Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Отчет по лабораторной работе №4**

По дисциплине: «Компьютерные системы моделирования»

Тема: «Имитационное моделирование. Моделирование частотной характеристики. Моделирование случайных событий»

|  |
| --- |
| Выполнил:  студент группы  Б.ПИН.РИС - 22.06  Иванов А.М. |
| Проверила:  старший преподаватель  кафедры ПО  Корнеева Е.И. |

Тверь 2025

**Оглавление**

[**Цель и постановка задачи для варианта.** 3](#_Toc190972657)

[**Теоритическая часть.** 4](#_Toc190972658)

[**Отчет по реализации.** 8](#_Toc190972659)

[**Python** 8](#_Toc190972660)

[**Excel** 20](#_Toc190972661)

[**Итоговая таблица** 22](#_Toc190972662)

[**Ссылка на репозиторий** 23](#_Toc190972663)

[**Выводы по результатам решения задачи.** 24](#_Toc190972664)

# **Цель и постановка задачи для варианта.**

**Цель работы**: Изучение методов и алгоритмов моделирования случайных чисел.

**Постановка задачи:**

Согласно варианту задания необходимо составить и отладить программу (подпрограмму) генерирования случайных чисел:

1) с равновероятным распределением на интервале [0; 1), используя формулу

2) используя готовый пакет Google | Excel надстройку Анализ данных и библиотеку numpy алгоритм из объекта random.

Провести статистическое исследование генератора при различных значениях выборки: малых n<25, средних , больших n>500.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вариант** | **Тип генератора** | **Начальные данные** | **Метод в numpy**  **(тип распределения в numpy.random)** | **Метод в Google|Excel**  **(надстройка ГСЧ)** |
| 10 | Мультипликативный конгруэнтный метод. | *k*=15; *а*=16807 | Лапласа | Бернулли |

# **Теоритическая часть.**

**Общие сведения**

Моделирование на ЭВМ процессов функционирования различных систем связано с выработкой большого количества случайных чисел с заданными законами распределения. Для этой цели используется обычно один из следующих способов:

- табличный (файловый) – ввод таблиц равномерно распределённых случайных чисел во внешнюю или оперативную память ЭВМ;

- аппаратный (физический) – использование специального приспособления к ЭВМ – "датчика" случайных чисел, формирующего случайные величины путём физического моделирования некоторых случайных процессов (излучения радиоактивных источников, шумов электронных ламп и др.);

- алгоритмический (программный) – использование псевдослучайных (квазислучайных) последовательностей, реализуемых программным генератором случайных чисел.

Псевдослучайными числами называются числа, вырабатываемые ЭВМ рекуррентным способом по специальным алгоритмам, когда каждое последующее число  получается из предыдущих в результате применения некоторых арифметических и логических операций. Такая последовательность чисел удовлетворяет известным критериям случайности, хотя входящие в эту последовательность числа зависимы между собой. Одним из недостатков этого метода является периодичность образованных программным способом псевдослучайных чисел, но для ряда задач, не требующих большого количества случайных чисел, длина периода является достаточной.

Достоинства метода псевдослучайных чисел.

1. На получение каждого случайного числа затрачивается несколько простых операций, так что скорость генерирования случайных чисел имеет тот же порядок, что и скорость работы ЭВМ.
2. Малый объем памяти ЭВМ для программирования.
3. Любое из чисел легко воспроизвести.
4. Качество генерируемых случайных чисел достаточно проверить один раз.

***Мультипликативный конгруэнтный метод (Алгоритм Лемера).***

Последовательность случайных чисел  получается с помощью следующего итерационного соотношения:

, (1)

где  и  - очередное и последующее случайное число;

*m* - модуль, *m* > 0;

*a*  -  множитель, 0 ≤ *a*< *m*;

 - начальное значение,  .

Если *m* и *а* являются целыми, то создается последовательность целых чисел в диапазоне . Из рекуррентной природы формулы следует, что при одном и том же значении  мы получим при повторной генерации ту же самую последовательность.

