МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра

Мойсейчик Елизавета Сергеевна

Отчет по лабораторным работам по курсу “Имитационное и статистическое моделирование”

студентки 2 курса 13 группы

Работа сдана 2020 г.

зачтена __________ 2020 г.

____

(подпись преподавателя)

Преподаватель

Лобач Сергей Викторович

ассистент кафедры ММАД

Минск 2020

Оглавление

Лабораторная работа 1.

Условие:

а) Осуществить моделирование n = 1000 реализаций БСВ с помощью мультипликативного конгруэнтного метода (МКМ) с параметрами a0= a01, β = max{c1, M – c1}, M = 231 и вывести 100-ый, 900-ый и 1000-ый элементы сгенерированной последовательности.

б) Осуществить моделирование n = 1000 реализаций БСВ с помощью метода Макларена-Марсальи, используя в качестве простейших датчиков БСВ датчики D1 – датчик из первого задания, D2 – датчик по методу МКМ с параметрами a0= a02, β = max{c2, M – c2}, M = 231, K – объем вспомогательной таблицы и вывести 100-ый, 900-ый и 1000-ый элементы сгенерированной последовательности.

Вариант	a_{01}	c_1	a_{02}	c_2	K
7	24149775	19581355	179029053	457816087	128

Дополнительные задания:

Для каждого из построенных генераторов:

1) (2 балла) Проверить точность моделирования с помощью теста «совпадения моментов» с уровнем значимости ε = 0.05.

2) (2 балла) Проверить точность моделирования с помощью теста «ковариация» с уровнем значимости ε = 0.05. В качестве параметра t выбрать значение 30. Вывести все такие значения лага, при котором тест не проходит.

3) (3 балла) Проверить точность моделирования с помощью теста «равномерность двумерного распределения» с уровнем значимости ε = 0.05. Параметр k выбирать самостоятельно.

Теория:

Мультипликативный конгруэнтный метод:

Псевдослучайная последовательность $\alpha_1,\ \alpha_2,\ ...,\alpha_n$ строится по следующим рекуррентным формулам:

$$\alpha_t = \alpha_t^* / M,\ \ \alpha_t^* = \{\beta\alpha_{t-1}^*\} \bmod M \quad (t = 1, 2, ...),$$

где β, M, α_0^* - параметры датчика: β - множитель ($\beta < M$), M – модуль, $\alpha_0^* \in \{1, ..., M-1\}$ - стартовое значение (нечетное число).

В данной работе брались значения: M=2147483648, $\alpha_0^* = \beta$ =65539.

Метод Маклорена-Марсальи:

Пусть $\{\beta_t\}$, $\{c_t\}$ - псевдослучайные последовательности, порожденные независимо работающими датчиками; $\{\alpha_t\}$ - результирующая псевдослучайная последовательность реализация БСВ;
V={V(0), V(1), …,V(K-1)} – вспомогательная таблица *K* чисел.

Процесс вычисления $\{\alpha_t\}$ включает следующие этапы:

- первоначальное заполнение таблицы

$$V\colon\ V(i)=\beta_i,\ i=\overline{0,K-1};$$

- случайный выбор из таблицы:

$$\alpha_t=V(s),\ s=[c_t\cdot K];$$

-обновление табличных значений:

$$V(s)=b_{t+K},\ t=0,\,1,\,2,\,\ldots.$$

В данной работе в качестве $\{\beta_t\}$ бралась последовательность (из 100 элементов), полученная мультипликативным конгруэнтным методом, описанным выше. В качестве $\{c_t\}$, бралась последовательности (из 10000) элементов, полученная аналогичным способом с тем же M и $\beta'=3\beta+1$. *K*=100.

Тест «совпадение моментов»:

Пусть в результате n-кратного обращения к датчику БСВ получена выбора значений $A=\{a_1,\ldots,a_n\}$. Известно, что БСВ имеет среднее знаечние $\mu=1/2$ и дисперсию $\sigma^2=1/12$.. Обозначим:

$$\xi_1=m-1/2,\qquad \xi_2=s^2-1/12$$

- случайные отклонения выборочных оценок

$$m=\frac{1}{n}\sum_{i=1}^{n}a_i\ ,\qquad s^2=\frac{1}{n-1}\sum_{i=1}^{n}(a_i-m)^2$$

От истинных характеристик μ,σ^2.

Тест «совпадения моментов» - это программа для ЭВМ, реализующая статические критерии проверки по выборке А гипотез:

$$H_0:\mu=1/2,\qquad H_1:\mu\neq 1/2;\qquad \mathbf{(4)}$$
$$H_0:\sigma^2=1/12,\qquad H_1:\sigma^2\neq 1/12;\qquad \mathbf{(5)}$$

Решающее правило для проверки гипотез (4), (5) имеет вид:

$$\text{принимается}\ \begin{cases}H_0, c_i(n)\cdot|\xi_i|<\Lambda\\ H_1, \textit{иначе},\end{cases}\qquad \mathbf{(6)}$$

Где i={1 – для соотношений (4), 2 – для соотношений (5)},

$c_1(n) = \sqrt{12n},$ $\qquad c_2(n) = \frac{n-1}{n}(0.0056n^{-1} + 0.0028n^{-2} - 0.0083n^{-3})^{-\frac{1}{2}}$

- нормировочные множители, Δ - порог критерия.

Если H_0 верна, а $n >> 1$ (практически $n \geq 20$), то в силу ЦПТ: $c_i(n)\xi_i \sim N_1(0,1)$ (распределено приближённо по стандартному нормальному закону). С учётом этого из ограничения на вероятность ошибки первого рода:

$$P\{H_0 \mid H_1\} = P\{c_i(n) \mid \xi_i \mid \geq 0 \mid H_0\} = \varepsilon(0 < \varepsilon < 1),$$

находится выражение для порога критериев:

$$\Delta = \Phi^{-1}(1 - \frac{\varepsilon}{2}),$$

где $\Phi^{-1}(\cdot)$ - квантиль стандартного нормального закона, ε - заданный уровень значимости.

Тест «ковариация»:

Ковариационной функцией случайной последовательности $\alpha_1,...,\alpha_n$ называется функция целочисленной переменной ξ ∈{0,1,..., n −1}:

$$R(j) = E\{(\alpha_1 - E\{\alpha_1\})(\alpha_{1+j} - E\{\alpha_{1+j}\})\} = E\{\alpha_1 \cdot \alpha_{1+j}\} - E\{\alpha_1\} \cdot E\{\alpha_{1+j}\}$$

Если $\alpha_1,...,\alpha_n$ -- независимые, одинаково распределённые по закону R(0,1) случайные величины, то α_1 и α_{1+j} независимы для любого j ≥ 1 и следовательно :

$$R(j) = \begin{cases} 1/12, j = 0, \\ 0, j \geq 0. \end{cases} \qquad \textbf{(7)}$$

Пусть $\hat{R}(j)$ -- оценка R(j) по выборке $A = \{a_1,...,a_n\}$, полученной в результате n - кратного обращения к исследуемому датчику :

$$\hat{R}(j) = \frac{1}{n-j-1}\sum_{i=1}^{n-j} a_1 \cdot a_{1+j} - \frac{n}{n-1}m^2, j = 0,1,...,t,$$

где $1<t<<n, m$ - выборочное среднее. Заметим , что $\hat{R}(j) = s^2 (s$ - выборочная дисперсия).

Тест «ковариация» позволяет проверить свойство (7) (гипотезу H0) для последовательности $a_1,...,a_n$ и описывается следующим решающим правилом:

принимается $\begin{cases} H_0, |\hat{R}(j) - R(j)| < \dfrac{c_j \Lambda}{12\sqrt{n-1}}, \\ H_1, иначе, \end{cases}$ **(8)**

где : $c_0 = \sqrt{2}$, $c_j = 1$ для $j \geq 1$; Δ - порог, определённый для заданного уровня значимости ε по формуле :

$$\Delta = \Phi^{-1}(1 - \frac{\varepsilon}{2}).$$

Тест «равномерность двумерного распределения»:

Из выборочных значений $a_1,...,a_n$ полученных в результате n-кратного обращения к датчику БСВ построим $m = [n/2]$ ($[z]$-целая часть числа z) векторов :

$$(a_1, a_2), (a_3, a_4), ...(a_{2(m-1)+1}, a_{2m}). \quad \textbf{(10)}$$

Единичный квадрат $\Xi \subset R^2$ с центром в точке $c = (0.5, 0.5) \in R^2$ разобьём на k ячеек :

$$\Xi_i = \{x : x \in \Xi, r_{i-1} \leq |x - c| < r_i\}, i = \overline{1, k-1};$$
$$\Xi_k = \Xi \setminus \bigcup_{i=1}^{k-1} \Xi_i,$$

где $r_0 = 0 < r_1 < ... < r_{k-1} \leq 0.5$ -- произвольные вещественные числа.

Вычислим частоты $\{m_i\}$ попадания m точек с координатами (10) в k ячеек $\{\Xi_i\}\left(\sum_{i=1}^{k} m_i = m\right)$ гиперкуба.

Если $\alpha_1,...,\alpha_n$ -- независимые, одинаково распределённые по закону R(0,1) случайные величины, то теоретические вероятности $\{p_i\}$ попадания точек с координатами $(a_{2(j-1)+1}, a_{2j}), j = \overline{1, m}$ в ячейки $\{\Xi_i\}$ равны площадям этих ячеек :

$$p_i = \begin{cases} \pi(r_i^2 - r_{i-1}^2), i = \overline{1, k-1} \\ 1 - \pi r_{k-1}^2, i = k. \end{cases}$$

Описываемый тест используется для проверки гипотезы H_0 о равномерности двухмерного распределения векторов $\{(a_{2(j-1)+1}, a_{2j})\}$ и представляет собой следующее решающее правило:

$$\text{принимается} \begin{cases} H_0, x^2 < \Delta, \\ H_1, x^2 \geq \Delta, \end{cases} \quad (11)$$

где в случае истинной гипотезы H_0 и $n \to \infty$ статистика

$$x^2 = \sum_{i=1}^{k} \frac{(m_i - mp_i)^2}{mp_i} \quad (12)$$

имеет x^2 – распределение с $k - 1$ степенями свободы, а порог Δ определяется как квантиль этого распределения: $\Delta = F^{-1}(1-\varepsilon)$, где ε – заданный уровень значимости.

