервалы: 714  
> Число попаданий в каждый из интервалов:  
 1: 583  
 2: 633  
 3: 803  
 4: 738  
 5: 743  
 6: 838  
 7: 726  
 8: 507  
 9: 767  
10: 631  
11: 768  
12: 686  
13: 760  
14: 817  
  
> Критическое значение хи-квадрат для 13 степеней свобод: 22.362  
> Значение критерия хи-квадрат с 13 степенью свободы: 164.639  
  
[---] Критерий не пройден  
-----
```

Рисунок 26 – Результаты выполнения программы для генератора rsa

```

-----
Критерий Хи-квадрат

> Количество интервалов: 14
> Ожидаемое число попаданий в интервалы: 714
> Число попаданий в каждый из интервалов:
  1: 713
  2: 723
  3: 673
  4: 717
  5: 703
  6: 662
  7: 727
  8: 731
  9: 672
  10: 706
  11: 777
  12: 749
  13: 738
  14: 709

> Критическое значение хи-квадрат для 13 степеней свобод: 22.362
> Значение критерия хи-квадрат с 13 степенью свободы: 17.7563

[+++] Критерий пройден
-----
```

Рисунок 27 – Результаты выполнения программы для генератора bbs

Исходный код программы:

```

bool checkXiSquareCriterion(vector <double> allElems) {
    double maxElem = -1.0, minElem = 1.0;
    int elemsCount = allElems.size();
    for (int i = 0; i < elemsCount; i++)
    {
        double checkElem = allElems[i];
        if (checkElem > maxElem)
            maxElem = checkElem;
        if (checkElem < minElem)
            minElem = checkElem;
    }
    vector <pair <double, double>> allIntervals;
    int intervalsCount = ceil(1 + 1.4 * log(elemsCount));
    double IntS = (round(((maxElem - minElem) / intervalsCount) * 10000)) / 10000;
    for (int i = 0; i < intervalsCount; i++) {
        double a_i = minElem + IntS * i;
        double b_i = minElem + IntS * (i + 1);
        allIntervals.push_back(make_pair(a_i, b_i));
    }
    double expectedNumOfHits = round((elemsCount / intervalsCount) * 1000) / 1000;
    vector <int> realDistribution(intervalsCount, 0);
    for (int i = 0; i < elemsCount; i++)
        for (int j = 0; j < intervalsCount; j++)
            if (allIntervals[j].first <= allElems[i] && allElems[i] < allIntervals[j].second) {
                realDistribution[j]++;
                break;
            }
    double XiSquareDistribution = 0;
```

```

    for (int i = 0; i < intervalsCount; i++)
        XiSquareDistribution = XiSquareDistribution + ((realDistribution[i] -
→ expectedNumOfHits) * (realDistribution[i] - expectedNumOfHits)) / expectedNumOfHits;
        XiSquareDistribution = round(XiSquareDistribution * 10000) / 10000;
        double criticalImportance = 22.36203;
        cout << "\n > Количество интервалов: " << intervalsCount;
        int k = intervalsCount - 1;
        cout << "\n > Ожидаемое число попаданий в интервалы: " << expectedNumOfHits;
        cout << "\n > Число попаданий в каждый из интервалов: \n";
        for (int i = 0; i < intervalsCount; i++)
            cout << "    " << i + 1 << ": " << realDistribution[i] << endl;
            cout << "\n > Критическое значение хи-квадрат для " << k << " степеней свобод: " <<
→ criticalImportance;
            cout << "\n > Значение критерия хи-квадрат с " << k << " степенью свободы: " <<
→ XiSquareDistribution << "\n";
            if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
                cout << "\n[++] Критерий пройден\n";
                return true;
            }
            else {
                cout << "\n[--] Критерий не пройден\n";
                return false;
            }
        }
    }
}

```

1.3.2 Критерий серий

Описание критерия:

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

1. Воспользуемся критерием Хи-квадрат. Для этого преобразуем последовательность $X^n = (x_1, x_2, \dots, x_n)$ в $Y^n = (d[x_1], d[x_2], \dots, d[n])$ с некоторым d (в реализации $d = 4$).
2. Подсчитываем количество совпадений

$$(y_{2j}, y_{2j+1}) = (q, r), 0 \leq j \leq n, 0 \leq q, r \leq d.$$

3. Применяем Хи-квадрат критерий к полученному набору с параметрами:

$$k = d^2, p_j = \frac{1}{d^2};$$

Приведем результаты выполнения программы для каждого из генераторов:

Критерий серий

```
> Параметр d: 4
> Ожидаемое количество чисел в каждой категории: 312
> Количество чисел в каждой категории:
  1: 313
  2: 314
  3: 321
  4: 305
  6: 317
  7: 316
  8: 325
  9: 304
 11: 316
 12: 313
 13: 323
 14: 307
 16: 305
 17: 317
 18: 292
 19: 312

> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962
> Значение критерия хи-квадрат с 16 степенью свободы: 3.353

[***] Критерий пройден
```

Рисунок 28 – Результаты выполнения программы для генератора lc

Критерий серий

```
> Параметр d: 4
> Ожидаемое количество чисел в каждой категории: 312
> Количество чисел в каждой категории:
  1: 333
  2: 321
  3: 316
  4: 308
  6: 348
  7: 315
  8: 317
  9: 293
 11: 301
 12: 319
 13: 277
 14: 326
 16: 289
 17: 326
 18: 310
 19: 301

> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962
> Значение критерия хи-квадрат с 16 степенью свободы: 15.019

[***] Критерий пройден
```

Рисунок 29 – Результаты выполнения программы для генератора add

```
-----  
Критерий серий
```

```
> Параметр d: 4  
> Ожидаемое количество чисел в каждой категории: 312  
> Количество чисел в каждой категории:  
1: 340  
2: 271  
3: 389  
4: 287  
6: 309  
7: 292  
8: 286  
9: 350  
11: 281  
12: 371  
13: 254  
14: 341  
16: 301  
17: 317  
18: 314  
19: 297  
  
> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962  
> Значение критерия хи-квадрат с 16 степенью свободы: 65.929  
  
[---] Критерий не пройден  
-----
```

Рисунок 30 – Результаты выполнения программы для генератора 5р

```
-----  
Критерий серий
```

```
> Параметр d: 4  
> Ожидаемое количество чисел в каждой категории: 312  
> Количество чисел в каждой категории:  
1: 324  
2: 356  
3: 304  
4: 307  
6: 588  
7: 25  
8: 615  
9: 23  
11: 316  
12: 326  
13: 330  
14: 285  
16: 16  
17: 579  
18: 22  
19: 584  
  
> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962  
> Значение критерия хи-квадрат с 16 степенью свободы: 2097.11  
  
[---] Критерий не пройден  
-----
```

Рисунок 31 – Результаты выполнения программы для генератора lfsr

Критерий серий

```
> Параметр d: 4
> Ожидаемое количество чисел в каждой категории: 312
> Количество чисел в каждой категории:
 1: 534
 2: 82
 3: 505
 4: 67
 6: 642
 7: 0
 8: 649
 9: 0
11: 74
12: 545
13: 89
14: 577
16: 0
17: 584
18: 0
19: 652

> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962
> Значение критерия хи-квадрат с 16 степенью свободы: 3947.99
```

[---] Критерий не пройден

Рисунок 32 – Результаты выполнения программы для генератора nfsr

Критерий серий

```
> Параметр d: 4
> Ожидаемое количество чисел в каждой категории: 312
> Количество чисел в каждой категории:
 1: 312
 2: 292
 3: 283
 4: 327
 6: 318
 7: 311
 8: 319
 9: 310
11: 320
12: 307
13: 346
14: 341
16: 305
17: 309
18: 298
19: 302

> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962
> Значение критерия хи-квадрат с 16 степенью свободы: 12.808
```

[+++] Критерий пройден

Рисунок 33 – Результаты выполнения программы для генератора mt

Критерий серий

```
> Параметр d: 4
> Ожидаемое количество чисел в каждой категории: 312
> Количество чисел в каждой категории:
 1: 283
 2: 251
 3: 353
 4: 298
 6: 287
 7: 238
 8: 316
 9: 304
11: 381
12: 354
13: 421
14: 351
16: 283
17: 252
18: 365
19: 263

> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962
> Значение критерия хи-квадрат с 16 степенью свободы: 135.25

[---] Критерий не пройден
```

Рисунок 34 – Результаты выполнения программы для генератора rc4

Критерий серий

```
> Параметр d: 4
> Ожидаемое количество чисел в каждой категории: 312
> Количество чисел в каждой категории:
 1: 263
 2: 306
 3: 277
 4: 352
 6: 332
 7: 380
 8: 272
 9: 361
11: 294
12: 297
13: 303
14: 293
16: 345
17: 304
18: 283
19: 338

> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962
> Значение критерия хи-квадрат с 16 степенью свободы: 57.526

[---] Критерий не пройден
```

Рисунок 35 – Результаты выполнения программы для генератора rsa

```

-----
Критерий серий

> Параметр d: 4
> Ожидаемое количество чисел в каждой категории: 312
> Количество чисел в каждой категории:
 1: 285
 2: 322
 3: 311
 4: 331
 6: 304
 7: 281
 8: 294
 9: 315
11: 293
12: 325
13: 317
14: 327
16: 339
17: 326
18: 296
19: 334

> Критическое значение хи-квадрат для 16 степеней свобод: 26.2962
> Значение критерия хи-квадрат с 16 степенью свободы: 16.006

[***] Критерий пройден
-----
```

Рисунок 36 – Результаты выполнения программы для генератора bbs

Исходный код программы:

```

bool checkSeriesCriterion(vector <double> allElems) {
    int d = 4;
    vector <vector <int>> nPairs(d);
    for (int i = 0; i < d; i++)
        nPairs[i].resize(d, 0);
    int k = d * d;
    vector <pair <int, int>> pairCategories;
    int elemsCount = allElems.size();
    if (elemsCount % 2 != 0)
        elemsCount--;
    for (int j = 0; j < elemsCount; j+=2)
    {
        double leftI = floor(allElems[j] * d);
        double rightI = floor(allElems[j + 1] * d);
        pairCategories.push_back(make_pair(leftI, rightI));
    }
    double criticalImportance = 26.29623;
    double expectedNumOfHits = round((elemsCount / (2 * k)) * 1000) / 1000;
    for (int i = 0; i < pairCategories.size(); i++) {
        nPairs[pairCategories[i].first][pairCategories[i].second] += 1;
    }
    double XiSquareDistribution = 0;
    for (int i = 0; i < nPairs.size(); i++)
        for (int x = 0; x < nPairs[i].size(); x++)
            XiSquareDistribution += ((nPairs[i][x] - expectedNumOfHits) * (nPairs[i][x] -
→ expectedNumOfHits)) / expectedNumOfHits;
    XiSquareDistribution = round(XiSquareDistribution * 1000) / 1000;
    cout << "\n > Параметр d: " << d;
```