Последовательность случайных чисел периодически повторяются. Это связано с тем, что числа  могут принимать только значения . То есть, самое большее через  шагов уже один раз полученное число должно появиться опять, а с ним повторяется и вся последовательность. Таким образом, длина периода при модуле  не может превышать . Поэтому выбор значений *а* и *m* является критичным для разработки хорошего генератора случайных чисел. Для практических расчетов принимают .

Последовательность чисел , равномерно распределенных на интервале от нуля до единицы, рассчитывается по формуле

. (2)

**Проверка качества работы генератора случайных чисел (ГСЧ)**

Применяемые генераторы случайных чисел перед моделированием должны пройти тщательное предварительное тестирование на равномерность, стохастичность и независимость получаемых последовательностей случайных чисел. Существует множество статистических критериев, которые можно использовать для проверки того, будет ли последовательность случайной. Наиболее точным считается спектральный критерий. Например, очень распространенный критерий, называемый *КС*-критерием, или критерием Колмогорова - Смирнова.

Можно воспользоваться приближенным тестом, который состоит в проверке на равномерность распределения *N* случайных чисел .

а) *проверка статистических характеристик*. Тест состоит в вычислении математического ожидания  и дисперсии  полученных случайных чисел. Согласно этому тесту, для равномерного распределения должны выполняться условия:

;

;

.

б) *частотный тест*. Позволяет выяснить, сколько чисел попало в интервал , то есть (0.5 – 0.2887; 0.5 + 0.2887) или, в конечном итоге, (0.2113; 0.7887). Так как 0.7887 – 0.2113 = 0.5774, заключаем, что в хорошем ГСЧ в этот интервал должно попадать около 57,7% из всех выпавших случайных чисел (см. **рис.  1**).

|  |  |
| --- | --- |
| |  | | --- | |  | |

**Рис. 1. Частотная диаграмма идеального ГСЧ  
в случае проверки его на частотный тест**

Также необходимо учитывать, что количество чисел, попавших в интервал (0; 0.5), должно быть примерно равно количеству чисел, попавших в интервал (0.5; 1).

# **Отчет по реализации.**

## **Python**

**1. Генератор Лемера (LemerGenerator):**

Этот генератор работает на основе мультипликативного конгруэнтного метода. Он берёт текущее число, умножает его на определённый коэффициент (у нас это 16807), а затем берёт остаток от деления на модуль (32768). Полученное число нормализуется, чтобы оно попадало в диапазон от 0 до 1.

*class* LemerGenerator:

*def* \_\_init\_\_(*self*, *seed*, *a*=16807, *m*=32768):

*self*.m = *m*

*self*.a = *a*

*self*.x = *seed*

*self*.\_validate\_params()

*def* \_validate\_params(*self*):

        if *self*.m <= 0:

            raise ValueError("Modulus m must be positive")

        if *self*.a <= 0 or *self*.a >= *self*.m:

            raise ValueError(*f*"Multiplier a must be in (0, {*self*.m})")

        if *self*.x <= 0 or *self*.x >= *self*.m:

            raise ValueError(*f*"Seed must be in (0, {*self*.m})")

*def* next(*self*):

*self*.x = (*self*.a \* *self*.x) % *self*.m

        return *self*.x / *self*.m

*def* current\_raw(*self*):

        return *self*.x

**2. Генератор Лапласа (LaplaceSampleGenerator):**

Здесь мы использовали готовую функцию из библиотеки numpy — numpy.random.laplace. Она позволяет генерировать числа с распределением Лапласа, которое имеет пик в центре и "тяжёлые хвосты". Мы задали центр распределения (loc) как 0.5 и параметр масштаба (scale) как 0.5.

*class* LaplaceSampleGenerator:

*def* \_\_init\_\_(*self*, *loc*=0.0, *scale*=1.0):

        """

        Генератор выборок с распределением Лапласа

        :param loc: параметр положения (среднее значение)

        :param scale: параметр масштаба (должен быть > 0)

        """

        if *scale* <= 0:

            raise ValueError("Scale parameter must be positive")

*self*.loc = *loc*

*self*.scale = *scale*

*def* generate\_sample(*self*, *size*):

        """Генерирует выборку заданного размера"""

        if *size* <= 0:

            raise ValueError("Sample size must be positive")

        data = np.random.laplace(*loc*=*self*.loc, *scale*=*self*.scale, *size*=*size*)

        return Sample(data.tolist())

**3. Класс Sample:**

Это просто контейнер для хранения данных.