Код программы:

```
public class RandomBSV : IGenerator
    {
        private int a0;

        private static int M = int.MaxValue;

        private long a, b, c;


        public RandomBSV(int a, int c)
        {
            this.a0 = a;
            this.c = c;

            int temp = M - c;
            b = (c < temp) ? temp : c;

            Refresh();
        }


        public double Next()
        {
            a = (b * a) % M;
            return (double)a / M;
        }


        public void Refresh()
```

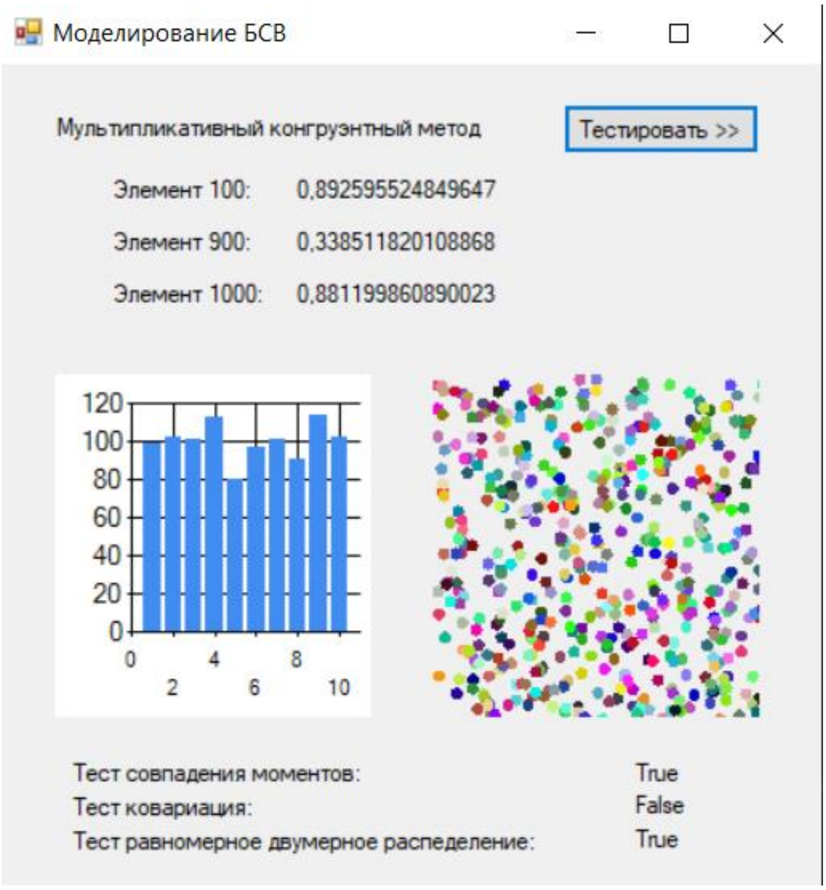
```
            {
                a = a0;
            }
        }

    public class MacLarenRandom : IGenerator
        {
            private static int K = 128;

            RandomBSV rand1, rand2;

            private double[] table;


            public MacLarenRandom(RandomBSV rand1, RandomBSV rand2)
            {
                table = new double[K];

                this.rand1 = rand1;
                this.rand2 = rand2;

                for (int i = 0; i < K; i++)
                    table[i] = rand1.Next();
            }


            public double Next()
            {
                int index = (int)(K * rand2.Next());
                double result = table[index];

                table[index] = rand1.Next();

                return result;
            }


            public void Refresh()
            {
                rand1.Refresh();
                rand2.Refresh();
            }
        }

    public class AccuracyTester
        {
            public double epsilon;

            Chart chart;

            Func<double, double> NormalDistribution;


            public AccuracyTester(double epsilon)
            {
                this.epsilon = epsilon;

                chart = new Chart();
                NormalDistribution = chart.DataManipulator.Statistics.NormalDistribution;
            }


            public bool TestGMM(IGenerator random, int sampleSize)
            {
                List<double> sample = SampleGenerator.GenerateSample(random, sampleSize);
```

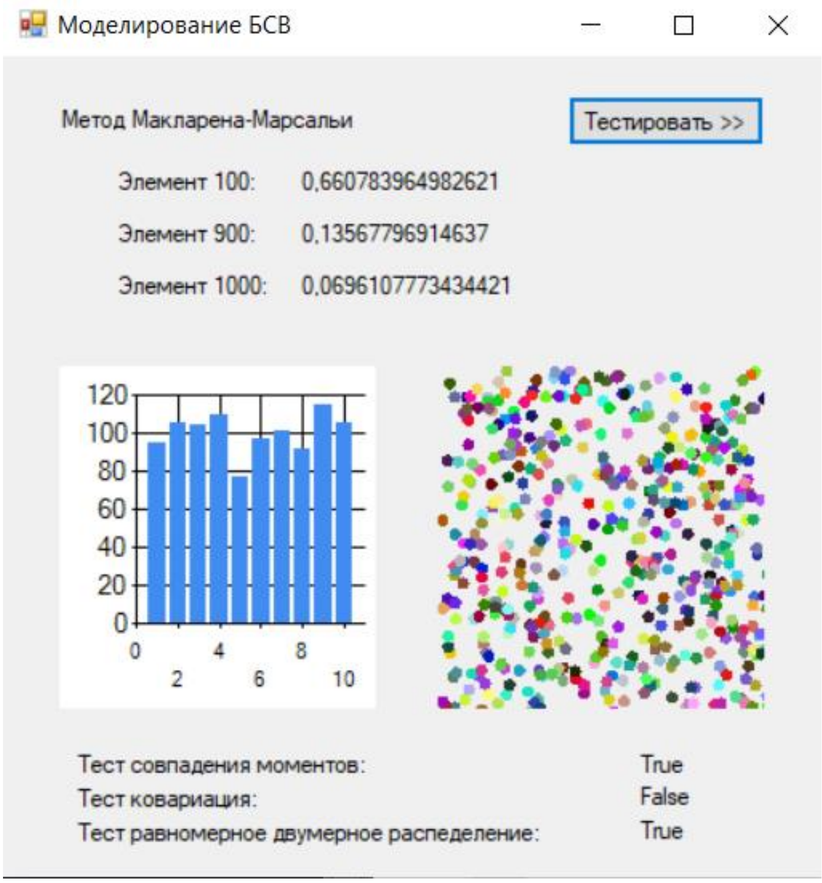
```
            double
                E = sample.Average(),
                D = 0;

            for (int i = 0; i < sampleSize; i++)
                D += Math.Pow(E - sample[i], 2);

            D /= sampleSize - 1;

            double[] x = new double[2];

            x[0] = E - 0.5;
            x[1] = D - 1 / 12;

            double[] c = new double[2];

            c[0] = Math.Sqrt(12 * sampleSize);

            c[1] = (sampleSize - 1) / sampleSize / Math.Sqrt((0.0056 / sampleSize
                + 0.0028 / Math.Pow(sampleSize, 2) - 0.0083 / Math.Pow(sampleSize,
3)));

            double[] P = new double[2];

            bool[] res = new bool[2];

            for (int i = 0; i < 2; i++)
            {
                P[i] = 2 * (1 - NormalDistribution(c[i] * Math.Abs(x[i])));
                res[i] = epsilon < P[i];
            }

            return res[0] && res[1];
        }


        public bool CovarianceTest(IGenerator random, int sampleSize, int t)
        {
            List<double> sample = SampleGenerator.GenerateSample(random, sampleSize);

            List<double>
                R = new List<double>(),   //^cov
                P = new List<double>();

            double E = sample.Average();

            for (int i = 0; i < sampleSize; i++)
            {
                int to = sampleSize - i;

                double sum = 0;

                for (int j = 0; j < to; j++)
                    sum += sample[j] * sample[j + sampleSize - to];

                R.Add(1.0 / (sampleSize - i - 1) * sum
                    - sampleSize / (sampleSize - 1) * E * E);
            }

            double
                r = 1.0 / 12, //cov
                c = Math.Sqrt(2);

            P.Add(2 * (1.0 - NormalDistribution(12 * Math.Sqrt(sampleSize - 1)
                * Math.Abs(R[0] - r) / c)));
```

```
            r = 0;
            c = 1;

            for (int i = 1; i <= t; i++)
            {
                P.Add(2 * (1.0 - NormalDistribution(12 * Math.Sqrt(sampleSize - 1)
                * Math.Abs(R[i] - r) / c)));
            }

            for (int i = 0; i <= t; i++)
            {
                if (epsilon >= P[i])
                    return false;
            }

            return true;
        }


        public bool UniformTwoDimensionalDistributionTest(IGenerator random, int
sampleSize, int t)
        {
            List<double> sample = SampleGenerator.GenerateSample(random, sampleSize);

            double
                deltaR = 0.5 / (t - 1),
                hi = 0;

            int[] frequency = new int[t];

            int m = sampleSize / 2;

            for (int i = 0; i < sampleSize; i += 2)
            {
                double
                    x = sample[i],
                    y = sample[i + 1];

                bool isInnerPoint = false;

                for (int j = 0; j < t - 1; j++)
                {
                    if (Math.Pow(x - 0.5, 2) + Math.Pow(y - 0.5, 2)
                        < Math.Pow(deltaR * (j + 1), 2))
                    {
                        frequency[j]++;
                        isInnerPoint = true;
                        break;
                    }
                }

                if (!isInnerPoint)
                    frequency[t - 1]++;
            }

            double p;

            for (int i = 0; i < t - 1; i++)
            {
                p = Math.PI *
                    (Math.Pow(deltaR * (i + 1), 2) - Math.Pow(deltaR * i, 2));

                hi += Math.Pow((frequency[i] - m * p), 2) / (m * p);
            }

            p = 1 - Math.PI * 0.25;
```

```
            hi += Math.Pow((frequency[t - 1] - m * p), 2) / (m * p);

            double P = 1 - MathNet.Numerics.Distributions.ChiSquared.CDF(t - 1, hi);

            return epsilon < P;
        }
    }
```

Результаты работы:




Моделирование БСВ
Метод Макларена-Марсальи
Тестировать >>
Элемент 100: 0,660783964982621
Элемент 900: 0,13567796914637
Элемент 1000: 0,0696107773434421
120
100
80
60
40
20
0
0
2
4
6
8
10
Тест совпадения моментов: True
Тест ковариация: False
Тест равномерное двумерное распеделение: True

Лабораторная работа 2.