```

cout << "\n > Ожидаемое количество чисел в каждой категории: " << expectedNumOfHits;
cout << "\n > Количество чисел в каждой категории: \n";
int chCount = 0;
for (int i = 0; i < nPairs.size(); i++) {
    for (int x = 0; x < nPairs[i].size(); x++) {
        chCount++;
        cout << "    " << chCount << ":" << nPairs[i][x] << "\n";
    }
    chCount++;
}
cout << "\n > Критическое значение хи-квадрат для " << k << " степеней свобод: " <<
criticalImportance;
cout << "\n > Значение критерия хи-квадрат с " << k << " степенью свободы: " <<
XiSquareDistribution << "\n";
if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
    cout << "\n[++] Критерий пройден\n";
    return true;
}
else {
    cout << "\n[--] Критерий не пройден\n";
    return false;
}
}

```

1.3.3 Критерий интервалов

Описание критерия:

Пусть a и b – два действительных числа таких, что $0 \leq a < b \leq 1$. Рассмотрим длины подпоследовательностей $x_j, x_{j+1}, \dots, x_{j+r}$, в которых $x_j, x_{j+1}, \dots, x_{j+r-1} \notin [a, b]$, $x_{j+r} \in [a, b]$. Такую последовательность будем называть интервалом длины r .

Сначала, нам нужно подсчитать число интервалов длиной $0, 1, \dots, n$.

Шаги алгоритма подсчета числа интервалов:

1. Инициализация. Присвоить $j = -1$, $s = 0$, $c_r = 0$, $0 \leq r \leq t$.
2. $r = 0$.
3. $j = j + 1$. Если $a \leq x_j \leq b$, то переход на шаг 5.
4. $r = r + 1$. Переход к шагу 3.
5. Если $r \leq t$, то $c_t = c_t + 1$, иначе – $c_r = c_r + 1$.
6. $s = s + 1$. Если $s < n$ то переход на шаг 2.

После этого мы можем применить хи-квадрат критерий для $k = t + 1$ к значениям c_i , $i = 0, 1, \dots, t$ с параметрами

$$p_r = p(1-p)r \text{ для } 0 \leq r \leq t - 1; p_t = (1-p)t; p = (a - b).$$

Здесь p – вероятность того, что $a \leq x_j \leq b$. Значения n и t выбираются так, чтобы ожидаемое значение c_r было больше 5.

Приведем результаты выполнения программы для каждого из генераторов:

```
-----  
Критерий интервалов  
  
> Максимальная длина интервала (t): 10  
> Количество интервалов (n): 1000  
> Границы: alpha = 0.411 beta = 0.654  
> Пересчитанные вероятности Pr и Pt:  
 0.243 0.184 0.139 0.105 0.08 0.06 0.046 0.035 0.026 0.02 0.0617958  
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):  
 250 185 134 106 83 54 63 21 27 19 58  
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751  
> Значение критерия хи-квадрат с 11 степенью свободы: 13.313  
  
[+++] Критерий пройден  
-----
```

Рисунок 37 – Результаты выполнения программы для генератора lc

```
-----  
Критерий интервалов  
  
> Максимальная длина интервала (t): 10  
> Количество интервалов (n): 1000  
> Границы: alpha = 0.107 beta = 0.342  
> Пересчитанные вероятности Pr и Pt:  
 0.235 0.18 0.138 0.105 0.08 0.062 0.047 0.036 0.028 0.021 0.0686459  
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):  
 254 169 128 110 76 58 37 29 13 68  
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751  
> Значение критерия хи-квадрат с 11 степенью свободы: 9.321  
  
[+++] Критерий пройден  
-----
```

Рисунок 38 – Результаты выполнения программы для генератора add

```
-----  
Критерий интервалов  
  
> Максимальная длина интервала (t): 10  
> Количество интервалов (n): 1000  
> Границы: alpha = 0.397 beta = 0.666  
> Пересчитанные вероятности Pr и Pt:  
 0.269 0.197 0.144 0.105 0.077 0.056 0.041 0.03 0.022 0.016 0.0435686  
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):  
 252 222 122 115 73 52 53 29 22 14 46  
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751  
> Значение критерия хи-квадрат с 11 степенью свободы: 12.985  
  
[+++] Критерий пройден  
-----
```

Рисунок 39 – Результаты выполнения программы для генератора 5р

```

-----
Критерий интервалов

> Максимальная длина интервала (t): 10
> Количество интервалов (n): 1000
> Границы: alpha = 0.01 beta = 0.395
> Пересчитанные вероятности Pr и Pt:
0.385 0.237 0.146 0.09 0.055 0.034 0.021 0.013 0.008 0.005 0.00774018
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):
331 296 150 77 59 28 16 17 12 5 9
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751
> Значение критерия хи-квадрат с 11 степенью свободы: 30.225

[---] Критерий не пройден
-----
```

Рисунок 40 – Результаты выполнения программы для генератора lfsr

```

-----
Критерий интервалов

> Максимальная длина интервала (t): 10
> Количество интервалов (n): 1000
> Границы: alpha = 0.482 beta = 0.698
> Пересчитанные вероятности Pr и Pt:
0.216 0.169 0.133 0.104 0.082 0.064 0.05 0.039 0.031 0.024 0.0077325
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):
0 241 234 145 110 68 49 40 21 27 65
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751
> Значение критерия хи-квадрат с 11 степенью свободы: 358.885

[---] Критерий не пройден
-----
```

Рисунок 41 – Результаты выполнения программы для генератора nfsr

```

-----
Критерий интервалов

> Максимальная длина интервала (t): 10
> Количество интервалов (n): 1000
> Границы: alpha = 0.343 beta = 0.71
> Пересчитанные вероятности Pr и Pt:
0.367 0.232 0.147 0.093 0.059 0.037 0.024 0.015 0.009 0.006 0.0103285
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):
384 217 141 86 65 39 21 16 10 9 12
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751
> Значение критерия хи-квадрат с 11 степенью свободы: 5.571

[+++] Критерий пройден
-----
```

Рисунок 42 – Результаты выполнения программы для генератора mt

```

-----
Критерий интервалов

> Максимальная длина интервала (t): 10
> Количество интервалов (n): 1000
> Границы: alpha = 0.53 beta = 0.91
> Пересчитанные вероятности Pr и Pt:
  0.38 0.236 0.146 0.091 0.056 0.035 0.022 0.013 0.008 0.005 0.00839299
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):
  434 249 140 69 51 25 15 9 6 1 1
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751
> Значение критерия хи-квадрат с 11 степенюю свободы: 30.929

[---] Критерий не пройден
-----
```

Рисунок 43 – Результаты выполнения программы для генератора rc4

```

-----
Критерий интервалов

> Максимальная длина интервала (t): 10
> Количество интервалов (n): 1000
> Границы: alpha = 0.281 beta = 0.523
> Пересчитанные вероятности Pr и Pt:
  0.242 0.183 0.139 0.105 0.08 0.061 0.046 0.035 0.026 0.02 0.062617
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):
  282 171 133 117 61 55 64 25 22 24 46
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751
> Значение критерия хи-квадрат с 11 степенюю свободы: 29.857

[---] Критерий не пройден
-----
```

Рисунок 44 – Результаты выполнения программы для генератора rsa

```

-----
Критерий интервалов

> Максимальная длина интервала (t): 10
> Количество интервалов (n): 1000
> Границы: alpha = 0.585 beta = 0.877
> Пересчитанные вероятности Pr и Pt:
  0.292 0.207 0.146 0.104 0.073 0.052 0.037 0.026 0.018 0.013 0.031647
> Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = 10):
  279 216 138 106 89 53 38 24 16 14 27
> Критическое значение хи-квадрат для 11 степеней свобод: 19.6751
> Значение критерия хи-квадрат с 11 степенюю свободы: 6.135

[+++] Критерий пройден
-----
```

Рисунок 45 – Результаты выполнения программы для генератора bbs

Исходный код программы:

```

bool checkIntervalsCriterion(vector <double> allElems) {
    int t = 10, n = 1000;
    srand((unsigned int)time(0));
    double alpha = (double)(rand()) / RAND_MAX, beta = (double)(rand()) / RAND_MAX;
    while (0.2 >= abs(alpha - beta) || abs(alpha - beta) >= 0.4) {
        alpha = (double)(rand()) / RAND_MAX;
        beta = (double)(rand()) / RAND_MAX;
```

```

}

if (beta < alpha) {
    double dopD = alpha;
    alpha = beta;
    beta = dopD;
}

alpha = round(alpha * 1000) / 1000;
beta = round(beta * 1000) / 1000;
double p = beta - alpha;
int s = 0;
vector <int> count, resCount;
double XiSquareDistribution = 100000000000.0;
double criticalImportance = 100000000000.0;
int resT = 100000000000;
int intervalsCount = 100000000000;
vector <double> resPs;
s = 0;
count.resize(t + 1, 0);
int r = 0;
for (int j = 0; j < allElems.size(); j++) {
    if (alpha <= allElems[j] && allElems[j] < beta) {
        if (r >= t)
            count[t]++;
        else
            count[r]++;
        s++;
        if (s < n)
            r = 0;
        else
            break;
    }
    else
        r++;
}
vector <double> probs;
for (int j = 0; j < t; j++)
probs.push_back(round((p * (powDouble(1.0 - p, j))) * 1000) / 1000);
double XiSch = 0.0;
probs.push_back(powDouble(1.0 - p, t));
double criticalImpDop = InterVector[t];
int qN = 0;
for (int j = 0; j < count.size(); j++)
qN += count[j];
for (int j = 0; j < probs.size(); j++) {
    double est = qN * probs[j];
    if (est == 0.0)
        break;
    XiSch += ((count[j] - est) * (count[j] - est)) / est;
}

```

```

    }

    if (XiSch < XiSquareDistribution) {
        XiSquareDistribution = XiSch;
        criticalImportance = criticalImpDop;
        resT = t;
        intervalsCount = n;
        resPs = probs;
        resCount = count;
    }

    int k = resT + 1;
    cout << "\n > Максимальная длина интервала (t): " << resT;
    cout << "\n > Количество интервалов (n): " << intervalsCount;
    cout << "\n > Границы: alpha = " << alpha << " beta = " << beta;
    cout << "\n > Пересчитанные вероятности Pr и Pt: \n";
    cout << "      ";
    for (int j = 0; j < resPs.size(); j++)
        cout << resPs[j] << " ";
    cout << "\n > Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = "
    ← << resT << "): \n";
    cout << "      ";
    for (int j = 0; j < resCount.size(); j++)
        cout << resCount[j] << " ";
    cout << "\n > Критическое значение хи-квадрат для " << k << " степеней свобод: " <<
    ← criticalImportance;
    cout << "\n > Значение критерия хи-квадрат с " << k << " степенью свободы: " <<
    ← round(XiSquareDistribution * 1000) / 1000 << "\n";
    if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
        cout << "\n[++] Критерий пройден\n";
        return true;
    }
    else {
        cout << "\n[--] Критерий не пройден\n";
        return false;
    }
}

```

1.3.4 Критерий разбиений

Описание критерия:

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

$$p_r = \frac{d(d-1)\dots(d-r+1)}{d^k} \binom{k}{r}.$$

Здесь $\{^k_r\} = S(n, k)$ – числа Стирлинга, задающие число способов разбиения множества из n элементов на k непересекающихся подмножеств, которые можно вычислить по формуле:

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k+j} \binom{k}{j} j^n,$$

$$\binom{k}{j} = \frac{k!}{j!(k-j)!}.$$

Так как вероятности p_r 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 p_r , следует подсчитать, сколько d^k групп из k чисел, расположенных между 0 и $d - 1$, имеют точно r различных элементов, и разделить это число на d^k .

Приведем результаты выполнения программы для каждого из генераторов:

```
-----  
Критерий разбиений  
  
> Параметр d: 8  
> Параметр k: 5 Количество пятерок, где 1 различных значений: 0  
Количество пятерок, где 2 различных значений: 50  
Количество пятерок, где 3 различных значений: 514  
Количество пятерок, где 4 различных значений: 1004  
Количество пятерок, где 5 различных значений: 432  
  
> Теоретические вероятности для каждой из пятерок: 1: 0.0002  
2: 0.0256  
3: 0.2563  
4: 0.5127  
5: 0.2051  
  
> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773  
> Значение критерия хи-квадрат с 4 степенью свободы: 2.037  
  
[+++] Критерий пройден  
-----
```

Рисунок 46 – Результаты выполнения программы для генератора lc

```
-----  
Критерий разбиений  
  
> Параметр d: 8  
> Параметр k: 5 Количество пятерок, где 1 различных значений: 2  
Количество пятерок, где 2 различных значений: 48  
Количество пятерок, где 3 различных значений: 482  
Количество пятерок, где 4 различных значений: 1057  
Количество пятерок, где 5 различных значений: 411  
  
> Теоретические вероятности для каждой из пятерок: 1: 0.0002  
2: 0.0256  
3: 0.2563  
4: 0.5127  
5: 0.2051  
  
> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773  
> Значение критерия хи-квадрат с 4 степенью свободы: 9.402  
  
[+++] Критерий пройден  
-----
```

Рисунок 47 – Результаты выполнения программы для генератора add

Критерий разбиений

```
> Параметр d: 8
> Параметр k: 5 Количество пятерок, где 1 различных значений: 1
Количество пятерок, где 2 различных значений: 60
Количество пятерок, где 3 различных значений: 500
Количество пятерок, где 4 различных значений: 1009
Количество пятерок, где 5 различных значений: 430

> Теоретические вероятности для каждой из пятерок: 1: 0.0002
2: 0.0256
3: 0.2563
4: 0.5127
5: 0.2051

> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773
> Значение критерия хи-квадрат с 4 степенью свободы: 3.94

[+++] Критерий пройден
```

Рисунок 48 – Результаты выполнения программы для генератора 5р

Критерий разбиений

```
> Параметр d: 8
> Параметр k: 5 Количество пятерок, где 1 различных значений: 2
Количество пятерок, где 2 различных значений: 92
Количество пятерок, где 3 различных значений: 454
Количество пятерок, где 4 различных значений: 994
Количество пятерок, где 5 различных значений: 458

> Теоретические вероятности для каждой из пятерок: 1: 0.0002
2: 0.0256
3: 0.2563
4: 0.5127
5: 0.2051

> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773
> Значение критерия хи-квадрат с 4 степенью свободы: 52.143

[--] Критерий не пройден
```

Рисунок 49 – Результаты выполнения программы для генератора lfsr

Критерий разбиений

```
> Параметр d: 8
> Параметр k: 5 Количество пятерок, где 1 различных значений: 16
Количество пятерок, где 2 различных значений: 85
Количество пятерок, где 3 различных значений: 425
Количество пятерок, где 4 различных значений: 958
Количество пятерок, где 5 различных значений: 516

> Теоретические вероятности для каждой из пятерок: 1: 0.0002
2: 0.0256
3: 0.2563
4: 0.5127
5: 0.2051

> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773
> Значение критерия хи-квадрат с 4 степенью свободы: 677.402

[--] Критерий не пройден
```

Рисунок 50 – Результаты выполнения программы для генератора nfsr

Критерий разбиений

```
> Параметр d: 8
> Параметр k: 5   Количество пятерок, где 1 различных значений: 0
    Количество пятерок, где 2 различных значений: 46
    Количество пятерок, где 3 различных значений: 521
    Количество пятерок, где 4 различных значений: 1042
    Количество пятерок, где 5 различных значений: 391

> Теоретические вероятности для каждой из пятерок:   1: 0.0002
    2: 0.0256
    3: 0.2563
    4: 0.5127
    5: 0.2051

> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773
> Значение критерия хи-квадрат с 4 степенью свободы: 2.233

[+++] Критерий пройден
```

Рисунок 51 – Результаты выполнения программы для генератора mt

Критерий разбиений

```
> Параметр d: 8
> Параметр k: 5   Количество пятерок, где 1 различных значений: 1
    Количество пятерок, где 2 различных значений: 62
    Количество пятерок, где 3 различных значений: 530
    Количество пятерок, где 4 различных значений: 1001
    Количество пятерок, где 5 различных значений: 406

> Теоретические вероятности для каждой из пятерок:   1: 0.0002
    2: 0.0256
    3: 0.2563
    4: 0.5127
    5: 0.2051

> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773
> Значение критерия хи-квадрат с 4 степенью свободы: 4.392

[+++] Критерий пройден
```

Рисунок 52 – Результаты выполнения программы для генератора rc4

Критерий разбиений

```
> Параметр d: 8
> Параметр k: 5   Количество пятерок, где 1 различных значений: 1
    Количество пятерок, где 2 различных значений: 40
    Количество пятерок, где 3 различных значений: 521
    Количество пятерок, где 4 различных значений: 1026
    Количество пятерок, где 5 различных значений: 412

> Теоретические вероятности для каждой из пятерок:   1: 0.0002
    2: 0.0256
    3: 0.2563
    4: 0.5127
    5: 0.2051

> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773
> Значение критерия хи-квадрат с 4 степенью свободы: 3.496

[+++] Критерий пройден
```

Рисунок 53 – Результаты выполнения программы для генератора rsa

```

-----
Критерий разбиений

> Параметр d: 8
> Параметр k: 5 Количество пятерок, где 1 различных значений: 1
Количество пятерок, где 2 различных значений: 42
Количество пятерок, где 3 различных значений: 525
Количество пятерок, где 4 различных значений: 1047
Количество пятерок, где 5 различных значений: 385

> Теоретические вероятности для каждой из пятерок:    1: 0.0002
2: 0.0256
3: 0.2563
4: 0.5127
5: 0.2051

> Критическое значение хи-квадрат для 4 степеней свобод: 9.48773
> Значение критерия хи-квадрат с 4 степенью свободы: 4.856

[***] Критерий пройден
-----
```

Рисунок 54 – Результаты выполнения программы для генератора bbs

Исходный код программы:

```

bool checkPartitionCriterion(vector <double> allElems) {
    int d = 8;
    int k = 5;
    vector <int> resFives(6, 0);
    int elemsCount = allElems.size();
    while (elemsCount % 5 != 0)
        elemsCount = elemsCount - 1;
    int sGr = elemsCount / k;
    for (int i = 0; i < sGr; i++) {
        set <int> dopGr;
        for (int j = i * k; j < (i + 1) * k; j++)
            dopGr.insert(floor(allElems[j] * d));
        resFives[dopGr.size()] += 1;
    }
    double criticalImportance = 9.48773;
    vector <double> fivePs = { 0, 0.0002, 0.0256, 0.2563, 0.5127, 0.2051 };
    double XiSquareDistribution = 0;
    for (int i = 1; i < resFives.size(); i++)
        XiSquareDistribution = XiSquareDistribution + ((resFives[i] - fivePs[i] * sGr) *
    ↵ (resFives[i] - fivePs[i] * sGr)) / (fivePs[i] * sGr);
    XiSquareDistribution = round(XiSquareDistribution * 1000) / 1000;
    cout << "\n > Параметр d: " << d;
    cout << "\n > Параметр k: " << k;
    for (int i = 1; i < resFives.size(); i++) {
        cout << "    Количество пятерок, где " << i << " различных значений: ";
        cout << resFives[i] << "\n";
    }
    cout << "\n > Теоретические вероятности для каждой из пятерок: ";
    for (int i = 1; i < fivePs.size(); i++)
        cout << "    " << i << ":" << fivePs[i] << "\n";
```

```

        cout << "\n > Критическое значение хи-квадрат для " << k - 1 << " степеней свобод: "
→ << criticalImportance;
    cout << "\n > Значение критерия хи-квадрат с " << k - 1 << " степенем свободы: " <<
→ XiSquareDistribution << "\n";
    if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
        cout << "\n[++] Критерий пройден\n";
        return true;
    }
    else {
        cout << "\n[--] Критерий не пройден\n";
        return false;
    }
}

```

1.3.5 Критерий перестановок

Описание критерия:

Последовательность $X^m = (x_1, x_2, \dots, x_m)$ разбивается на n групп по t элементов в каждой:

$$u_j = (x_{jt}, x_{jt+1}, \dots, x_{jt+t-1}), 0 \leq j < n.$$

Элементы в каждой группе можно упорядочивать $t!$ различными способами. Подсчитывается число групп с любым возможным порядком и применяется хи-квадрат критерий с $k = t!$ возможными категориями и вероятностью $\frac{1}{t!}$ для каждой категории. В этом критерии предполагается, что x_s не могут быть равны между собой. В реализации берется $t = 4$.

Приведем результаты выполнения программы для каждого из генераторов:

```

Критерий перестановок

> Параметр t: 4
> Теоретическое количество попаданий в каждую категорию: 105
> Вычисленное распределение по частотам: 134 74 89 128 66 92 90 142 107 64 106 116 116 101 86 126 112 90 132 100 125
132 63 99
> Теоретическое значение вероятности для каждой категории: 0.042
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415
> Значение критерия хи-квадрат с 24 степенем свободы: 119.752

[--] Критерий не пройден

```

Рисунок 55 – Результаты выполнения программы для генератора lc

```
-----  
Критерий перестановок  
  
> Параметр t: 4  
> Теоретическое количество попаданий в каждую категорию: 105  
> Вычисленное распределение по частотам: 90 97 96 94 101 106 109 115 88 105 97 107 96 102 103 129 108 115 105 108 102  
111 111 93  
> Теоретическое значение вероятности для каждой категории: 0.042  
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415  
> Значение критерия хи-квадрат с 24 степенью свободы: 18.99  
  
[+++] Критерий пройден  
-----
```

Рисунок 56 – Результаты выполнения программы для генератора add