*class* Sample:

*def* \_\_init\_\_(*self*, *data*=None):

*self*.data = *data* if *data* is not None else []

*def* add\_value(*self*, *value*):

*self*.data.append(*value*)

*def* clear(*self*):

*self*.data.clear()

*def* \_\_len\_\_(*self*):

        return len(*self*.data)

*def* \_\_getitem\_\_(*self*, *index*):

        return *self*.data[*index*]

*def* \_\_str\_\_(*self*):

        return *f*"Sample[size={len(*self*.data)}]"

**4. Класс SampleGenerator:**

Этот класс использует один из генераторов (Лемера или Лапласа) и создаёт выборку заданного размера. Например, можно сгенерировать 1000 чисел и сохранить их в объекте Sample.

*class* SampleGenerator:

*def* \_\_init\_\_(*self*, *rng*):

*self*.rng = *rng*

*def* generate\_sample(*self*, *size*):

        if *size* <= 0:

            raise ValueError("Sample size must be positive")

        sample = Sample()

        for \_ in range(*size*):

            sample.add\_value(*self*.rng.next())

        return sample

**5. Класс SampleStatistics:**

Здесь мы считаем основные статистические характеристики: среднее значение, дисперсию и стандартное отклонение. Также мы реализовали частотный тест, который проверяет, сколько чисел попадает в определённый интервал (для равномерного распределения это должно быть около 57.7%).

*class* SampleStatistics:

*def* \_\_init\_\_(*self*, *sample*):

        if len(*sample*) == 0:

            raise ValueError("Sample is empty")

*self*.sample = *sample*

*def* mean(*self*):

        return sum(*self*.sample) / len(*self*.sample)

*def* variance(*self*):

        n = len(*self*.sample)

        avg = *self*.mean()

        return sum((x - avg)\*\*2 for x in *self*.sample) / n

*def* std\_deviation(*self*):

        return math.sqrt(*self*.variance())

*def* frequency\_test(*self*, *expected\_std*=0.2887):

        """Частотный тест для равномерного распределения"""

        lower\_bound = 0.5 - *expected\_std*

        upper\_bound = 0.5 + *expected\_std*

        count = sum(1 for x in *self*.sample if lower\_bound < x < upper\_bound)

        total = len(*self*.sample)

        percentage = (count / total) \* 100

        return {

            'interval': (lower\_bound, upper\_bound),

            'count': count,

            'percentage': percentage,

            'expected\_percentage': 57.7,

            'deviation': abs(percentage - 57.7)

        }

*def* get\_report(*self*):

        freq = *self*.frequency\_test()

        return (

*f*"Statistical Report:\n"

*f*"• Size: {len(*self*.sample)*:,*}\n"

*f*"• Mean: {*self*.mean()*:.4f*}\n"

*f*"• Variance: {*self*.variance()*:.4f*}\n"

*f*"• Standard Deviation: {*self*.std\_deviation()*:.4f*}\n"

*f*"• Frequency Test ({freq['interval'][0]*:.4f*} - {freq['interval'][1]*:.4f*}):\n"

*f*"  - Numbers in range: {freq['count']*:,*} ({freq['percentage']*:.1f*}%)\n"

*f*"  - Expected: {freq['expected\_percentage']}%\n"

*f*"  - Deviation: {freq['deviation']*:.1f*}%"

        )

**6. Класс SampleVisualizer:**

Этот класс отвечает за визуализацию данных. Мы можем построить гистограмму, boxplot или даже наложить теоретическую кривую распределения для сравнения. Всё это сохраняется в виде картинок или отображается на экране.