Условие:

1) Осуществить моделирование n = 1000 реализаций СВ из заданных дискретных распределений для этого можно использовать любой генератор БСВ (как реализованный в 1-ой лабораторной работе, так и встроенный в язык программирования). Вывести на экран несмещенные оценки математического ожидания и дисперсии, сравнить их с истинными значениями.

Вариант:

4) Геометрическое – G(p), p = 0.3; Биномиальное – Bi(m,p), m = 4, p = 0.2.

13) Бернулли – Bi(1,p), p = 0.2013; Биномиальное – Bi(m,p), m = 8, p = 0.5.

Дополнительные задания:

1) (1 балл) Вычислить несмещенные оценки коэффициентов эксцесса и асимметрии и сравнить с истинными значениями.

2) (1 балл) Построить гистограмму и сравнить с графиком теоретического распределения вероятностей (на одном графике).

3) (2 балла) Построить график эмпирической функции распределения и сравнить с графиком теоретической функции распределения.

4) (2 балла) Реализовать критерий хи-квадрат Пирсона проверки статистической гипотезы о принадлежности смоделированной последовательности к заданному распределению.

Теория:

Распределение Бернулли:

ДСВ ξ имеет распределение Бернулли *Bi(1,p)*, если: $\xi \in \{0,1\}$, $P\{\xi = 1\} = p, P\{\xi = 0\} = 1 - p$, где p ε (0,1) – параметр распределения.

Характеристики распределения Бернулли (x ε{0,1}):

- функция распределения:

$$F_\xi(x) = \sum_{i=0}^{x} p^i (1-p)^{1-i};$$

- функция вероятности:

$$P_\xi(x) = p^x (1-p)^{1-x};$$

- среднее значение: $\mu = p$;
- дисперсия: $\sigma^2 = p(1-p)$.

Алгоритм моделирования одной реализации случайной величины Бернулли состоит из двух шагов:

1. Моделирование реализации БСВ.
2. Принятие решения о том, что реализацией ξ является значение x определяемое по правилу:

$$x = \begin{cases} 1, a \le p, \\ 0, a > p. \end{cases}$$

Коэффициент использования БСВ $k = 1$.

Биномиальное распределение:

ДСВ ξ имеет биномиальное распределение *Bi(m,p)*, если: $\xi \in \{0,1,\dots m\},\ P\{\xi = i\} = C_i^m p^i (1-p)^{m-i}, i \in \{0,1,\dots,m\}$.

Параметры распределения: m – натуральное число; p ε (0,1).

Характеристики распределения *Bi(m,p)* (x ε {0,1,…,m}):

- функция распределения:

$$F_\xi(x) = \sum_{i=0}^{x} C_i^m p^i (1-p)^{m-i};$$

- функция вероятности:

$$P_\xi(x) = C_x^m p^x (1-p)^{m-x};$$

- среднее значение: $\mu = m \cdot p$;
- дисперсия: $\sigma^2 = mp(1-p)$.

1. Метод браковки. Алгоритм моделирования реализации биноминальной СВ ξ по методу браковки состоит из двух шагов:

   (a) Моделирование m реализаций $a_1, \dots, a_m$ БСВ.

   (b) Принятие решения о том, что реализацией ξ является значение x, вычисляемое по формуле:

$$x = \sum_{i=1}^{m} 1(p - a_i),$$

где $1(z) = \begin{cases} 0, z \le 0; \\ 1, z > 0. \end{cases}$

Таким образом, x – количество значений из $\{a_i\}$, меньших p.

Коэффициент использования БСВ $k = \frac{1}{m}$.

Геометрическое распределение:

ДСВ ξ имеет геометрическое распределение $G(p)$, если: $\xi \varepsilon \{1,2,\dots\}$, $P\{\xi = i\} = p \bullet (1-p)^{i-1}$, $i \varepsilon \{1,2,\dots\}$, где $p \varepsilon (0,1)$ – параметр распределения.

Характеристики распределения $G(p)$ $(x \varepsilon \{1,2,\dots\})$:

- функция распределения:
$$F_\xi(x) = 1 - q^x, q = 1 - p;$$
- функция вероятности:
$$P_\xi(x) = p \cdot q^{x-1};$$
- среднее значение: $\mu = 1/p$;
- дисперсия: $\sigma^2 = q/p^2$.

Алгоритм моделирования ДСВ ξ с законом распределения $G(p)$ состоит из двух шагов:

1. Моделирование реализации a БСВ.
2. Принятие решения о том, что реализацией ξ является значение x, определяемое соотношением:
$$x = [\ln a / \ln q],$$

здесь $[z]$ означает округление числа z в большую сторону до ближайшего целого значения.

Коэффициент использования БСВ $k = 1$.

χ^2 **- критерий согласия Пирсона:**

Область возможных значений случайной величины разбивается на интервалы $[x_{k-1}, x_k),\ k = \overline{1, K}$.

Рассматривается следующая статистика,
$$\chi^2 = \sum_{k=1}^{K} \frac{(\nu_k - n \cdot p_k)}{n \cdot p_k},$$

n – объем выборки,

ν_k - количество элементов выборки, попавших в k-ый интервал,

p_k - вероятность попадания случайной величины в k-ый интервал.

Проверяется условие $\chi^2 < \Delta$, где $\Delta = G^{-1}(1-\varepsilon)$, G функция распределения χ^2, ε - уровень значимости (обычно ε=0.05).

В данной работе отрезок [0;1] разбивался на 26 интервалов.

Эмпирическая функция распределения:

Эмпирической функцией распределения, построенной по выборке $X_1, \dots, X_n$ объёма n, называется случайная функция $\hat{F}(x)$:

$$\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{X_i \le x\}} = \frac{1}{n}\sum_{i=1}^{n} \theta(x - X_i),$$

где $\mathbf{1}_A$ - индикатор события, $\theta(x)$ - функция Хевисайда.

Код программы:

```
public class BernoulliDistribution : IDistribution
    {
        private double p;
        private RandomBSV random;


        public string Name => "Распределение Бернулли Bi(1, " + p + ")";


        public BernoulliDistribution(double p)
        {
            this.p = p;
            this.random = new RandomBSV(24149775, 19581355);
        }

        public double Mean()
        {
            return p;
        }

        public double Variance()
        {
            return p * (1 - p);
        }

        public double Next()
        {
            double a = random.Next();
            return (a > p) ? 0 : 1;
        }

        public void Refresh()
        {
            random.Refresh();
        }

        public double Skewness()
        {
            return (1 - 2 * p) / Math.Sqrt(p * (1 - p));
        }

        public double Kurtosis()
        {
            return (6 * p * p - 6 * p + 1) / (p * (1 - p));
        }

        public double PMF(int x)
        {
            return MathNet.Numerics.Distributions.Bernoulli.PMF(p, x);
        }

        public double CDF(int x)
        {
            return MathNet.Numerics.Distributions.Bernoulli.CDF(p, x);
        }
       }
```

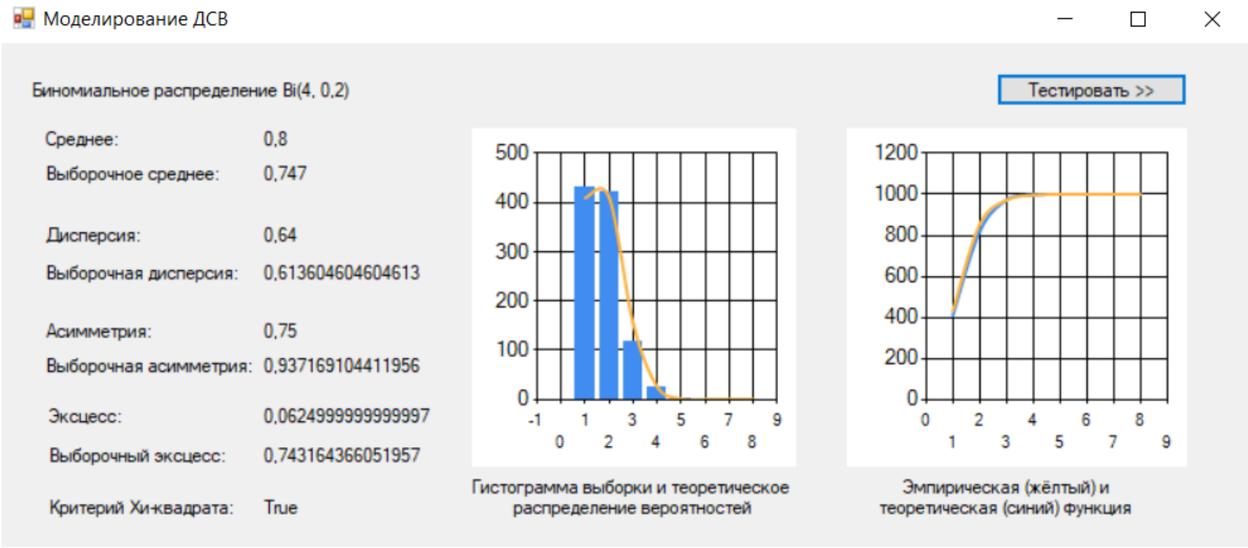
```
public class BinomialDistribution : IDistribution
    {
        private int m;
        private double p;

        private RandomBSV random;


        public string Name => "Биномиальное распределение Bi(" + m + ", " + p + ")";


        public BinomialDistribution(int m, double p)
        {
            this.m = m;
            this.p = p;
            this.random = new RandomBSV(24149775, 19581355);
        }

        public double Mean()
        {
            return m * p;
        }

        public double Variance()
        {
            return m * p * (1 - p);
        }

        public double Next()
        {
            int sum = 0;

            for (int i = 0; i < m; i++)
            {
                double a = random.Next();
                sum += (p - a > 0) ? 1 : 0;
            }

            return sum;
        }

        public void Refresh()
        {
            random.Refresh();
        }

        public double Skewness()
        {
            return (1 - 2 * p) / Math.Sqrt(m * p * (1 - p));
        }

        public double Kurtosis()
        {
            return (1 - 6 * p * (1 - p)) / (m * p * (1 - p));
        }

        public double PMF(int x)
        {
            return MathNet.Numerics.Distributions.Binomial.PMF(p, m, x);
        }

        public double CDF(int x)
        {
            return MathNet.Numerics.Distributions.Binomial.CDF(p, m, x);
        }
       }
```

```
public class GeometricDistribution : IDistribution
    {
        private double p;
        RandomBSV random;


        public string Name => "Геометрическое распределение G(" + p + ")";


        public GeometricDistribution(double p)
        {
            this.p = p;
            this.random = new RandomBSV(146051657, 1928884637);
        }

        public double Mean()
        {
            return 1 / p;
        }

        public double Variance()
        {
            return (1 - p) / (p * p);
        }

        public double Next()
        {
            return Math.Ceiling(Math.Log(random.Next()) / Math.Log(1 - p));
        }

        public void Refresh()
        {
            random.Refresh();
        }

        public double Skewness()
        {
            return (2 - p) / Math.Sqrt(1 - p);
        }

        public double Kurtosis()
        {
            return 6 + p * p / (1 - p);
        }

        public double PMF(int x)
        {
            return MathNet.Numerics.Distributions.Geometric.PMF(p, x);
        }

        public double CDF(int x)
        {
            return MathNet.Numerics.Distributions.Geometric.CDF(p, x);
        }
       }
public class DistributionTester
    {
        private int n;
        private List<double> sample;


        public DistributionTester(int sampleSize)
        {
            this.n = sampleSize;
        }
```