```
-----  
Критерий перестановок  
  
> Параметр t: 4  
> Теоретическое количество попаданий в каждую категорию: 105  
> Вычисленное распределение по частотам: 101 117 56 136 73 91 137 97 118 75 95 82 111 97 122 114 96 71 108 133 69 116  
126 149  
> Теоретическое значение вероятности для каждой категории: 0.042  
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415  
> Значение критерия хи-квадрат с 24 степенью свободы: 131.638  
  
[--] Критерий не пройден  
-----
```

Рисунок 57 – Результаты выполнения программы для генератора 5p

```
-----  
Критерий перестановок  
  
> Параметр t: 4  
> Теоретическое количество попаданий в каждую категорию: 105  
> Вычисленное распределение по частотам: 82 142 44 166 29 114 178 90 97 84 73 77 120 48 161 139 106 47 58 142 68 73 1  
56 193  
> Теоретическое значение вероятности для каждой категории: 0.042  
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415  
> Значение критерия хи-квадрат с 24 степенью свободы: 481.057  
  
[--] Критерий не пройден  
-----
```

Рисунок 58 – Результаты выполнения программы для генератора lfsr

```
-----  
Критерий перестановок  
  
> Параметр t: 4  
> Теоретическое количество попаданий в каждую категорию: 105  
> Вычисленное распределение по частотам: 39 117 11 248 7 109 219 89 130 16 57 55 115 7 193 121 89 7 51 203 7 91 228 2  
82  
> Теоретическое значение вероятности для каждой категории: 0.042  
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415  
> Значение критерия хи-квадрат с 24 степенью свободы: 1584.28  
  
[--] Критерий не пройден  
-----
```

Рисунок 59 – Результаты выполнения программы для генератора nfsr

```
-----  
Критерий перестановок  
  
> Параметр t: 4  
> Теоретическое количество попаданий в каждую категорию: 105  
> Вычисленное распределение по частотам: 107 94 96 112 99 102 116 106 106 88 100 123 88 98 101 108 96 103 117 116 102  
89 101 112  
> Теоретическое значение вероятности для каждой категории: 0.042  
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415  
> Значение критерия хи-квадрат с 24 степенью свободы: 20.038  
  
[+++] Критерий пройден  
-----
```

Рисунок 60 – Результаты выполнения программы для генератора mt

```
-----  
Критерий перестановок  
  
> Параметр t: 4  
> Теоретическое количество попаданий в каждую категорию: 105  
> Вычисленное распределение по частотам: 81 88 119 88 103 101 104 92 107 87 91 111 103 98 100 103 111 97 104 101 122  
98 108 92  
> Теоретическое значение вероятности для каждой категории: 0.042  
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415  
> Значение критерия хи-квадрат с 24 степенью свободы: 26.81  
  
[+++] Критерий пройден  
-----
```

Рисунок 61 – Результаты выполнения программы для генератора rc4

```
-----  
Критерий перестановок  
  
> Параметр t: 4  
> Теоретическое количество попаданий в каждую категорию: 105  
> Вычисленное распределение по частотам: 101 91 106 97 99 98 91 96 101 99 104 113 102 113 116 108 103 95 105 109 114  
105 107 109  
> Теоретическое значение вероятности для каждой категории: 0.042  
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415  
> Значение критерия хи-квадрат с 24 степенью свободы: 11.238  
  
[+++] Критерий пройден  
-----
```

Рисунок 62 – Результаты выполнения программы для генератора rsa

```
-----  
Критерий перестановок  
  
> Параметр t: 4  
> Теоретическое количество попаданий в каждую категорию: 105  
> Вычисленное распределение по частотам: 98 117 112 98 85 100 101 100 104 103 93 117 111 114 106 96 97 96 106 95 99 1  
14 121 100  
> Теоретическое значение вероятности для каждой категории: 0.042  
> Критическое значение хи-квадрат для 24 степеней свобод: 36.415  
> Значение критерия хи-квадрат с 24 степенью свободы: 18.029  
  
[+++] Критерий пройден  
-----
```

Рисунок 63 – Результаты выполнения программы для генератора bbs

Исходный код программы:

```

int dopPermutationFunction(vector <double> permVec) {
    int r = 4;
    int fRes = 0;
    while (r > 0) {
        int s = 0;
        double maxV = -10.0;
        for (int i = 0; i < permVec.size(); i++) {
            if (maxV < permVec[i]) {
                maxV = permVec[i];
                s = i;
            }
        }
        s++;
        fRes = r * fRes + s - 1;
        double dopV = permVec[r - 1];
        permVec[r - 1] = permVec[s - 1];
        permVec[s - 1] = dopV;
        permVec.resize(permVec.size() - 1);
        r--;
    }
    return fRes;
}

bool checkPermutationsCriterion(vector <double> allElems) {
    int t = 4;
    int myT = 1;
    for (int i = 1; i <= t; i++)
        myT *= i;
    int sizeinVals = allElems.size();
    while (sizeinVals % 4 != 0)
        sizeinVals = sizeinVals - 1;
    double criticalImportance = 36.41503;
    int nGr = sizeinVals / t;
    vector <int> resCategories(myT, 0);
    for (int i = 0; i < nGr; i++) {
        vector <double> cGr;
        set <double> dopGrc;
        for (int j = i * t; j < (i + 1) * t; j++) {
            cGr.push_back(allElems[j]);
            dopGrc.insert(allElems[j]);
        }
        if (dopGrc.size() != t)
            continue;
        else
            resCategories[dopPermutationFunction(cGr)] += 1;
    }
    double expectedP = round((1.0 / myT) * 1000) / 1000;
    double XiSquareDistribution = 0.0;
    double expectedCategoriesValue = round((expectedP * nGr) * 1000) / 1000;
}

```

```

        cout << "\n > Параметр t: " << t;
        cout << "\n > Теоретическое количество попаданий в каждую категорию: " <<
→ expectedCategoriesValue;
        cout << "\n > Вычисленное распределение по частотам: ";
        for (int i = 0; i < resCategories.size(); i++)
            cout << resCategories[i] << " ";
        cout << "\n > Теоретическое значение вероятности для каждой категории: " <<
→ expectedP;
        for (int i = 0; i < myT; i++)
            XiSquareDistribution = XiSquareDistribution + ((resCategories[i] -
→ expectedCategoriesValue) * (resCategories[i] - expectedCategoriesValue)) /
→ expectedCategoriesValue;
            XiSquareDistribution = round(XiSquareDistribution * 1000) / 1000;
            cout << "\n > Критическое значение хи-квадрат для " << myT << " степеней свобод: "
→ << criticalImportance;
            cout << "\n > Значение критерия хи-квадрат с " << myT << " степенью свободы: " <<
→ XiSquareDistribution << "\n";
            if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
                cout << "\n[++] Критерий пройден\n";
                return true;
            }
            else {
                cout << "\n[--] Критерий не пройден\n";
                return false;
            }
        }
    }
}

```

1.3.6 Критерий монотонности

Описание критерия:

Последовательность можно проверить на предмет равномерности распределения монотонных серий чисел.

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

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

1. «Выбрасываем» элемент последовательности, который следует непосредственно за серией.
2. Если x_j больше x_{j+1} , то начнем следующую серию с x_{j+2} .
3. Мы получаем серии, длины которых независимы и, поэтому, можно использовать критерий хи-квадрат.

Приведем результаты выполнения программы для каждого из генераторов:

Критерий монотонности

```
> Теоретические значения длин серий: 1854 1236 464 124 26 4  
> Вычисленные значения длин серий: 1: 1913 2: 1212 3: 427 4: 125 5: 14 6: 17  
> Критическое значение хи-квадрат для 6 степеней свобод: 12.5916  
> Значение критерия хи-квадрат с 6 степенью свободы: 46.48
```

[---] Критерий не пройден

Рисунок 64 – Результаты выполнения программы для генератора lc

Критерий монотонности

```
> Теоретические значения длин серий: 1828 1219 457 122 25 4  
> Вычисленные значения длин серий: 1: 1797 2: 1224 3: 476 4: 129 5: 27 6: 3  
> Критическое значение хи-квадрат для 6 степеней свобод: 12.5916  
> Значение критерия хи-квадрат с 6 степенью свободы: 2.279
```

[+++] Критерий пройден

Рисунок 65 – Результаты выполнения программы для генератора add

Критерий монотонности

```
> Теоретические значения длин серий: 1797 1198 449 120 25 4 1 0 0 0  
> Вычисленные значения длин серий: 1: 1796 2: 1106 3: 461 4: 178 5: 32 6: 13 7: 3 8: 4 10: 1  
> Критическое значение хи-квадрат для 9 степеней свобод: 16.919  
> Значение критерия хи-квадрат с 9 степенью свободы: 3.99194e+06
```

[---] Критерий не пройден

Рисунок 66 – Результаты выполнения программы для генератора 5р

Критерий монотонности

```
> Теоретические значения длин серий: 1757 1171 439 117 24 4 1 0 0  
> Вычисленные значения длин серий: 1: 1672 2: 1099 3: 478 4: 179 5: 63 6: 16 7: 3 8: 3 9: 1  
> Критическое значение хи-квадрат для 9 степеней свобод: 16.919  
> Значение критерия хи-квадрат с 9 степенью свободы: 371.465
```

[---] Критерий не пройден

Рисунок 67 – Результаты выполнения программы для генератора lfsr

```
Критерий монотонности
```

```
> Теоретические значения длин серий: 1676 1117 419 112 23 4 1 0 0 0 0  
> Вычисленные значения длин серий: 1: 1644 2: 846 3: 444 4: 223 5: 113 6: 54 7: 20 8: 6 11: 1  
> Критическое значение хи-квадрат для 9 степеней свобод: 16.919  
> Значение критерия хи-квадрат с 9 степенью свободы: 4.35479e+07
```

```
[---] Критерий не пройден
```

Рисунок 68 – Результаты выполнения программы для генератора nfsr

```
Критерий монотонности
```

```
> Теоретические значения длин серий: 1830 1220 458 122 25 4 1  
> Вычисленные значения длин серий: 1: 1782 2: 1259 3: 473 4: 118 5: 23 6: 3 7: 2  
> Критическое значение хи-квадрат для 7 степеней свобод: 14.0671  
> Значение критерия хи-квадрат с 7 степенью свободы: 6.745
```

```
[+++] Критерий пройден
```

Рисунок 69 – Результаты выполнения программы для генератора mt

```
Критерий монотонности
```

```
> Теоретические значения длин серий: 1865 1243 466 124 26 4 1  
> Вычисленные значения длин серий: 1: 1923 2: 1222 3: 463 4: 102 5: 16 6: 3 7: 1  
> Критическое значение хи-квадрат для 7 степеней свобод: 14.0671  
> Значение критерия хи-квадрат с 7 степенью свободы: 10.65
```

```
[+++] Критерий пройден
```

Рисунок 70 – Результаты выполнения программы для генератора rc4

```
Критерий монотонности
```

```
> Теоретические значения длин серий: 1835 1223 459 122 25  
> Вычисленные значения длин серий: 1: 1757 2: 1342 3: 418 4: 126 5: 26  
> Критическое значение хи-квадрат для 5 степеней свобод: 11.0705  
> Значение критерия хи-квадрат с 5 степенью свободы: 18.575
```