*class* SampleVisualizer:

*def* \_\_init\_\_(*self*, *sample*):

        if len(*sample*) == 0:

            raise ValueError("Cannot visualize empty sample")

*self*.sample = *sample*

*self*.data = np.array(*sample*.data)

*def* plot\_histogram(*self*, *bins*=30, *density*=False, *title*="Histogram", *color*='skyblue', *edgecolor*='black'):

        """Построение гистограммы распределения"""

        plt.figure(*figsize*=(10, 6))

        plt.hist(*self*.data, *bins*=*bins*, *density*=*density*,

*color*=*color*, *edgecolor*=*edgecolor*, *alpha*=0.7)

        plt.title(*title*)

        plt.xlabel("Values")

        plt.ylabel("Frequency" if not *density* else "Density")

        plt.grid(True, *linestyle*='--', *alpha*=0.7)

        return *self*

*def* plot\_density(*self*, *title*="Density Plot", *color*='blue', *linewidth*=2):

        """Оценка плотности распределения (KDE)"""

        from sklearn.neighbors import KernelDensity

        plt.figure(*figsize*=(10, 6))

        kde = KernelDensity(*bandwidth*=0.02, *kernel*='gaussian')

        kde.fit(*self*.data[:, None])

        x = np.linspace(*self*.data.min(), *self*.data.max(), 1000)

        log\_prob = kde.score\_samples(x[:, None])

        plt.plot(x, np.exp(log\_prob), *color*=*color*, *linewidth*=*linewidth*)

        plt.title(*title*)

        plt.xlabel("Values")

        plt.ylabel("Density")

        plt.grid(True, *linestyle*='--', *alpha*=0.7)

        return *self*

*def* plot\_boxplot(*self*, *title*="Boxplot", *color*='teal', *vert*=False):

        """Построение boxplot"""

        plt.figure(*figsize*=(10, 4))

        plt.boxplot(*self*.data, *vert*=*vert*, *patch\_artist*=True,

*boxprops*=dict(*facecolor*=*color*, *color*='black'),

*whiskerprops*=dict(*color*='black'),

*capprops*=dict(*color*='black'),

*medianprops*=dict(*color*='red'))

        plt.title(*title*)

        plt.grid(True, *linestyle*='--', *alpha*=0.7)

        return *self*

*def* add\_theoretical(*self*, *dist\_type*='uniform', \*\**params*):

        """Добавление теоретической кривой для сравнения"""

        x = np.linspace(*self*.data.min(), *self*.data.max(), 1000)

        if *dist\_type* == 'uniform':

            y = np.ones\_like(x) / (*params*.get('b', 1) - *params*.get('a', 0))

            plt.plot(x, y, 'r--', *linewidth*=2, *label*='Uniform PDF')

        elif *dist\_type* == 'laplace':

            loc = *params*.get('loc', 0)

            scale = *params*.get('scale', 1)

            y = (1/(2\*scale)) \* np.exp(-np.abs(x - loc)/scale)

            plt.plot(x, y, 'r--', *linewidth*=2, *label*='Laplace PDF')

        plt.legend()

        return *self*

*def* save(*self*, *filename*="plot.png", *dpi*=300):

        """Сохранение последнего графика"""

        plt.savefig(*filename*, *bbox\_inches*='tight', *dpi*=*dpi*)

        plt.close()

        return *self*

*def* show(*self*):

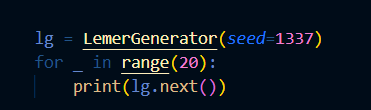
        """Отображение всех графиков"""

        plt.show()