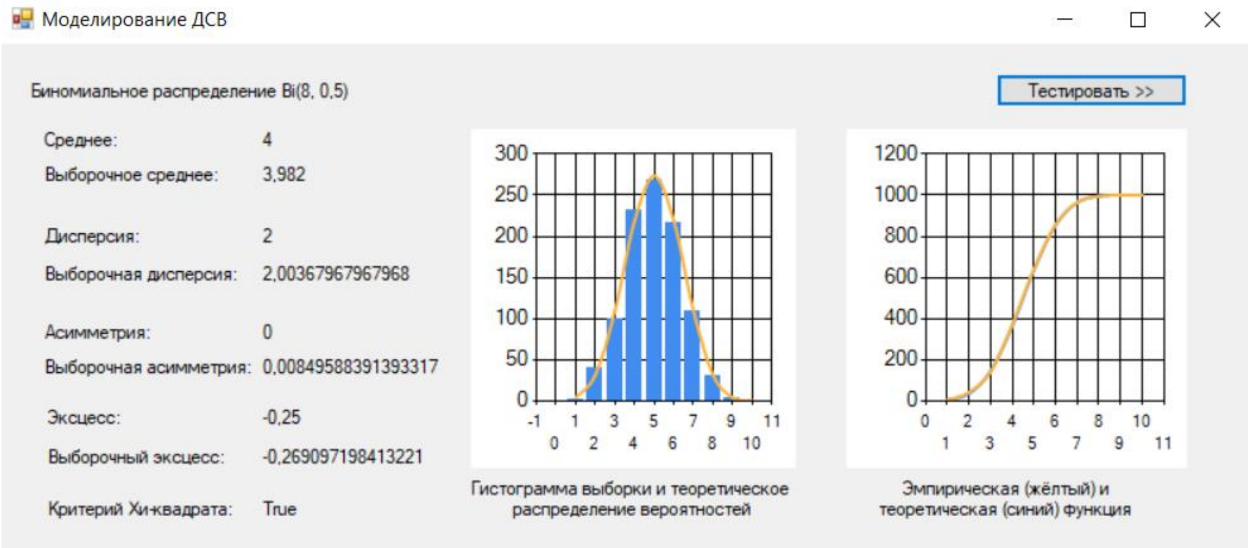
```
    public double SampleMean(IDistribution distribution)
    {
        sample = SampleGenerator.GenerateSample(distribution, n);

        double sum = 0;
        foreach (double x in sample)
            sum += x;

        return sum / n;
    }

    public double MomentAboutTheMean(IDistribution distribution, int k)
    {
        double xMean = SampleMean(distribution);

        double sum = 0;
        foreach (double x in sample)
            sum += Math.Pow(x - xMean, k);

        return sum / n;
    }

    public double SampleVariance(IDistribution distribution)
    {
        return MomentAboutTheMean(distribution, 2) * n / (n - 1);
    }

    public double SampleSkewness(IDistribution distribution)
    {
        double b = MomentAboutTheMean(distribution, 3);
        return b / Math.Pow(Math.Sqrt(SampleVariance(distribution)), 3);
    }

    public double SampleKurtosis(IDistribution distribution)
    {
        double g = MomentAboutTheMean(distribution, 4);
        return g / Math.Pow(SampleVariance(distribution), 2) - 3;
    }
   }

//Примеры построения на форме графиков из дополнительных заданий.
private void PaintChart()
    {
        chart.Series[0].Points.Clear();
        chart.Series["line"].Points.Clear();
        funcChart.Series["sampleFunc"].Points.Clear();
        funcChart.Series["func"].Points.Clear();

        chart.Series[0].ChartType = SeriesChartType.Column;
        chart.Series["line"].ChartType = SeriesChartType.Spline;
        chart.Series["line"].BorderWidth = 2;
        funcChart.Series["sampleFunc"].ChartType = SeriesChartType.Spline;
        funcChart.Series["sampleFunc"].BorderWidth = 2;
        funcChart.Series["func"].ChartType = SeriesChartType.Spline;
        funcChart.Series["func"].BorderWidth = 2;

        int size = freqSizes[distNum];
        int[] frequency = new int[size];

        for (int i = 0; i < sampleSize; i++)
        {
            int val = (int)(sample[i]);
            frequency[val]++;
        }

        for (int i = 0; i < size; i++)
```

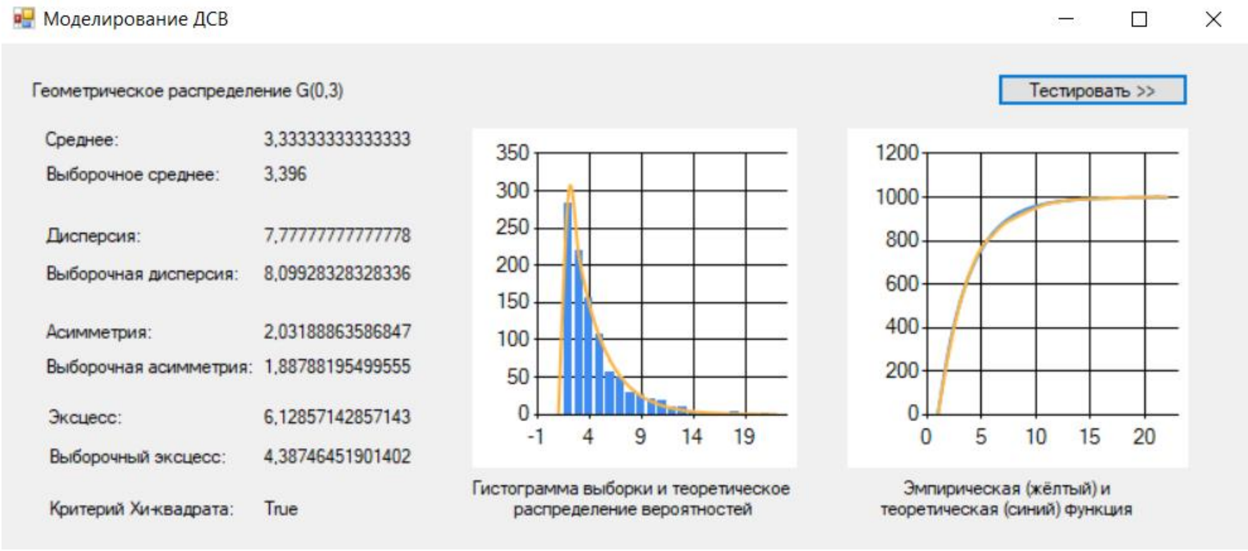
```
            {
                chart.Series[0].Points.AddY(frequency[i]);
                chart.Series["line"].Points.AddXY(i + 1, distributions[distNum].PMF(i) *
1000);
                funcChart.Series["sampleFunc"].Points.AddXY(i + 1, SamplePMF(sample, i +
1) * 1000);
                funcChart.Series["func"].Points.AddXY(i + 1,
distributions[distNum].CDF(i) * 1000);
            }
        }

        //Вычисление эмпирической функции распределения.

        private double SamplePMF(List<double> sample, double x)
        {
            double amount = 0.0;

            foreach(var element in sample)
            {
                if (element < x)
                    amount++;
            }

            return amount / sample.Count;
        }

        //Проверка гипотезы о совпадении распределений при помощи критерия согласия
Пирсона.

        private static int freeDeg = 26;
        private static double lvl = 37.7;

        private bool ChiSquared(List<double> sample)
        {
            int[] hits = new int[freeDeg];
            double chi2 = 0;
            double probability = 1.0 / freeDeg;

            for (int i = 0; i < sampleSize; i++)
            {
                int idx = (int)(sample[i] * freeDeg / freqSizes[distNum]);
                ++hits[idx];
            }

            for (int i = 0; i < freeDeg; i++)
                chi2 += ((double)hits[i] / sampleSize - probability)
                    * ((double)hits[i] / sampleSize - probability) / probability;

            return lvl > chi2;
           }
```

Результат:


Моделирование ДСВ
Биномиальное распределение Bi(4, 0,2)
Тестировать >>
Среднее: 0,8
Выборочное среднее: 0,747
Дисперсия: 0,64
Выборочная дисперсия: 0,613604604604613
Асимметрия: 0,75
Выборочная асимметрия: 0,937169104411956
Эксцесс: 0,0624999999999997
Выборочный эксцесс: 0,743164366051957
Критерий Хи-квадрата: True
Гистограмма выборки и теоретическое распределение вероятностей
Эмпирическая (жёлтый) и теоретическая (синий) функция


Моделирование ДСВ
Распределение Бернулли Bi(1, 0,2013)
Тестировать >>
Среднее: 0,2013
Выборочное среднее: 0,177
Дисперсия: 0,16077831
Выборочная дисперсия: 0,145816816816816
Асимметрия: 1,48988068042257
Выборочная асимметрия: 1,69002916580149
Эксцесс: 0,219744441896423
Выборочный эксцесс: 0,857058638898646
Критерий Хи-квадрата: True
Гистограмма выборки и теоретическое распределение вероятностей
Эмпирическая (жёлтый) и теоретическая (синий) функция


Моделирование ДСВ
Биномиальное распределение Bi(8, 0,5)
Тестировать >>
Среднее: 4
Выборочное среднее: 3,982
Дисперсия: 2
Выборочная дисперсия: 2,00367967967968
Асимметрия: 0
Выборочная асимметрия: 0,00849588391393317
Эксцесс: -0,25
Выборочный эксцесс: -0,269097198413221
Критерий Хи-квадрата: True
Гистограмма выборки и теоретическое распределение вероятностей
Эмпирическая (жёлтый) и теоретическая (синий) функция


Моделирование ДСВ
Геометрическое распределение G(0,3)
Тестировать >>
Среднее: 3,33333333333333
Выборочное среднее: 3,396
Дисперсия: 7,77777777777778
Выборочная дисперсия: 8,09928328328336
Асимметрия: 2,03188863586847
Выборочная асимметрия: 1,88788195499555
Эксцесс: 6,12857142857143
Выборочный эксцесс: 4,38746451901402
Критерий Хи-квадрата: True
Гистограмма выборки и теоретическое распределение вероятностей
Эмпирическая (жёлтый) и теоретическая (синий) функция

Лабораторная работа 3.