```
[---] Критерий не пройден
```

Рисунок 71 – Результаты выполнения программы для генератора rsa

```

-----
Критерий монотонности

> Теоретические значения длин серий: 1842 1228 461 123 26 4
> Вычисленные значения длин серий: 1: 1855 2: 1223 3: 451 4: 120 5: 29 6: 6
> Критическое значение хи-квадрат для 6 степеней свобод: 12.5916
> Значение критерия хи-квадрат с 6 степенью свободы: 1.422

[***] Критерий пройден
-----
```

Рисунок 72 – Результаты выполнения программы для генератора bbs

Исходный код программы:

```

bool checkMonotonicityCriterion(vector <double> allElems) {
    vector <pair <int, int>> categor;
    vector <double> ojid;
    double XiSquareDistribution = 0.0;
    int posl = 1, i = 1;
    while (i < allElems.size() - 1) {
        if (allElems[i] < allElems[i + 1])
            posl += 1;
        else {
            int num = -1;
            if (categor.size() != 0) {
                for (int j = 0; j < categor.size(); j++) {
                    if (categor[j].first == posl) {
                        num = j;
                        break;
                    }
                }
                if (num == -1)
                    categor.push_back(make_pair(posl, 1));
                else
                    categor[num].second += 1;
            }
            else
                categor.push_back(make_pair(posl, 1));
            posl = 1;
            i = i + 1;
        }
        i += 1;
    }
    int sumS = 0;
    for (int j = 0; j < categor.size(); j++)
        sumS = sumS + categor[j].second;
    int num_categs = categor.size();
    double criticalImportance = InterVector[num_categs - 1];
    int maximal = -1;
    for (int j = 0; j < categor.size(); j++)
        if (categor[j].first > maximal)
```

```

maximal = categor[j].first;
long long fact = 1;
for (int j = 1; j < maximal + 1; j++) {
    fact *= j;
    ojid.push_back(1.0 / fact - 1.0 / (fact * (j + 1)));
}
for (int j = 0; j < num_categs; j++)
ojid[j] = (sumS * ojid[j]);
for (int j = 0; j < num_categs; j++)
XiSquareDistribution = XiSquareDistribution + round((((categor[j].second -
→ ojid[categor[j].first - 1]) * (categor[j].second - ojid[categor[j].first - 1])) /
→ ojid[categor[j].first - 1]) * 1000) / 1000;
cout << "\n > Теоретические значения длин серий: ";
for (int j = 0; j < ojid.size(); j++)
cout << round(ojid[j]) << " ";
cout << "\n > Вычисленные значения длин серий: ";
set<pair<int, int>> need;
for (int j = 0; j < categor.size(); j++)
need.insert(categor[j]);
for (pair<int, int> a : need)
cout << a.first << ":" << a.second << " ";
cout << "\n > Критическое значение хи-квадрат для " << num_categs << " степеней
→ свобод: " << criticalImportance;
cout << "\n > Значение критерия хи-квадрат с " << num_categs << " степенью свободы:
→ " << XiSquareDistribution << "\n";
if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
    cout << "\n[++] Критерий пройден\n";
    return true;
}
else {
    cout << "\n[--] Критерий не пройден\n";
    return false;
}
}

```

1.3.7 Критерий конфликтов

Описание критерия:

Предположим, нам нужно оценить последовательность случайных чисел, в которой число величин в последовательности намного меньше числа категорий. 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 $m = 2^{20}$, а $n = 2^{14}$. В среднем, число урн, приходящихся на один шар – 64. Вероятность того, что в конкретную урну попадет ровно k шаров, равна

$$p_k = \binom{n}{k} m^{-k} (1 - m^{-1})^{n-k},$$

отсюда, среднее число конфликтов в урне вычисляется по формуле

$$\sum_{k \leq 1} (k - 1)p_k = \sum_{k \leq 1} kp_k - \sum_{k \leq 1} p_k = \frac{n}{m} - 1 + p_0.$$

Так как $p_0 = (1 - m^{-1})^n = 1 - nm^{-1} + \binom{n}{2} m^{-2} - \dots$ – маленькое число, получим, что общее среднее число конфликтов во всех m урнах намного меньше

$$\frac{n^2}{2m} = 128.$$

Приведем результаты выполнения программы для каждого из генераторов:

```
Критерий конфликтов

> Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 148
> Приведем случай, которая параметры подходят:
    Размерность вектора Vj: 10
    Количество векторов: 1000
    Значение m: 19000
    Множитель для m: 19
    Параметр нормирования d: 4
    Количество возникших конфликтов: 21
    Таблица процентных точек: (14, 0.007) (17, 0.039) (21, 0.19) (25, 0.485) (28, 0.714) (33, 0.936) (37, 0.988)

[***] Критерий пройден
```

Рисунок 73 – Результаты выполнения программы для генератора lc

```
Критерий конфликтов

> Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 106
> Приведем случай, которая параметры подходят:
    Размерность вектора Vj: 8
    Количество векторов: 1250
    Значение m: 66250
    Множитель для m: 53
    Параметр нормирования d: 4
    Количество возникших конфликтов: 8
    Таблица процентных точек: (4, 0.009) (5, 0.023) (8, 0.172) (11, 0.494) (13, 0.713) (17, 0.95) (19, 0.984)

[***] Критерий пройден
```

Рисунок 74 – Результаты выполнения программы для генератора add

```

Критерий конфликтов

> Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 378
> Приведем случай, которая параметры подходят:
    Размерность вектора Vj: 8
    Количество векторов: 1250
    Значение m: 27500
    Множитель для m: 22
    Параметр нормирования d: 4
    Количество возникших конфликтов: 24
    Таблица процентных точек: (16, 0.009) (19, 0.044) (23, 0.194) (27, 0.476) (30, 0.697) (36, 0.947) (40, 0.99)

[+++] Критерий пройден

```

Рисунок 75 – Результаты выполнения программы для генератора 5р

```

Критерий конфликтов

> Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 211
> Приведем случай, которая параметры подходят:
    Размерность вектора Vj: 8
    Количество векторов: 1250
    Значение m: 20000
    Множитель для m: 16
    Параметр нормирования d: 5
    Количество возникших конфликтов: 33
    Таблица процентных точек: (24, 0.007) (28, 0.046) (33, 0.215) (37, 0.46) (41, 0.715) (47, 0.937) (52, 0.99)

[+++] Критерий пройден

```

Рисунок 76 – Результаты выполнения программы для генератора lfsr

```

Критерий конфликтов

> Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 333
> Приведем случай, которая параметры подходят:
    Размерность вектора Vj: 10
    Количество векторов: 1000
    Значение m: 19000
    Множитель для m: 19
    Параметр нормирования d: 8
    Количество возникших конфликтов: 21
    Таблица процентных точек: (14, 0.007) (17, 0.039) (21, 0.19) (25, 0.485) (28, 0.714) (33, 0.936) (37, 0.988)

[+++] Критерий пройден

```

Рисунок 77 – Результаты выполнения программы для генератора nfsr

```

Критерий конфликтов

> Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 115
> Приведем случай, которая параметры подходят:
    Размерность вектора Vj: 8
    Количество векторов: 1250
    Значение m: 45000
    Множитель для m: 36
    Параметр нормирования d: 4
    Количество возникших конфликтов: 13
    Таблица процентных точек: (7, 0.004) (10, 0.042) (13, 0.184) (16, 0.448) (19, 0.724) (23, 0.934) (26, 0.985)

[+++] Критерий пройден

```

Рисунок 78 – Результаты выполнения программы для генератора mt

```

-----
| Критерий конфликтов
|
| > Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 130
| > Приведем случай, которая параметры подходят:
|   Размерность вектора Vj: 8
|   Количество векторов: 1250
|   Значение m: 48750
|   Множитель для m: 39
|   Параметр нормирования d: 4
|   Количество возникших конфликтов: 12
|   Таблица процентных точек: (7, 0.01) (9, 0.043) (12, 0.197) (15, 0.477) (17, 0.673) (22, 0.949) (25, 0.989)
|
| [+++] Критерий пройден
-----

```

Рисунок 79 – Результаты выполнения программы для генератора rc4

```

-----
| Критерий конфликтов
|
| > Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 87
| > Приведем случай, которая параметры подходят:
|   Размерность вектора Vj: 8
|   Количество векторов: 1250
|   Значение m: 45000
|   Множитель для m: 36
|   Параметр нормирования d: 4
|   Количество возникших конфликтов: 13
|   Таблица процентных точек: (7, 0.004) (10, 0.042) (13, 0.184) (16, 0.448) (19, 0.724) (23, 0.934) (26, 0.985)
|
| [+++] Критерий пройден
-----

```

Рисунок 80 – Результаты выполнения программы для генератора rsa

```

-----
| Критерий конфликтов
|
| > Число параметров, при которых представленная последовательность удовлетворяет критерию конфликтов: 99
| > Приведем случай, которая параметры подходят:
|   Размерность вектора Vj: 8
|   Количество векторов: 1250
|   Значение m: 40000
|   Множитель для m: 32
|   Параметр нормирования d: 4
|   Количество возникших конфликтов: 15
|   Таблица процентных точек: (9, 0.007) (12, 0.049) (15, 0.19) (18, 0.439) (21, 0.703) (26, 0.947) (29, 0.987)
|
| [+++] Критерий пройден
-----

```

Рисунок 81 – Результаты выполнения программы для генератора bbs

Исходный код программы:

```

vector<pair<int, double>> ConflictCriterionPercentPoints(int m, int n) {
    vector<double> auxiliaryTableT = { 0.01, 0.05, 0.25, 0.50, 0.75, 0.95, 0.99, 1.0 };
    vector<double> auxiliaryA(n + 1, 0.0);
    vector<pair<int, double>> conflictsResP;
    auxiliaryA[1] = 1.0;
    int j0 = 1;
    int j1 = 1;
    for (int i = 0; i < n - 1; i++) {
        if (auxiliaryA[i] <= auxiliaryTableT[j0]) {
            conflictsResP.push_back({ i, auxiliaryTableT[j0] });
            j0++;
        }
        if (auxiliaryA[i] > auxiliaryTableT[j1]) {
            conflictsResP.push_back({ i, auxiliaryTableT[j1] });
            j1++;
        }
    }
}

```