        return *self*

## 

**Пример вывода случайных чисел**

****

**0.759246826171875**

**0.661407470703125**

**0.275360107421875**

**0.977325439453125**

**0.908660888671875**

**0.863555908203125**

**0.784149169921875**

**0.195098876953125**

**0.026824951171875**

**0.846954345703125**

**0.761688232421875**

**0.694122314453125**

**0.113739013671875**

**0.611602783203125**

**0.207977294921875**

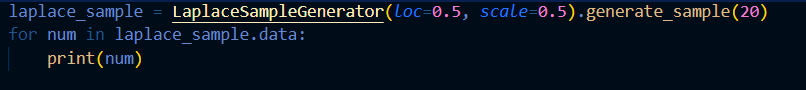
**0.474395751953125**

**0.169403076171875**

**0.157501220703125**

**0.123016357421875**

**0.535919189453125**

****

**0.3579730980457207**

**-0.30808780603409525**

**-0.5845024511358863**

**0.920734025397234**

**-0.10224252934855749**

**0.30276705721477104**

**0.6311944833771878**

**-0.5742203556564234**

**1.1483093017684027**

**0.2745834060837873**

**1.3876830822084028**

**0.09129800465522003**

**0.8445143369576383**

**1.2460556684889488**

**1.936083654882029**

**-0.03995726666681032**

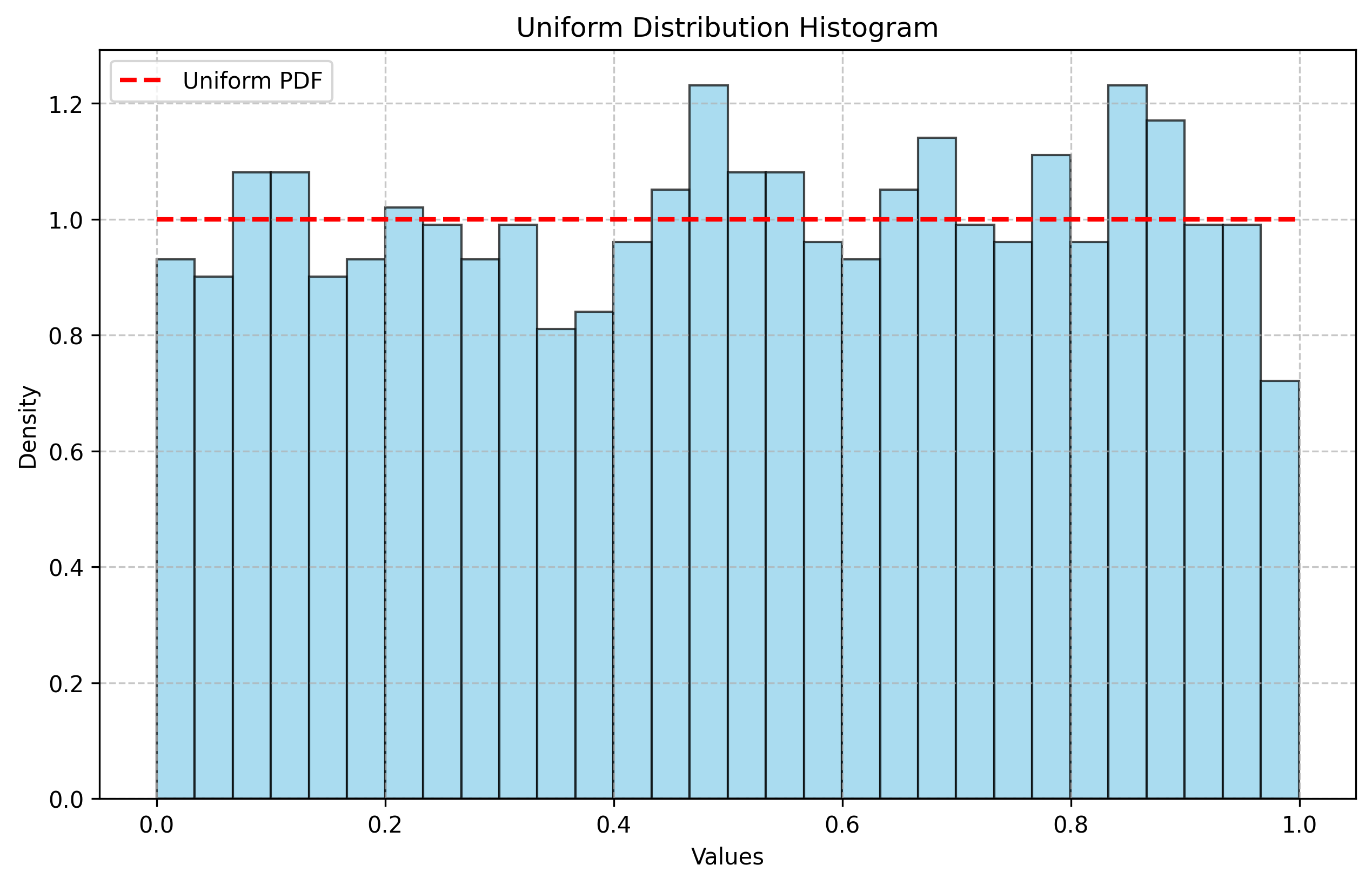
**0.4532113946246983**

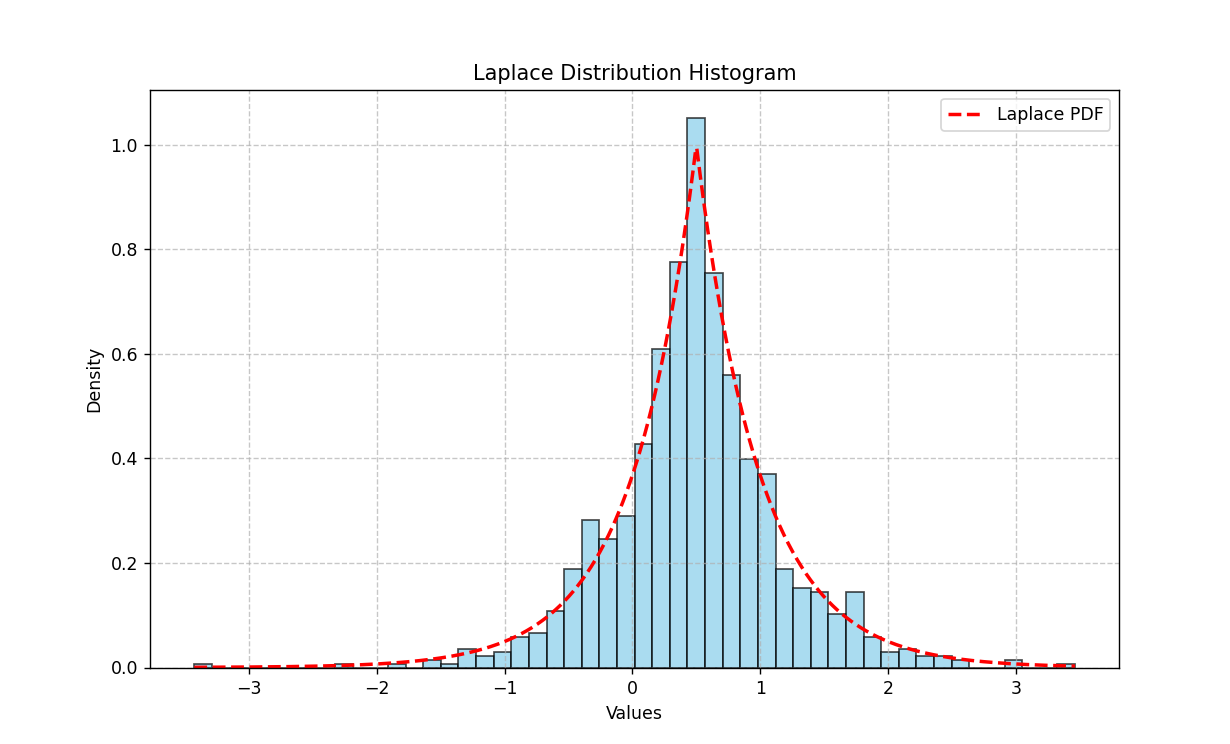
**1.859030330422762**

**1.2720590058236971**

**0.6239437333550539**

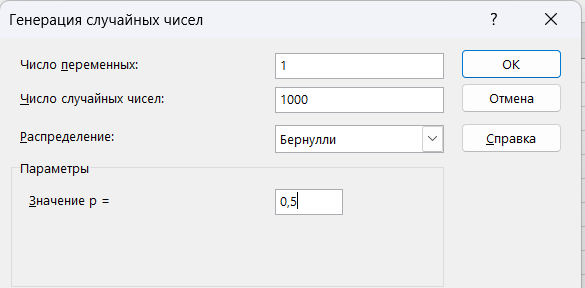
**Диаграммы (выборка 1000 элементов)**



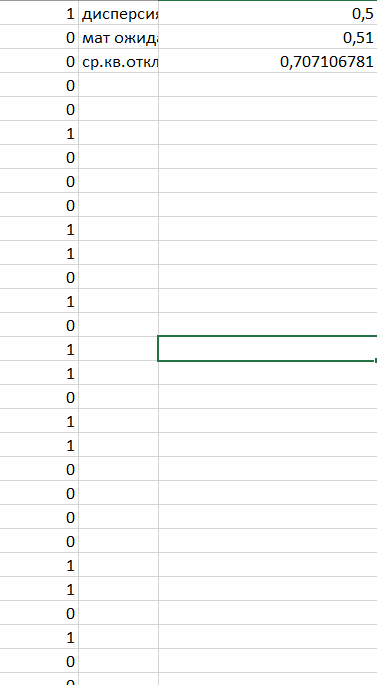
****

## **Excel**

По условию задания были заданы следующие настройки ГСЧ:

****

**Полученный результат**

****

## **Итоговая таблица**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Метод Лемера** | | | **np.random** | | | **Метод Бернулли (Excel)** | | |
| **n выборки** | **Мат.ожидание** | **Дисперсия** | **Ср. Кв. откл** | **Мат.ожидание** | **Дисперсия** | **Ср. Кв. откл** | **Мат.ожидание** | **Дисперсия** | **Ср. Кв. откл** |
| **10** | **0.5240** | **0.0338** | **0.1839** | **0.3514** | **1.0080** | **1.0040** | **0.3** | **0.23** | **0.483** |
| **25** | **0.5132** | **0.0757** | **0.2751** | **0.5237** | **0.7458** | **0.8636** | **0.44** | **0.2567** | **0.506** |
| **120** | **0.5189** | **0.0702** | **0.2650** | **0.5260** | **0.7087** | **0.8418** | **0.508** | **0.252** | **0.502** |
| **400** | **0.5198** | **0.0833** | **0.2885** | **0.4710** | **0.4607** | **0.6787** | **0.5125** | **0.2504** | **0.5** |
| **800** | **0.5017** | **0.0819** | **0.2862** | **0.5040** | **0.4408** | **0.6639** | **0.50125** | **0.250** | **0.5** |
| **1500** | **0.4981** | **0.0832** | **0.2885** | **0.4866** | **0.4914** | **0.7010** | **0.511** | **0.250** | **0.5** |
| **10000** | **0.5002** | **0.0834** | **0.2887** | **0.5043** | **0.4847** | **0.6962** | **0.5013** | **0.250** | **0.5** |

## 

# **Ссылка на репозиторий**

https://github.com/ghimik/Computer-Simulating-Systems-Labs

# **Выводы по результатам решения задачи.**

Проведенное исследование позволило сравнить три метода генерации случайных чисел: **мультипликативный конгруэнтный метод Лемера**, генерацию распределения Лапласа через numpy.random и метод Бернулли в Excel. Основные результаты анализа:

**Метод Лемера**

Продемонстрировал высокую согласованность с теоретическими характеристиками равномерного распределения. При больших выборках (n ≥ 10000) математическое ожидание стремится к **0.5**, дисперсия — к **≈0.0833**, а стандартное отклонение — к **≈0.2887**, что соответствует требованиям равномерного распределения на интервале [0; 1).

Частотный тест показал, что **57–58% чисел (57% при выборке в 400 элементов, 57.6% при выборках выше, при выборке в 10000 элементов – 57.7% ровно) попадают в интервал (0.2113; 0.7887)**, что близко к ожидаемым 57.7%. Это подтверждает качество генератора.

Метод устойчив к увеличению объема выборки: при n=10000 отклонения минимальны.

**Генератор Лапласа (numpy.random)**:

Математическое ожидание близко к **0.5**, но дисперсия значительно выше (**≈0.48**), что характерно для распределения Лапласа с «тяжелыми хвостами».

Метод не подходит для генерации равномерного распределения, но соответствует заявленному распределению Лапласа.

**Метод Бернулли (Excel)**:

Показал математическое ожидание **≈0.5** и дисперсию **≈0.25**, что соответствует бинарному распределению Бернулли (p=0.5).

Данный метод не предназначен для генерации равномерно распределенных чисел, что указывает на несоответствие постановке задачи.

**Итоговая оценка**

**Метод Лемера** полностью удовлетворяет требованиям задания: он стабилен, точен и воспроизводит равномерное распределение даже на больших выборках.

Генераторы Лапласа и Бернулли оказались неприменимы для решения поставленной задачи из-за несоответствия типов распределений.