Условие:

1) Осуществить моделирование n = 10000 реализаций случайной величины из нормального закона распределения N(m, s2) с заданными параметрами. Для моделирования воспользоваться алгоритмом, основанным на ЦПТ; (в качестве количества используемых слагаемых можно взять N = 48, или 192, но должна быть возможность быстро изменить данный параметр). Вычислить несмещенные оценки математического ожидания и дисперсии, сравнить их с истинными значениями.

Вариант:

9) m = 0, s2 = 64;   10) m = 1, s2 = 9;

2) Смоделировать n = 10000 случайных величин из заданных абсолютно непрерывных распределений. Вычислить несмещенные оценки математического ожидания и дисперсии, сравнить их с истинными значениями (если это возможно). Если математического ожидания не существует, то вычислить выборочное значение медианы и сравнить его с теоретическим.

Вариант:

9) χ2-распределение с m степенями свободы, m = 4; Лапласа L(a), a = 2.

10) Логнормальное LN(m, s2), m = 1, s2 = 9.Экспоненциальное E(a), a = 2.

Дополнительные задания:

1) (1 балл) Смоделировать n = 10000 случайных величин из смеси двух распределений. Распределения взять из своего варианта задания 2, π – вероятность выбора элемента из первого распределения. Важно: В случае если у одного из распределений из вашего варианта математическое ожидание или дисперсия не существует, то заменить его на нормальное распределение из своего варианта задания 1.

2) (2 балла) Вычислить несмещенные оценки математического ожидания и дисперсии, сравнить их с истинными значениями (найти в литературе (интернете) или вывести самостоятельно формулы для нахождения математического ожидания и дисперсии смеси распределений).

Вариант:

9) π = 0.5;   10) π = 0.4;

3) (1 балл) Осуществить моделирование n = 10000 реализаций случайной величины из стандартного нормального закона распределения N(0, 1), используя преобразование Бокса — Мюллера

http://ru.wikipedia.org/wiki/Преобразование_Бокса_—_Мюллера (в источнике Лобач В.И. [и др] такой метод моделирования называется: моделирование, «используя функциональное преобразование»).

4) (1 балл) для смоделированной в бонусном пункте 3 выборки оценить коэффициент корреляции между элементами, стоящими на четных позициях, и элементах, стоящих на нечетных позициях.

Теория:

Одномерное нормальное распределение $N(\mu, \sigma^2)$:

Для моделирования распределения определим теоретическую базу, которая будет необходима. Одномерное нормальное распределение представляет собой НСВ с плотностью распределения следующего вида:

$$p_\xi(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{\frac{-(x-\mu)^2}{2\sigma^2}\right\}$$

Где μ и σ^2 – параметры распределения.

Из определения математического ожидания и дисперсии для НСВ не трудно показать, что выполняются следующие равенства:

$$E_\xi(x) = \mu, \qquad D_\xi(x) = \sigma^2$$

Также из свойств нормального распределения известно, что любое нормальное распределение вида $N(\mu, \sigma^2)$ выражается через стандартное нормальное распределение $N(0, 1)$, что и будем использовать при моделировании НСВ.

Функция распределения $N(0, 1)$ называется функцией Лапласа и имеет вид:

$$\Phi(x) = F_\eta(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} dt$$

Тогда случайные величины ξ и η связаны следующим соотношением:

$\xi = \mu + \sigma * \eta \qquad (1)$

Для моделирования распределений будем использовать датчики БСВ, полученные нами в лабораторной работе №1.

Алгоритм моделирования суммированием. Согласно заданию, сгенерируем N независимых БСВ. При $N \to \infty$ следующая формула позволит нам получить СВ, распределённую асимптотически нормально:

$$\xi = \sqrt{\frac{12}{N}}\left(\sum_{i=1}^{N} a_i - \frac{N}{2}\right)$$

Далее при помощи формулы (1) получим необходимое нам распределение вида $N(\mu, \sigma^2)$.

Преобразование Бокса-Мюллера:

Существует также альтернативный более точный способ моделирования НСВ с нормальным распределением. Для выполнения дополнительного задания 3) рассмотрим данный алгоритм моделирования.

Пусть x, y – независимые случайные величины, равномерно распределённые на отрезке [-1, 1]. Для моделирования данных величин будем использовать преобразование вида $2x - 1$ для смоделированной БСВ. Вычислим $s = x^2 + y^2$, если данная величина окажется больше 1 или равна 0, сгенерируем заново значения x, y.

Далее вычислим реализации искомой НСВ по формулам:

$$z_0 = x * \sqrt{\frac{-2\ ln\ s}{s}}, \qquad z_1 = y * \sqrt{\frac{-2\ ln\ s}{s}}$$

Для выполнения дополнительного задания 4) воспользуемся формулами для расчёта линейного коэффициента корреляции:

$$corr_{XY} = \frac{cov_{XY}}{\sigma_X \sigma_Y} = \frac{\sum(X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum(X - \bar{X})^2 (Y - \bar{Y})^2}}$$

Где $\bar{X}, \bar{Y}$ – средние значения выборок.

Итого, в результате вычисления коэффициента корреляции чётных и нечётных элементов выборки получаем, что он приближается к 0.

χ^2-распределение с m степенями свободы:

Распределение вида $\chi^2(m)$ представляет собой НСВ $\xi \epsilon [0, +\infty)$ с плотностью распределения:

$$p_\xi(x) = \frac{x^{\frac{m-2}{2}} e^{-\frac{s}{2}}}{2^{\frac{m}{2}} \Gamma\left(\frac{m}{2}\right)}, \qquad x \geq 0$$

При этом среднее значение и дисперсия равны:

$$\mu = m, \qquad \sigma^2 = 2m.$$

Основным свойством такого распределения является возможность получения его путём суммирования m квадратов независимых стандартных нормальных СВ:

$$\xi = \sum_{i=1}^{m} {\eta_i}^2 \qquad (2)$$

Алгоритм моделирования. Таким образом для моделирования будем использовать свойство (2). Сгенерируем m независимых реализаций $N(0,1\,)$ и просуммируем их квадраты. Далее примем решение о том, что полученное значение является реализацией искомой НСВ.

Распределение Лапласа *L(a)*:

Говорят, что НСВ имеет распределение Лапласа вида *L(a)*, если НСВ имеет плотность распределения:

$$p_\xi(x) = \frac{a}{2} e^{-a|x|}$$

Математически показывается, что для такого распределения среднее значение и дисперсия будут иметь следующие значения:

$$\mu = 0, \qquad \sigma^2 = 2/a^2$$

Алгоритм моделирования. Моделирование данного распределения основано на методе обратной функции. Для начала смоделируем реализацию БСВ y. Данное значение будем использовать для принятия итогового решения.

Обратная для $F_\xi(x)$ функция будет иметь вид:

$$x = {F_\xi}^{-1}(x) = \frac{1}{a}\ln(2y)\,, \;\; y \,\epsilon\, [0, 0.5)$$

$$x = {F_\xi}^{-1}(x) = \frac{-1}{a}\ln(2(1-y))\,, \;\; y \,\epsilon\, [0.5, 1)$$

Логнормальное распределение *LN(μ, σ^2)*:

Логарифмически-нормальным называют распределение НСВ с плотностью вида:

$$p_\xi(x) = \frac{1}{x\sqrt{2\pi}\sigma} exp\left\{-\frac{(\ln x - \mu)^2}{2\sigma^2}\right\}$$

Среднее значение и дисперсия такого распределения определяются формулами:

$$E_\xi(x) = \mu\sqrt{\exp\{\sigma^2\}}, \qquad D_\xi(x) = \mu^2 \exp\{\sigma^2\}(\exp\{\sigma^2\} - 1)$$

Как очевидно из названия, данное распределение напрямую связано с нормальным распределением, а именно: искомая НСВ будет равна экспоненте нормальной НСВ.

Алгоритм моделирования. Воспользуемся приведённым выше свойством и смоделируем нормально распределённую случайную величину с необходимыми нам параметрами μ, σ^2. Далее получим реализацию искомой НСВ, взяв от смоделированной величины экспоненту.

Экспоненциальное распределение *E(a)*:

Плотность распределения экспоненциальной НСВ имеет следующий вид:

$$p_\xi(x) = ae^{-ax}$$

Нетрудно показать, что среднее значение и дисперсия для такого распределения примут вид:

$$\mu = \frac{1}{a}, \qquad \sigma^2 = \frac{1}{a^2}$$

Алгоритм моделирования. Для моделирования такого распределения воспользуемся методом обратной функции. Опеределим $F_\xi(x)$ и обратную к ней функцию:

$$F_\xi(x) = 1 - e^{-ax}$$

$$x = F^{-1}{}_\xi(y) = -\frac{\ln(1-y)}{a}$$

Алгоритм моделирования будет заключаться в моделировании некоторой реализации БСВ x, для которой будет вычисляться реализация искомой НСВ вида:

$$x = -\frac{1}{a}\ln x$$

Смесь двух распределений с вероятностью π:

Смесь двух распределений для выполнения задания будем моделировать следующим образом.

Будем генерировать реализацию БСВ a, расположенную на отрезке [0, 1]. Далее по этой реализации будем принимать решение, какое из распределений реализует искомую СВ. Если сгенерированное a не превосходит π – вероятности выбора первого из двух распределений, реализацией СВ будет первое распределение, иначе – второе.

Для выполнения дополнительного задания 2) воспользуемся следующими формулами для вычисления среднего значения и дисперсии такого распределения:

$$E_\xi(x) = \mu = \sum_{i-1}^{n} w_i\mu_i\,, \qquad D_\xi(x) = \sum_{i-1}^{n} w_i(\sigma_i^2 + \mu_i{}^2 - \mu^2)$$

Где n – количество распределений в смеси (в нашем случае 2), w_i – вероятность выбора i-го распределения из смеси (в нашем случае для первого распределения это π, а для второго – $(1-\pi)$), μ_i – среднее значение i-го распределения из смеси, σ_i^2 – дисперсия i-го распределения из смеси, μ – среднее значение смеси распределений.