```

        j1++;
        for (int j = j1; j > j0 - 1; j--) {
            double jm = j / (m * 1.0);
            auxiliaryA[j] = jm * auxiliaryA[j] + (1.0 + 1.0 / m - jm) *
    ↵ auxiliaryA[j - 1];
            if (auxiliaryA[j] < 1e-20) {
                auxiliaryA[j] = 0.0;
                if (j == j1) {
                    j1--;
                    continue;
                }
                if (j == j0)
                    j0++;
            }
        }
    }

    int t = 0;
    int j = j0 - 1;
    double p = 0.0;
    while (t != auxiliaryTableT.size() - 1) {
        while (p <= auxiliaryTableT[t]) {
            j++;
            p += auxiliaryA[j];
        }
        conflictsResP.push_back(make_pair(n - j - 1, round((1 - p) * 1000) / 1000));
        t++;
    }
    reverse(conflictsResP.begin(), conflictsResP.end());
    return conflictsResP;
}

bool checkConflictCriterion(vector <double> allElems) {
    srand((unsigned int)time(0));
    double eps = 1e-20;
    vector <vector <pair<int, double>>> appropriatePercentPoints;
    vector <vector <int>> appropriatePointsTable;
    for (int bigSize = 8; bigSize < 21; bigSize++) {
        int nParameter = allElems.size() / bigSize;
        for (int dParameter = 2; dParameter < 9; dParameter++) {
            int j = 0;
            vector <int> normSeq;
            while (j < allElems.size()) {
                if (allElems[j] == 1.0)
                    allElems[j] = 0.965;
                normSeq.push_back((floor(allElems[j] * dParameter)));
                j++;
            }
        }
    }
}

```

```

        set <vector <int>> words;
        int nDimension = 0;
        for (int jI = 0; jI < nParameter; jI++) {
            vector <int> dopSliced;
            for (int i = jI * bigSize; i < (jI + 1) * bigSize; i++) {
                dopSliced.push_back(normSeq[i]);
            }
            auto search = words.find(dopSliced);
            if (search == words.end())
                words.insert(dopSliced);
            else
                nDimension += 1;
        }
        for (int mParameter = 16; mParameter < 129; mParameter++) {
            int mV = nParameter * mParameter;
            vector <pair <int, double>> confs_et_probs =
    ↵ ConflictCriterionPercentPoints(mV, nParameter);
            if (nDimension == 0 || confs_et_probs[0].first == -1 ||
    ↵ confs_et_probs[0].first == 0)
                continue;
            if (confs_et_probs[2].first <= nDimension && nDimension <=
    ↵ confs_et_probs[confs_et_probs.size() - 2].first) {
                appropriatePercentPoints.push_back(confs_et_probs);
                vector <int> prom = { nDimension, bigSize,
    ↵ nParameter, dParameter, mParameter, mV };
                appropriatePointsTable.push_back(prom);
            }
        }
    }
    int appropriatePercentPointsCount = appropriatePercentPoints.size();
    set <int> setRand;
    for (int j = 0; j < 5; j++) {

        int randInt = (double)(rand()) / RAND_MAX * appropriatePercentPointsCount -
    ↵ 1;
        auto search1 = setRand.find(randInt);
        if (search1 == setRand.end())
            setRand.insert(randInt);
        else {
            while (search1 == setRand.end()) {
                randInt = (double)(rand()) / RAND_MAX *
    ↵ appropriatePercentPointsCount - 1;
                search1 = setRand.find(randInt);
            }
        }
    }
}

```

```

        cout << "\n > Число параметров, при которых представленная последовательность
→ удовлетворяет критерию конфликтов: " << appropriatePercentPointsCount;
        cout << "\n > Приведем случай, которая параметры подходят:" ;
        cout << "\n      Размерность вектора Vj: " << appropriatePointsTable[0][1];
        cout << "\n      Количество векторов: " << appropriatePointsTable[0][2];
        cout << "\n      Значение m: " << appropriatePointsTable[0][5];
        cout << "\n      Множитель для m: " << appropriatePointsTable[0][4];
        cout << "\n      Параметр нормирования d: " << appropriatePointsTable[0][3];
        cout << "\n      Количество возникших конфликтов: " << appropriatePointsTable[0][0];
        cout << "\n      Таблица процентных точек: ";
        for (int k = 0; k < appropriatePercentPoints[0].size(); k++) {
            cout << "(" << appropriatePercentPoints[0][k].first << ", " <<
→ appropriatePercentPoints[0][k].second << ") ";
        }
        cout << "\n";
        if (appropriatePercentPointsCount > 0) {
            cout << "\n[+] Критерий пройден\n";
            return true;
        }
        else {
            cout << "\n[-] Критерий не пройден\n";
            return false;
        }
    }
}

```

1.3.8 Результаты проверки всех критериев для последовательностей, сгенерированных каждым генератором

В следующей таблице представлены результаты проверки для всех сгенерированных последовательностей по каждому из рассмотренных выше критериев. 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

```
#define _USE_MATH_DEFINES

#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <iomanip>
#include <bitset>
#include <sstream>
#include <deque>
#include <cmath>
#include <set>

using namespace std;

string findInStr(string const& str, int n) {
    if (str.length() < n) {
        return str;
    }
    return str.substr(0, n);
}

void helpFunc() {
    cout << "\n\n/f:<имя_файла> - имя файла с входной последовательностью";
    cout << "\n\n/h - информация о допустимых параметрах командной строки программы.\n";
}

double calculateExpValue(vector <double> allElems) {
    double elemsCount = allElems.size();
    double elemsSum = 0;
    for (int i = 0; i < allElems.size(); i++) {
        elemsSum += allElems[i];
    }
    return elemsSum / elemsCount;
}

double calculateStandartDeviation(vector <double> allElems, double expValue) {
    double elemsCount = allElems.size();
    double elemsSum = 0;
    for (int i = 0; i < allElems.size(); i++) {
        elemsSum = elemsSum + (allElems[i] - expValue) * (allElems[i] - expValue);
    }
    return sqrt(elemsSum / elemsCount);
}
```

```

double calculateExpValueError(double expValue) {
    return abs(expValue - 0.5) / expValue;
}

double calculateStandartDeviationError(double standartDeviation) {
    double sqrt1_12 = sqrt(1.0 / 12);
    return abs(standartDeviation - sqrt1_12) / standartDeviation;
}

bool checkXiSquareCriterion(vector <double> allElems) {
    double maxElem = -1.0, minElem = 1.0;
    int elemsCount = allElems.size();
    for (int i = 0; i < elemsCount; i++) {
        double checkElem = allElems[i];
        if (checkElem > maxElem)
            maxElem = checkElem;
        if (checkElem < minElem)
            minElem = checkElem;
    }
    vector <pair <double, double>> allIntervals;
    int intervalsCount = ceil(1 + 1.4 * log(elemsCount));
    double IntS = (round(((maxElem - minElem) / intervalsCount) * 10000)) / 10000;
    for (int i = 0; i < intervalsCount; i++) {
        double a_i = minElem + IntS * i;
        double b_i = minElem + IntS * (i + 1);
        allIntervals.push_back(make_pair(a_i, b_i));
    }
    double expectedNumOfHits = round((elemsCount / intervalsCount) * 1000) / 1000;
    vector <int> realDistribution(intervalsCount, 0);
    for (int i = 0; i < elemsCount; i++)
        for (int j = 0; j < intervalsCount; j++)
            if (allIntervals[j].first <= allElems[i] && allElems[i] < allIntervals[j].second) {
                realDistribution[j]++;
                break;
            }
    double XiSquareDistribution = 0;
    for (int i = 0; i < intervalsCount; i++)
        XiSquareDistribution = XiSquareDistribution + ((realDistribution[i] -
→ expectedNumOfHits) * (realDistribution[i] - expectedNumOfHits)) / expectedNumOfHits;
    XiSquareDistribution = round(XiSquareDistribution * 10000) / 10000;
    double criticalImportance = 22.36203;
    cout << "\n > Количество интервалов: " << intervalsCount;
    int k = intervalsCount - 1;
    cout << "\n > Ожидаемое число попаданий в интервалы: " << expectedNumOfHits;
    cout << "\n > Число попаданий в каждый из интервалов: ";
    for (int i = 0; i < intervalsCount; i++)

```

```

        cout << i + 1 << ":" << realDistribution[i] << endl;
        cout << "\n > Критическое значение хи-квадрат для " << k << " степеней свободы: " <<
→ criticalImportance;
        cout << "\n > Значение критерия хи-квадрат с " << k << " степенью свободы: " <<
→ XiSquareDistribution << "\n";
        if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
            cout << "\n[++] Критерий пройден\n";
            return true;
        }
        else {
            cout << "\n[--] Критерий не пройден\n";
            return false;
        }
    }

bool checkSeriesCriterion(vector <double> allElems) {
    int d = 4;
    vector <vector <int>> nPairs(d);
    for (int i = 0; i < d; i++)
        nPairs[i].resize(d, 0);
    int k = d * d;
    vector <pair <int, int>> pairCategories;
    int elemsCount = allElems.size();
    if (elemsCount % 2 != 0)
        elemsCount--;
    for (int j = 0; j < elemsCount; j+=2)
    {
        double leftI = floor(allElems[j] * d);
        double rightI = floor(allElems[j + 1] * d);
        pairCategories.push_back(make_pair(leftI, rightI));
    }
    double criticalImportance = 26.29623;
    double expectedNumOfHits = round((elemsCount / (2 * k)) * 1000) / 1000;
    for (int i = 0; i < pairCategories.size(); i++) {
        nPairs[pairCategories[i].first][pairCategories[i].second] += 1;
    }
    double XiSquareDistribution = 0;
    for (int i = 0; i < nPairs.size(); i++)
        for (int x = 0; x < nPairs[i].size(); x++)
            XiSquareDistribution += ((nPairs[i][x] - expectedNumOfHits) * (nPairs[i][x] -
→ expectedNumOfHits)) / expectedNumOfHits;
    XiSquareDistribution = round(XiSquareDistribution * 1000) / 1000;
    cout << "\n > Параметр d: " << d;
    cout << "\n > Ожидаемое количество чисел в каждой категории: " << expectedNumOfHits;
    cout << "\n > Количество чисел в каждой категории: ";
    int chCount = 1;
    for (int i = 0; i < nPairs.size(); i++) {
        for (int x = 0; x < nPairs[i].size(); x++) {

```

```

        chCount++;
        cout << "    " << chCount << ":" " << nPairs[i][x] << "\n";
    }
    chCount++;
}
cout << "\n > Критическое значение хи-квадрат для " << k << " степеней свобод: " <<
criticalImportance;
cout << "\n > Значение критерия хи-квадрат с " << k << " степенью свободы: " <<
XiSquareDistribution << "\n";
if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
    cout << "\n[++] Критерий пройден\n";
    return true;
}
else {
    cout << "\n[--] Критерий не пройден\n";
    return false;
}
}

double powDouble(double c, int stepen) {
    if (stepen == 0)
        return 1.0;
    if (stepen == 1)
        return c;
    double res = c;
    for (int i = 2; i <= stepen; i++)
        res = res * c;
    return res;
}

vector < double > InterVector = { 3.84146 , 5.99146, 7.81473, 9.48773, 11.07050, 12.59159,
14.06714, 15.50731, 16.91898, 18.30704, 19.67514, 21.02607, 22.36203, 23.68479,
24.99579, 26.29623, 27.58711, 28.86930, 30.14353, 31.41043, 32.67057, 33.92444,
35.17246, 36.41503, 37.65248, 38.88514, 40.11327, 41.33714, 42.55697, 43.77297 };

bool checkIntervalsCriterion(vector <double> allElems) {
    int t = 10, n = 1000;
    srand((unsigned int)time(0));
    double alpha = (double)(rand()) / RAND_MAX, beta = (double)(rand()) / RAND_MAX;
    while (0.2 >= abs(alpha - beta) || abs(alpha - beta) >= 0.4) {
        alpha = (double)(rand()) / RAND_MAX;
        beta = (double)(rand()) / RAND_MAX;
    }
    if (beta < alpha) {
        double dopD = alpha;
        alpha = beta;
        beta = dopD;
    }
}

```