Код программы:

```
public class Normal : IDistribution
    {
        public int N
        {
            get;
            set;
        } = 48;

        private RandomBSV random;

        private double m, s;


        public Normal() : this(0.0, 1.0)
        { }
```

```
        public Normal(double mean, double stdDeviation)
        {
            m = mean;
            s = stdDeviation;
            random = new RandomBSV(24149775, 19581355);
        }

        public string Name => "Нормальное распределение N("
            + m.ToString() + ", " + s.ToString() + ")";

        public double Mean()
        {
            return m;
        }

        public double Variance()
        {
            return s;
        }

        public double Next()
        {
            return Next(random);
        }

        public double Next(RandomBSV rand)
        {
            List<double> a = SampleGenerator.GenerateSample(rand, N);

            double x = 0.0;

            foreach (double element in a)
                x += element;

            x -= N / 2;
            x *= Math.Sqrt(12.0 / N);

            return m + Math.Sqrt(s) * x;
        }

        public void Refresh()
        {
            random.Refresh();
        }
       }


public class BoxMullerNormal : IDistribution
    {
        private double m;

        private double s;

        private RandomBSV random;

        private double saved;

        private bool isSaved;


        public BoxMullerNormal(double m, double stdDeviation)
        {
            this.m = m;
            this.s = stdDeviation;

            isSaved = false;
```

```
            random = new RandomBSV(24149775, 19581355);
        }

        public string Name => "Нормальное распределение через преобразование Бокса-
Мюллера N("
            + m.ToString() + ", " + s.ToString() + ")";

        public double Mean()
        {
            return m;
        }

        public double Variance()
        {
            return s;
        }

        public double Next()
        {
            if (isSaved)
            {
                isSaved = false;
                return saved;
            }

            double
                x = 1.0,
                y = 1.0,
                s = 1.0;

            do
            {
                x = 2 * random.Next() - 1;
                y = 2 * random.Next() - 1;
                s = x * x + y * y;
            } while (s > 1 || s == 0);

            double
                z0 = x * Math.Sqrt(-2.0 * Math.Log(s) / s),
                z1 = y * Math.Sqrt(-2.0 * Math.Log(s) / s);

            saved = z1;
            isSaved = true;

            return z0;
        }

        public void Refresh()
        {
            random.Refresh();
        }
       }


public class ChiSquared : IDistribution
    {
        private double m;

        private RandomBSV random;


        public ChiSquared(double m)
        {
            this.m = m;
            random = new RandomBSV(179029053, 457816087);
        }
```

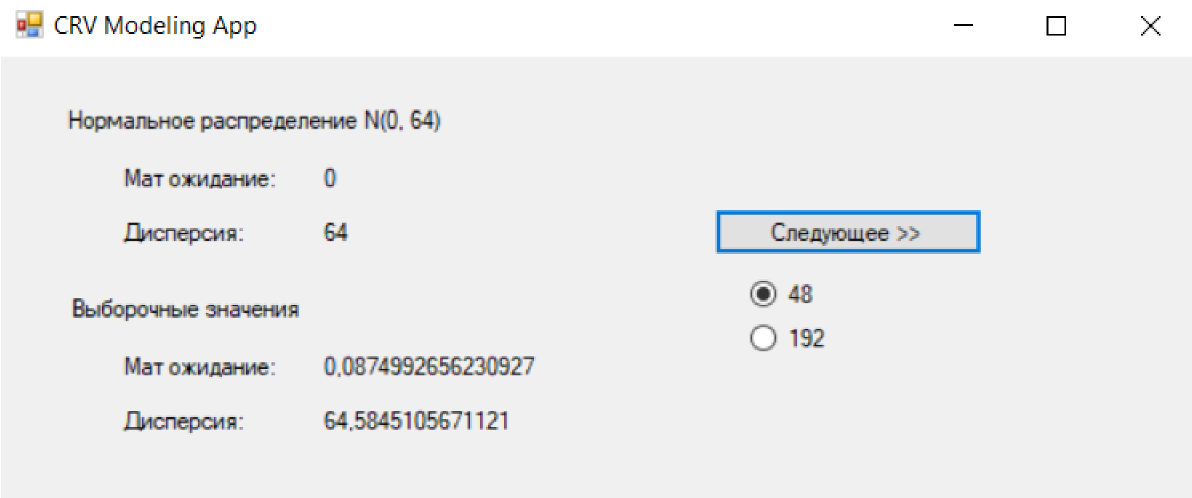
```
        public string Name => "Распределение Хи-квадрат с "
            + m.ToString() + " степенями свободы";

        public double Mean()
        {
            return m;
        }

        public double Variance()
        {
            return 2.0 * m;
        }

        public double Next()
        {
            double sum = 0.0;
            Normal n = new Normal();

            for (int i = 0; i < m; i++)
                sum += Math.Pow(n.Next(random), 2);

            return sum;
        }

        public void Refresh()
        {
            random.Refresh();
        }
       }


public class Laplace : IDistribution
    {
        private double a;

        private RandomBSV random;


        public Laplace(double a)
        {
            this.a = a;
            random = new RandomBSV(179029053, 457816087);
        }

        public string Name => "Распределение Лапласа L(" + a.ToString() + ")";

        public double Mean()
        {
            return 0.0;
        }

        public double Variance()
        {
            return 2.0 / a / a;
        }

        public double Next()
        {
            double y = random.Next();

            if (y < 0.5)
                return 1.0 / a * Math.Log(2 * y);
            else
                return -1.0 / a * Math.Log(2 * (1 - y));
        }
```

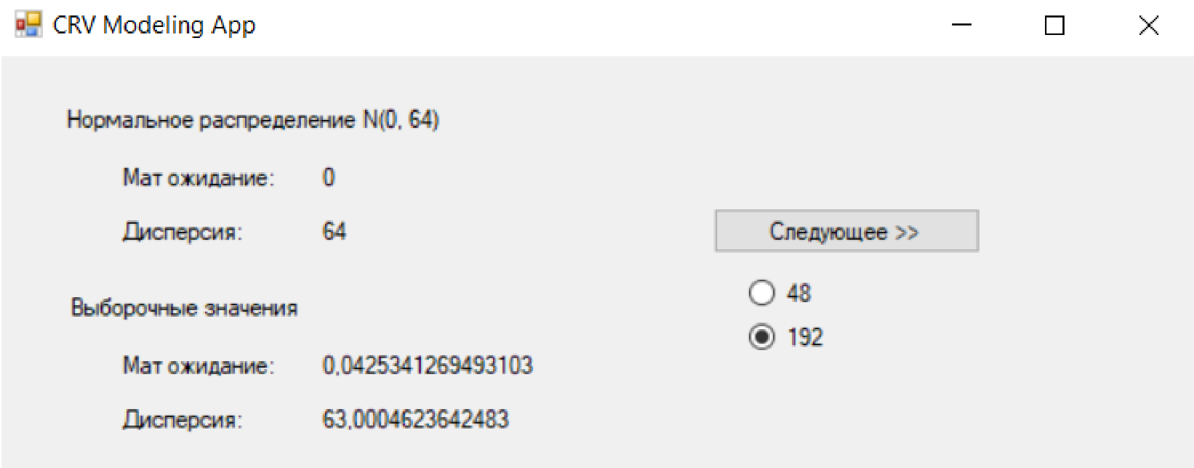
```
        public void Refresh()
        {
            random.Refresh();
        }
       }


public class LogNormal : IDistribution
    {
        private double m;

        private double s;

        private RandomBSV random;


        public LogNormal(double m, double s)
        {
            this.m = m;
            this.s = s;

            random = new RandomBSV(179029053, 457816087);
        }

        public string Name => "Лог-нормальное распределение LN("
            + m.ToString() + ", " + s.ToString() + ")";

        public double Mean()
        {
            return Math.Exp(m + s / 2);
        }

        public double Variance()
        {
            return (Math.Exp(s) - 1) * Math.Exp(2 * m + s);
        }

        public double Next()
        {
            Normal n = new Normal(m, s);
            return Math.Exp(n.Next(random));
        }

        public void Refresh()
        {
            random.Refresh();
        }
       }


public class Exponential : IDistribution
    {
        private double a;

        private RandomBSV random;


        public Exponential(double a)
        {
            this.a = a;
            random = new RandomBSV(179029053, 457816087);
        }

        public string Name => "Экспоненциальное распределение E("
            + a.ToString() + ")";
```

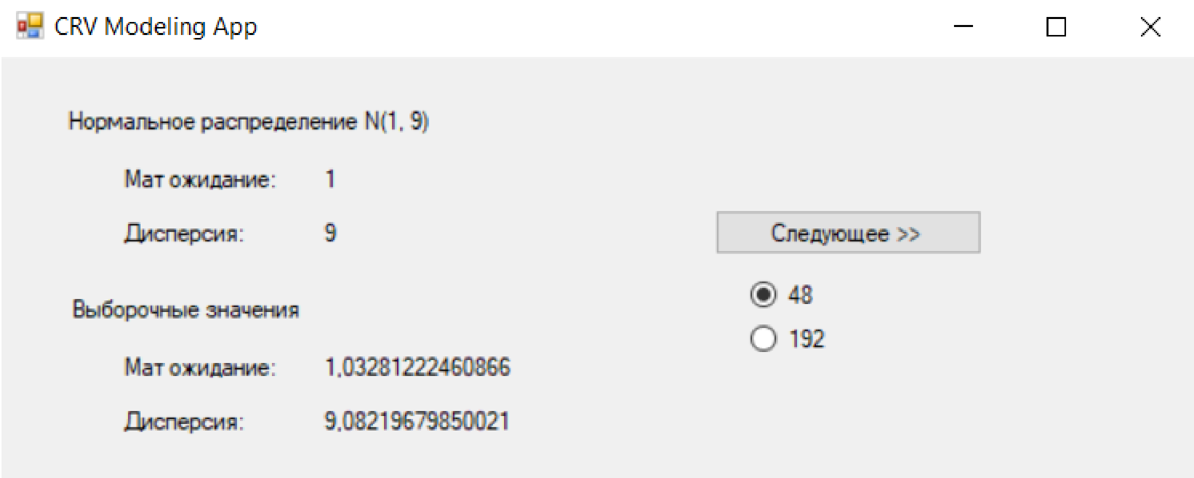
```
        public double Mean()
        {
            return 1.0 / a;
        }

        public double Variance()
        {
            return 1.0 / a / a;
        }

        public double Next()
        {
            double x = 0.0;

            do
            {
                x = random.Next();
            } while (x == 0.0);

            return -Math.Log(x) / a;
        }

        public void Refresh()
        {
            random.Refresh();
        }
       }


public class MixtureDistribution : IDistribution
    {
        private RandomBSV random;

        private IDistribution[] distributions;

        private double[] pi;


        public MixtureDistribution(IDistribution[] distributions, double[] pi)
        {
            this.distributions = distributions;
            this.pi = pi;

            random = new RandomBSV(179029053, 457816087);
        }

        public string Name
        {
            get
            {
                StringBuilder name = new StringBuilder("Mixure distribution: ");

                for (int i = 0; i < distributions.Length; i++)
                    name.Append(distributions[i].Name + " (" + pi[i] + ") ");

                return name.ToString();
            }
        }

        public double Mean()
        {
            double m = 0.0;

            for (int i = 0; i < distributions.Length; i++)
                m += pi[i] * distributions[i].Mean();
```

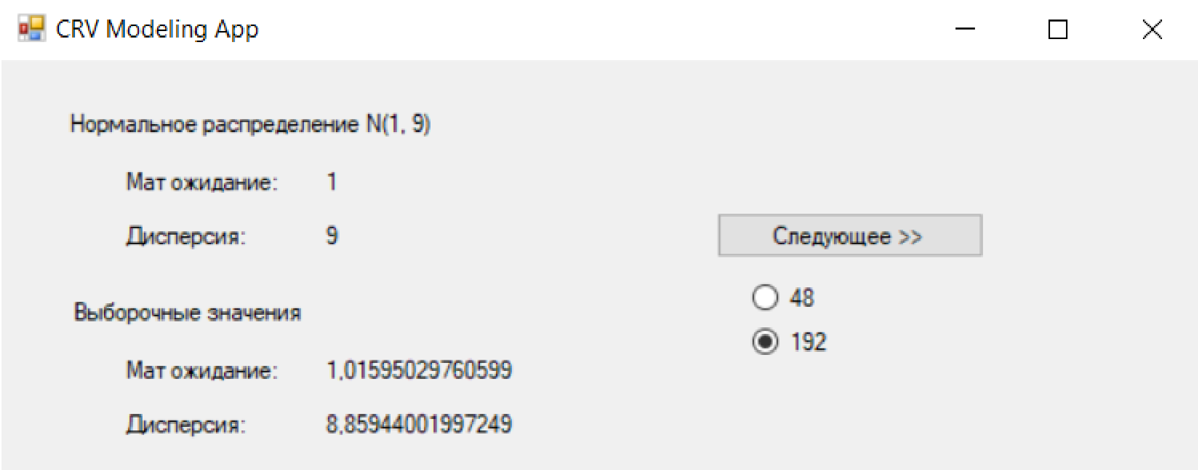
```
        return m;
    }

    public double Variance()
    {
        double
            v = 0.0,
            m = Mean();

        for (int i = 0; i < distributions.Length; i++)
            v += pi[i] * (distributions[i].Variance() +
                Math.Pow(distributions[i].Mean(), 2) - m * m);

        return v;
    }

    public double Next()
    {
        double a = random.Next();

        if (a < pi[0])
            return distributions[0].Next();
        else
            return distributions[1].Next();
    }

    public void Refresh()
    {
        random.Refresh();
    }
}
```

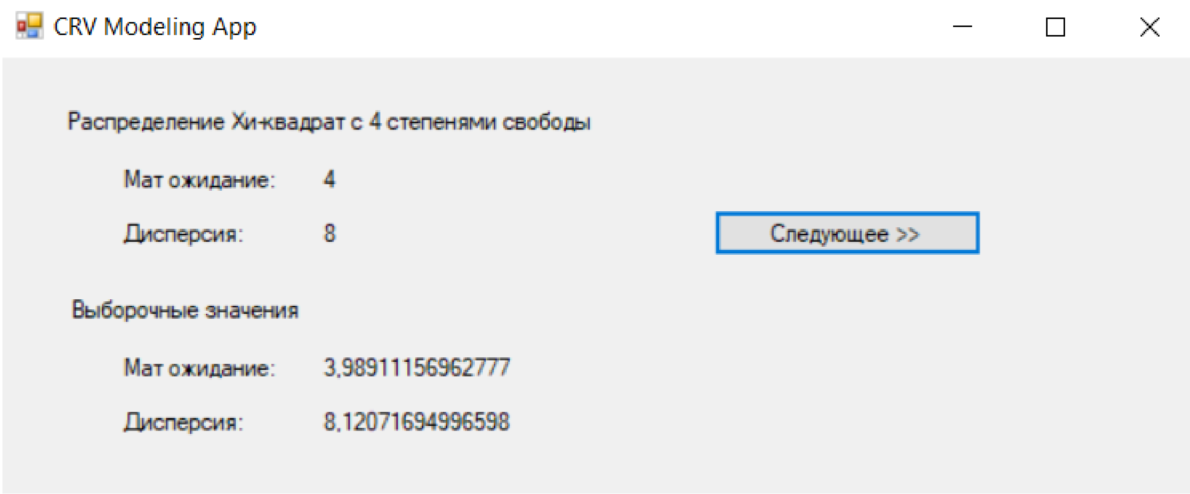
Результат:

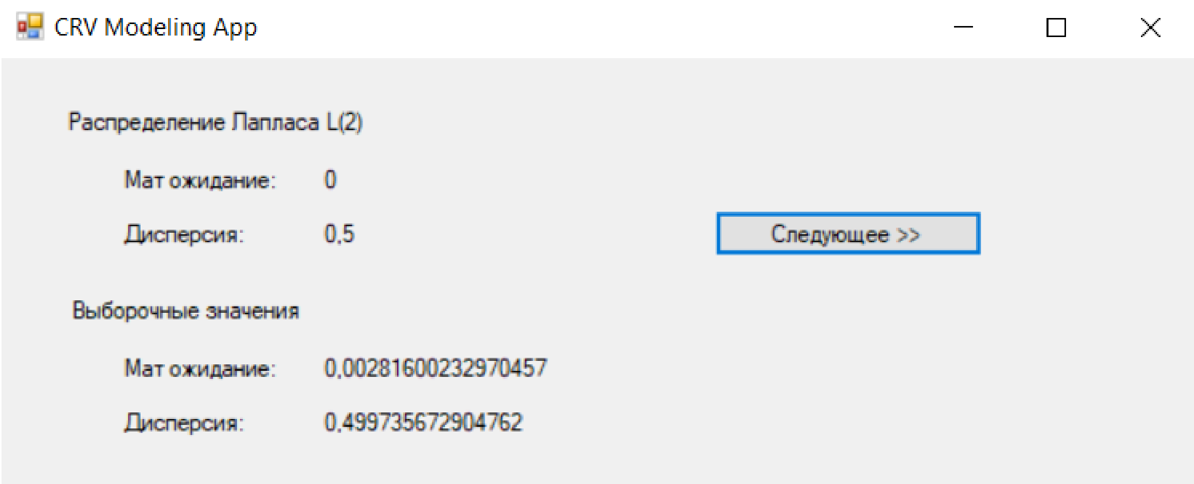


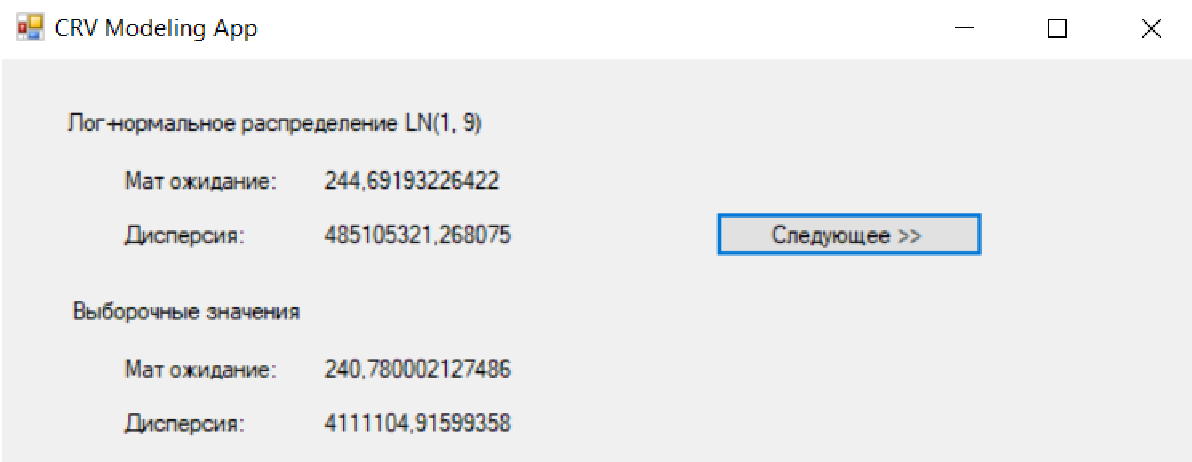

CRV Modeling App
Нормальное распределение N(0, 64)
Мат ожидание: 0
Дисперсия: 64
Следующее >>
48
192
Выборочные значения
Мат ожидание: 0,0425341269493103
Дисперсия: 63,0004623642483