```

alpha = round(alpha * 1000) / 1000;
beta = round(beta * 1000) / 1000;
double p = beta - alpha;
int s = 0;
vector <int> count, resCount;
double XiSquareDistribution = 100000000000.0;
double criticalImportance = 100000000000.0;
int resT = 100000000000;
int intervalsCount = 100000000000;
vector <double> resPs;
s = 0;
count.resize(t + 1, 0);
int r = 0;
for (int j = 0; j < allElems.size(); j++) {
    if (alpha <= allElems[j] && allElems[j] < beta) {
        if (r >= t)
            count[t]++;
        else
            count[r]++;
        s++;
        if (s < n)
            r = 0;
        else
            break;
    }
    else
        r++;
}
vector <double> probs;
for (int j = 0; j < t; j++)
probs.push_back(round((p * (powDouble(1.0 - p, j))) * 1000) / 1000);
double XiSch = 0.0;
probs.push_back(powDouble(1.0 - p, t));
double criticalImpDop = InterVector[t];
int qN = 0;
for (int j = 0; j < count.size(); j++)
qN += count[j];
for (int j = 0; j < probs.size(); j++) {
    double est = qN * probs[j];
    if (est == 0.0)
        break;
    XiSch += ((count[j] - est) * (count[j] - est)) / est;
}
if (XiSch < XiSquareDistribution) {
    XiSquareDistribution = XiSch;
    criticalImportance = criticalImpDop;
    resT = t;
    intervalsCount = n;
}

```

```

        resPs = probs;
        resCount = count;
    }

    int k = resT + 1;
    cout << "\n > Максимальная длина интервала (t): " << resT;
    cout << "\n > Количество интервалов (n): " << intervalsCount;
    cout << "\n > Границы: alpha = " << alpha << " beta = " << beta;
    cout << "\n > Пересчитанные вероятности Pr и Pt: \n";
    cout << "   ";
    for (int j = 0; j < resPs.size(); j++)
        cout << resPs[j] << " ";
    cout << "\n > Подсчитанные значения интервалов длиной 0, 1, ..., t - 1 и >= t (t = "
    << resT << "):\n";
    cout << "   ";
    for (int j = 0; j < resCount.size(); j++)
        cout << resCount[j] << " ";
    cout << "\n > Критическое значение хи-квадрат для " << k << " степеней свобод: " <<
    criticalImportance;
    cout << "\n > Значение критерия хи-квадрат с " << k << " степенью свободы: " <<
    round(XiSquareDistribution * 1000) / 1000 << "\n";
    if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
        cout << "\n[++] Критерий пройден\n";
        return true;
    }
    else {
        cout << "\n[--] Критерий не пройден\n";
        return false;
    }
}

bool checkPartitionCriterion(vector <double> allElems) {
    int d = 8;
    int k = 5;
    vector <int> resFives(6, 0);
    int elemsCount = allElems.size();
    while (elemsCount % 5 != 0)
        elemsCount = elemsCount - 1;
    int sGr = elemsCount / k;
    for (int i = 0; i < sGr; i++) {
        set <int> dopGr;
        for (int j = i * k; j < (i + 1) * k; j++)
            dopGr.insert(floor(allElems[j] * d));
        resFives[dopGr.size()] += 1;
    }
    double criticalImportance = 9.48773;
    vector <double> fivePs = { 0, 0.0002, 0.0256, 0.2563, 0.5127, 0.2051 };
    double XiSquareDistribution = 0;
    for (int i = 1; i < resFives.size(); i++)

```

```

        XiSquareDistribution = XiSquareDistribution + ((resFives[i] - fivePs[i] * sGr) *
→ (resFives[i] - fivePs[i] * sGr)) / (fivePs[i] * sGr);
        XiSquareDistribution = round(XiSquareDistribution * 1000) / 1000;
        cout << "\n > Параметр d: " << d;
        cout << "\n > Параметр k: " << k;
        for (int i = 1; i < resFives.size(); i++) {
            cout << "    Количество пятерок, где " << i << " различных значений: ";
            cout << resFives[i] << "\n";
        }
        cout << "\n > Теоретические вероятности для каждой из пятерок: ";
        for (int i = 1; i < fivePs.size(); i++)
            cout << "    " << i << ":" << fivePs[i] << "\n";
        cout << "\n > Критическое значение хи-квадрат для " << k - 1 << " степеней свобод: "
→ << criticalImportance;
        cout << "\n > Значение критерия хи-квадрат с " << k - 1 << " степенью свободы: " <<
→ XiSquareDistribution << "\n";
        if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
            cout << "\n[++] Критерий пройден\n";
            return true;
        }
        else {
            cout << "\n[--] Критерий не пройден\n";
            return false;
        }
    }

int dopPermutationFunction(vector <double> permVec) {
    int r = 4;
    int fRes = 0;
    while (r > 0) {
        int s = 0;
        double maxV = -10.0;
        for (int i = 0; i < permVec.size(); i++)
            if (maxV < permVec[i])
                maxV = permVec[i];
                s = i;
        }
        s++;
        fRes = r * fRes + s - 1;
        double dopV = permVec[r - 1];
        permVec[r - 1] = permVec[s - 1];
        permVec[s - 1] = dopV;
        permVec.resize(permVec.size() - 1);
        r--;
    }
    return fRes;
}

```

```

bool checkPermutationsCriterion(vector <double> allElems) {
    int t = 4;
    int myT = 1;
    for (int i = 1; i <= t; i++)
        myT *= i;
    int sizeinVals = allElems.size();
    while (sizeinVals % 4 != 0)
        sizeinVals = sizeinVals - 1;
    double criticalImportance = 36.41503;
    int nGr = sizeinVals / t;
    vector <int> resCategories(myT, 0);
    for (int i = 0; i < nGr; i++) {
        vector <double> cGr;
        set <double> dopGrc;
        for (int j = i * t; j < (i + 1) * t; j++) {
            cGr.push_back(allElems[j]);
            dopGrc.insert(allElems[j]);
        }
        if (dopGrc.size() != t)
            continue;
        else
            resCategories[dopPermutationFunction(cGr)] += 1;
    }
    double expectedP = round((1.0 / myT) * 1000) / 1000;
    double XiSquareDistribution = 0.0;
    double expectedCategoriesValue = round((expectedP * nGr) * 1000) / 1000;
    cout << "\n > Параметр t: " << t;
    cout << "\n > Теоретическое количество попаданий в каждую категорию: " <<
→ expectedCategoriesValue;
    cout << "\n > Вычисленное распределение по частотам: ";
    for (int i = 0; i < resCategories.size(); i++)
        cout << resCategories[i] << " ";
    cout << "\n > Теоретическое значение вероятности для каждой категории: " <<
→ expectedP;
    for (int i = 0; i < myT; i++)
        XiSquareDistribution = XiSquareDistribution + ((resCategories[i] -
→ expectedCategoriesValue) * (resCategories[i] - expectedCategoriesValue)) /
→ expectedCategoriesValue;
    XiSquareDistribution = round(XiSquareDistribution * 1000) / 1000;
    cout << "\n > Критическое значение хи-квадрат для " << myT << " степеней свобод: " <<
→ criticalImportance;
    cout << "\n > Значение критерия хи-квадрат с " << myT << " степенью свободы: " <<
→ XiSquareDistribution << "\n";
    if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
        cout << "\n[++] Критерий пройден\n";
        return true;
    }
    else {

```

```

        cout << "\n[---] Критерий не пройден\n";
        return false;
    }

}

bool checkMonotonicityCriterion(vector <double> allElems) {
    vector <pair <int, int>> categor;
    vector <double> ojid;
    double XiSquareDistribution = 0.0;
    int posl = 1, i = 1;
    while (i < allElems.size() - 1) {
        if (allElems[i] < allElems[i + 1])
            posl += 1;
        else {
            int num = -1;
            if (categor.size() != 0) {
                for (int j = 0; j < categor.size(); j++) {
                    if (categor[j].first == posl) {
                        num = j;
                        break;
                    }
                }
                if (num == -1)
                    categor.push_back(make_pair(posl, 1));
                else
                    categor[num].second += 1;
            }
            else
                categor.push_back(make_pair(posl, 1));
            posl = 1;
            i = i + 1;
        }
        i += 1;
    }
    int sumS = 0;
    for (int j = 0; j < categor.size(); j++)
        sumS = sumS + categor[j].second;
    int num_categs = categor.size();
    double criticalImportance = InterVector[num_categs - 1];
    int maximal = -1;
    for (int j = 0; j < categor.size(); j++)
        if (categor[j].first > maximal)
            maximal = categor[j].first;
    long long fact = 1;
    for (int j = 1; j < maximal + 1; j++) {
        fact *= j;
        ojid.push_back(1.0 / fact - 1.0 / (fact * (j + 1)));
    }
}

```

```

        for (int j = 0; j < num_categs; j++)
            ojid[j] = (sumS * ojid[j]);
        for (int j = 0; j < num_categs; j++)
            XiSquareDistribution = XiSquareDistribution + round(((categor[j].second -
← ojid[categor[j].first - 1]) * (categor[j].second - ojid[categor[j].first - 1])) /
← ojid[categor[j].first - 1]) * 1000) / 1000;
            cout << "\n > Теоретические значения длин серий: ";
            for (int j = 0; j < ojid.size(); j++)
                cout << round(ojid[j]) << " ";
            cout << "\n > Вычисленные значения длин серий: ";
            set<pair<int, int>> need;
            for (int j = 0; j < categor.size(); j++)
                need.insert(categor[j]);
            for (pair<int, int> a : need)
                cout << a.first << ":" << a.second << " ";
            cout << "\n > Критическое значение хи-квадрат для " << num_categs << " степеней
← свобод: " << criticalImportance;
            cout << "\n > Значение критерия хи-квадрат с " << num_categs << " степенью свободы:
← " << XiSquareDistribution << "\n";
            if (0 < XiSquareDistribution && XiSquareDistribution < criticalImportance) {
                cout << "\n[++] Критерий пройден\n";
                return true;
            }
            else {
                cout << "\n[--] Критерий не пройден\n";
                return false;
            }
        }

vector<pair<int, double>> ConflictCriterionPercentPoints(int m, int n) {
    vector<double> auxiliaryTableT = { 0.01, 0.05, 0.25, 0.50, 0.75, 0.95, 0.99, 1.0 };
    vector<double> auxiliaryA(n + 1, 0.0);
    vector<pair<int, double>> conflictsResP;
    auxiliaryA[1] = 1.0;
    int j0 = 1;
    int j1 = 1;
    for (int i = 0; i < n - 1; i++) {
        j1++;
        for (int j = j1; j > j0 - 1; j--) {
            double jm = j / (m * 1.0);
            auxiliaryA[j] = jm * auxiliaryA[j] + (1.0 + 1.0 / m - jm) *
← auxiliaryA[j - 1];
            if (auxiliaryA[j] < 1e-20) {
                auxiliaryA[j] = 0.0;
                if (j == j1) {
                    j1--;
                    continue;
                }
            }
        }
    }
}

```

```

        }
        if (j == j0)
            j0++;
    }
}

int t = 0;
int j = j0 - 1;
double p = 0.0;
while (t != auxiliaryTableT.size() - 1) {
    while (p <= auxiliaryTableT[t]) {
        j++;
        p += auxiliaryA[j];
    }
    conflictsResP.push_back(make_pair(n - j - 1, round((1 - p) * 1000) / 1000));
    t++;
}
reverse(conflictsResP.begin(), conflictsResP.end());
return conflictsResP;
}

bool checkConflictCriterion(vector <double> allElems) {
    srand((unsigned int)time(0));
    double eps = 1e-20;
    vector <vector <pair<int, double>>> appropriatePercentPoints;
    vector <vector <int>> appropriatePointsTable;
    for (int bigSize = 8; bigSize < 21; bigSize++) {
        int nParameter = allElems.size() / bigSize;
        for (int dParameter = 2; dParameter < 9; dParameter++) {
            int j = 0;
            vector <int> normSeq;
            while (j < allElems.size()) {
                if (allElems[j] == 1.0)
                    allElems[j] = 0.965;
                normSeq.push_back(floor(allElems[j] * dParameter));
                j++;
            }
            set <vector <int>> words;
            int nDimension = 0;
            for (int jI = 0; jI < nParameter; jI++) {
                vector <int> dopSliced;
                for (int i = jI * bigSize; i < (jI + 1) * bigSize; i++) {
                    dopSliced.push_back(normSeq[i]);
                }
                auto search = words.find(dopSliced);
                if (search == words.end())
                    words.insert(dopSliced);
            }
        }
    }
}
```

```

                else
                    nDimension += 1;
            }
            for (int mParameter = 16; mParameter < 129; mParameter++) {
                int mV = nParameter * mParameter;
                vector <pair <int, double>> confs_et_probs =
        ↵ ConflictCriterionPercentPoints(mV, nParameter);
                if (nDimension == 0 || confs_et_probs[0].first == -1 ||
        ↵ confs_et_probs[0].first == 0)
                    continue;
                if (confs_et_probs[2].first <= nDimension && nDimension <=
        ↵ confs_et_probs[confs_et_probs.size() - 2].first) {
                    appropriatePercentPoints.push_back(confs_et_probs);
                    vector <int> prom = { nDimension, bigSize,
        ↵ nParameter, dParameter, mParameter, mV };
                    appropriatePointsTable.push_back(prom);
                }
            }
        }
        int appropriatePercentPointsCount = appropriatePercentPoints.size();
        set <int> setRand;
        for (int j = 0; j < 5; j++) {

            int randInt = (double)(rand()) / RAND_MAX * appropriatePercentPointsCount -
        ↵ 1;
            auto search1 = setRand.find(randInt);
            if (search1 == setRand.end())
                setRand.insert(randInt);
            else {
                while (search1 == setRand.end()) {
                    randInt = (double)(rand()) / RAND_MAX *
        ↵ appropriatePercentPointsCount - 1;
                    search1 = setRand.find(randInt);
                }
            }
        }
        cout << "\n > Число параметров, при которых представленная последовательность
        ↵ удовлетворяет критерию конфликтов: " << appropriatePercentPointsCount;
        cout << "\n > Приведем случай, которая параметры подходят:";
        cout << "\n      Размерность вектора Vj: " << appropriatePointsTable[0][1];
        cout << "\n      Количество векторов: " << appropriatePointsTable[0][2];
        cout << "\n      Значение m: " << appropriatePointsTable[0][5];
        cout << "\n      Множитель для m: " << appropriatePointsTable[0][4];
        cout << "\n      Параметр нормирования d: " << appropriatePointsTable[0][3];
        cout << "\n      Количество возникших конфликтов: " << appropriatePointsTable[0][0];
        cout << "\n      Таблица процентных точек: ";
        for (int k = 0; k < appropriatePercentPoints[0].size(); k++) {

```

```

                cout << "(" << appropriatePercentPoints[0][k].first << ", " <<
→  appropriatePercentPoints[0][k].second << ")" " ;
}
cout << "\n";
if (appropriatePercentPointsCount > 0) {
    cout << "\n[+] Критерий пройден\n";
    return true;
}
else {
    cout << "\n[--] Критерий не пройден\n";
    return false;
}
}

vector <bool> checkCriterions(string inputFileName, bool& errorMessage) {
    vector <bool> criteriaResults(7);
    ifstream inFile(inputFileName);
    string lineFile;
    vector <double> allElems;
    if (!inFile.is_open()) {
        cout << "Ошибка открытия файла " << inputFileName << endl;
        errorMessage = true;
        return criteriaResults;
    }
    string getlineElem;
    getline(inFile, getlineElem);
    stringstream ss(getlineElem);
    while (getline(ss, getlineElem, ',')) {
        if (getlineElem.size() > 1)
            getlineElem[1] = ',';
        double intElem = stod(getlineElem);
        allElems.push_back(intElem);
    }
    inFile.close();

    double expValue = calculateExpValue(allElems);
    cout << "Мат. ожидание: " << round(expValue * 1000) / 1000 << "\n";
    double standartDeviation = calculateStandartDeviation(allElems, expValue);
    cout << "Среднекв. отклонение: " << round(standartDeviation * 1000) / 1000 << "\n";
    double expValueError = calculateExpValueError(expValue);
    cout << "Погрешность мат. ожидания: " << round(expValueError * 1000) / 1000 << "\n";
    double standartDeviationError = calculateStandartDeviationError(standartDeviation);
    cout << "Погрешность среднекв. отклонения: " << round(standartDeviationError * 1000)
→  / 1000 << "\n";

    bool res = false;
    cout << "\n -----"
→  "\n";

```

```

        cout << "\n Критерий Хи-квадрат \n\n";
        criteriaResults[0] = checkXiSquareCriterion(allElems);
        cout << "\n -----\n";
    }

    cout << "\n Критерий серий \n\n";
    criteriaResults[1] = checkSeriesCriterion(allElems);
    cout << "\n -----\n";
}

cout << "\n Критерий интервалов \n\n";
criteriaResults[2] = checkIntervalsCriterion(allElems);
cout << "\n -----\n";
}

cout << "\n Критерий разбиений \n\n";
criteriaResults[3] = checkPartitionCriterion(allElems);
cout << "\n -----\n";
}

cout << "\n Критерий перестановок \n\n";
criteriaResults[4] = checkPermutationsCriterion(allElems);
cout << "\n -----\n";
}

cout << "\n Критерий монотонности \n\n";
criteriaResults[5] = checkMonotonicityCriterion(allElems);
cout << "\n -----\n";
}

cout << "\n Критерий конфликтов \n\n";
criteriaResults[6] = checkConflictCriterion(allElems);
cout << "\n -----\n";
}

return criteriaResults;
}

void coutCriterionsResult(vector <bool> criteriaResults) {
    vector <string> criteriaNames = { "Критерий хи-квадрат", "Критерий серий",
    "Критерий интервалов", "Критерий разбиений", "Критерий перестановок", "Критерий
    монотонности", "Критерий конфликтов"};
    cout << "\nРезультат тестирования статистических свойств последовательности ПСЧ:
    \n\n";
    for (int i = 0; i < criteriaResults.size(); i++)
    {
        cout << criteriaNames[i] << ": ";
        if (criteriaResults[i])
            cout << "+++" << endl;
        else
            cout << "---" << endl;
    }
    return;
}

```

```

int main(int argc, char* argv[])
{
    /*
    Программа для тестирования статистических свойств последовательности ПСЧ
    */
    setlocale(LC_ALL, "Rus");
    string inputFileName = "rntInput.dat";
    for (int i = 0; argv[i]; i++)
    {
        string checkStr = string(argv[i]);
        if (findInStr(checkStr, 2) == "/h") {
            helpFunc();
            return 0;
        }
        if (checkStr.length() > 2) {
            string ifStr = findInStr(checkStr, 3);
            string subStr = checkStr.substr(3, checkStr.length());
            if (ifStr == "/f:") {
                inputFileName = subStr;
            }
        }
    }
    bool errorMessage = false;
    vector <bool> resString = checkCriterions(inputFileName, errorMessage);
    if (errorMessage)
        return 0;

    cout << "Input File: " << inputFileName << endl;

    coutCriterionsResult(resString);
    return 0;
}

```

ПРИЛОЖЕНИЕ Б

Исходный код программы построения графиков

```
import matplotlib.pyplot as plt
import math

def expectedValue(vec):
    val = 0
    for i in vec:
        val = val + i
    val = val / len(vec)
    return val

f = open("D:\\cpp_projects\\rnt\\Debug\\stOutputLc.dat", "r")
#f = open("D:\\cpp_projects\\rnt\\Debug\\stOutputAdd.dat", "r")
#f = open("D:\\cpp_projects\\rnt\\Debug\\stOutput5p.dat", "r")
#f = open("D:\\cpp_projects\\rnt\\Debug\\stOutputLfsr.dat", "r")
#f = open("D:\\cpp_projects\\rnt\\Debug\\stOutputNfsr.dat", "r")
#f = open("D:\\cpp_projects\\rnt\\Debug\\stOutputMt.dat", "r")
#f = open("D:\\cpp_projects\\rnt\\Debug\\stOutputRc4.dat", "r")
#f = open("D:\\cpp_projects\\rnt\\Debug\\stOutputRsa.dat", "r")
#f = open("D:\\cpp_projects\\rnt\\Debug\\stOutputBbs.dat", "r")
currElem = []
mVec = []
dVec = []
nVec = []
acc = 0
allElems = list(map(float, (f.read()).split(',')))
for a in allElems:
    currElem.append(float(a))
    if (len(currElem) % 50 == 0):
        m = expectedValue(currElem)
        dev = 0
        for i in currElem:
            dev = dev + ((i - m) ** 2)
        dev = dev / (len(currElem) - 1)
        d = math.sqrt(dev)
        acc = acc + 50
        mVec.append(round(m, 3))
        dVec.append(round(d, 3))
        nVec.append(acc)
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,10))
for i in range(len(nVec)):
    ax1.scatter(nVec[i], mVec[i], c = 'b', s = 1)
    ax2.scatter(nVec[i], dVec[i], c = 'b', s = 1)
ax1.set_title("Математическое ожидание")
ax2.set_title("Среднеквадратическое отклонение")
plt.show()
```