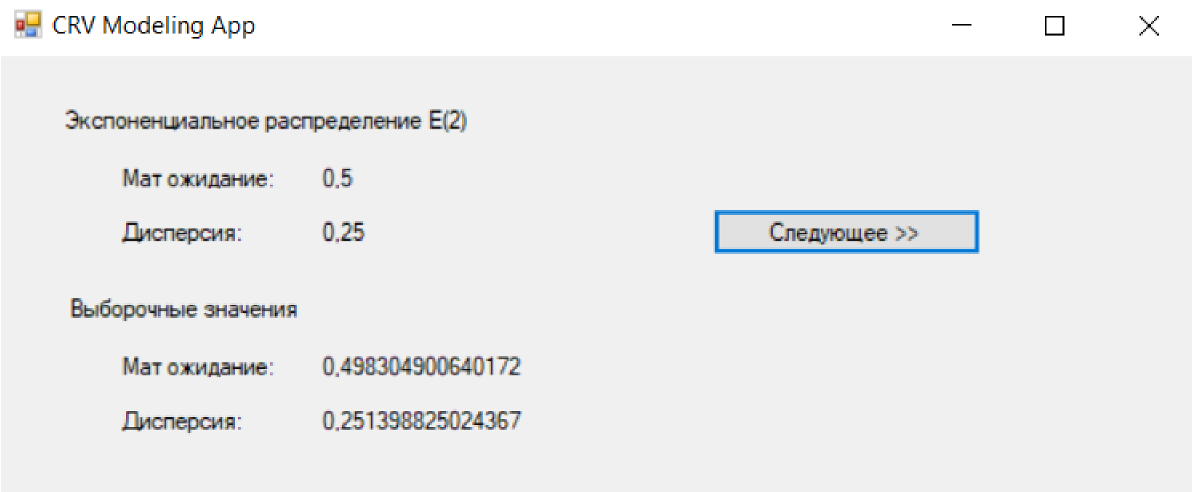

CRV Modeling App
Нормальное распределение N(1, 9)
Мат ожидание: 1
Дисперсия: 9
Следующее >>
48
192
Выборочные значения
Мат ожидание: 1,03281222460866
Дисперсия: 9,08219679850021

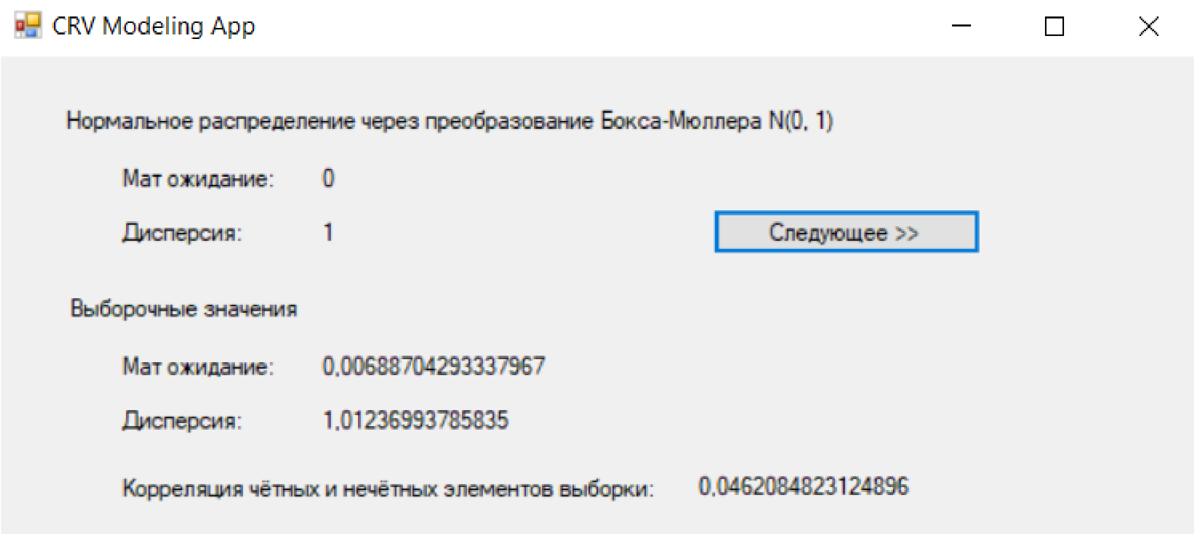

CRV Modeling App
Нормальное распределение N(1, 9)
Мат ожидание: 1
Дисперсия: 9
Следующее >>
48
192
Выборочные значения
Мат ожидание: 1,01595029760599
Дисперсия: 8,85944001997249

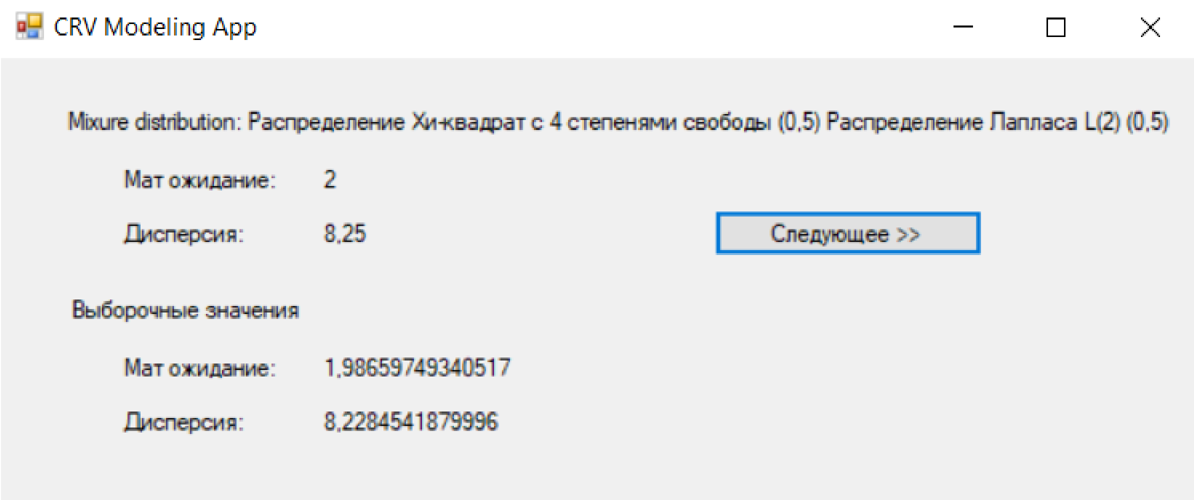

CRV Modeling App
Распределение Хи-квадрат с 4 степенями свободы
Мат ожидание: 4
Дисперсия: 8
Следующее >>
Выборочные значения
Мат ожидание: 3,98911156962777
Дисперсия: 8,12071694996598

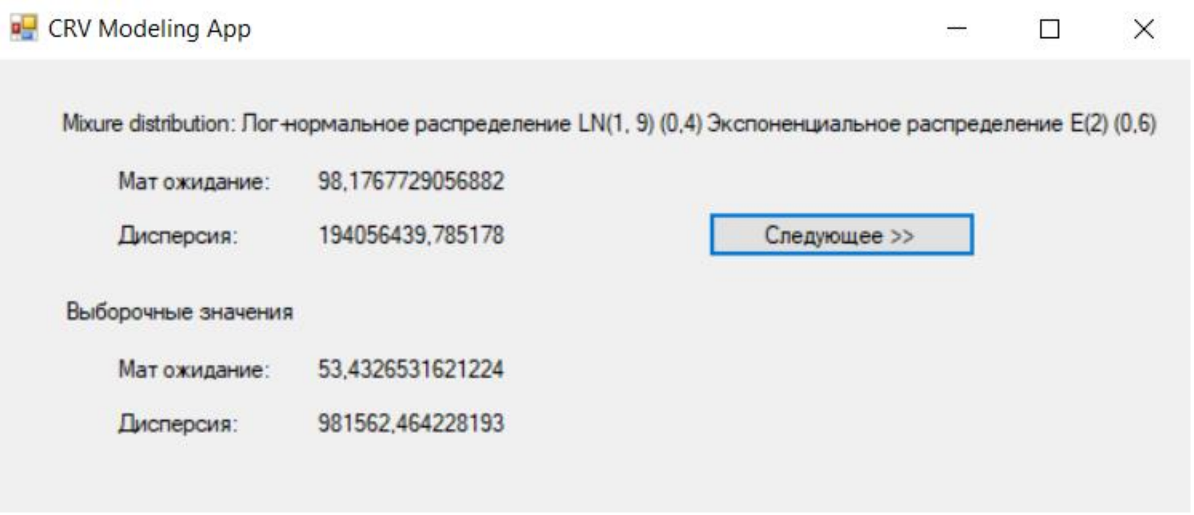

CRV Modeling App
Распределение Лапласа L(2)
Мат ожидание: 0
Дисперсия: 0,5
Следующее >>
Выборочные значения
Мат ожидание: 0,00281600232970457
Дисперсия: 0,499735672904762


CRV Modeling App
Лог-нормальное распределение LN(1, 9)
Мат ожидание: 244,69193226422
Дисперсия: 485105321,268075
Следующее >>
Выборочные значения
Мат ожидание: 240,780002127486
Дисперсия: 4111104,91599358


CRV Modeling App
Экспоненциальное распределение E(2)
Мат ожидание: 0,5
Дисперсия: 0,25
Следующее >>
Выборочные значения
Мат ожидание: 0,498304900640172
Дисперсия: 0,251398825024367


CRV Modeling App
Нормальное распределение через преобразование Бокса-Мюллера N(0, 1)
Мат ожидание: 0
Дисперсия: 1
Следующее >>
Выборочные значения
Мат ожидание: 0,00688704293337967
Дисперсия: 1,01236993785835
Корреляция чётных и нечётных элементов выборки: 0,0462084823124896


CRV Modeling App
Mixure distribution: Распределение Хи-квадрат с 4 степенями свободы (0,5) Распределение Лапласа L(2) (0,5)
Мат ожидание: 2
Дисперсия: 8,25
Следующее >>
Выборочные значения
Мат ожидание: 1,98659749340517
Дисперсия: 8,2284541879996


CRV Modeling App
Mixure distribution: Лог-нормальное распределение LN(1, 9) (0,4) Экспоненциальное распределение E(2) (0,6)
Мат ожидание:
98,1767729056882
Дисперсия:
194056439,785178
Следующее >>
Выборочные значения
Мат ожидание:
53,4326531621224
Дисперсия:
981562,464228193

Лабораторная работа 4.1.

Условие:

Вычислить интеграл методом Монте-Карло:

$$\int_{88}^{99} \ln(x)\sin(x)dx$$

Теория:

Метод Монте-Карло приближенного вычисления интеграла:

Необходимо вычислить $\int_{x_0}^{x_1} g(x)dx$.

Пусть η - произвольная случайная величина с плотностью распределения $P_\eta(x),\ x \in [x_0, x_1],$ имеющая конечный момент второго порядка.

Пусть $\xi = \dfrac{g(\eta)}{P_\eta(\eta)}$. Тогда $M\{\xi\} = a,\ D\{\xi\} < \infty.$

В качестве приближенного значения a можно взять

$$\frac{1}{n}\sum_{i=1}^{n}\xi_i = \frac{1}{n}\sum_{i=1}^{n}\frac{g(\eta_i)}{P_\eta(\eta_i)}.$$

В данной работе в качестве η бралась случайная величина, равномерно распределенная на [88; 99].

Ожидаемый результат вычисления:

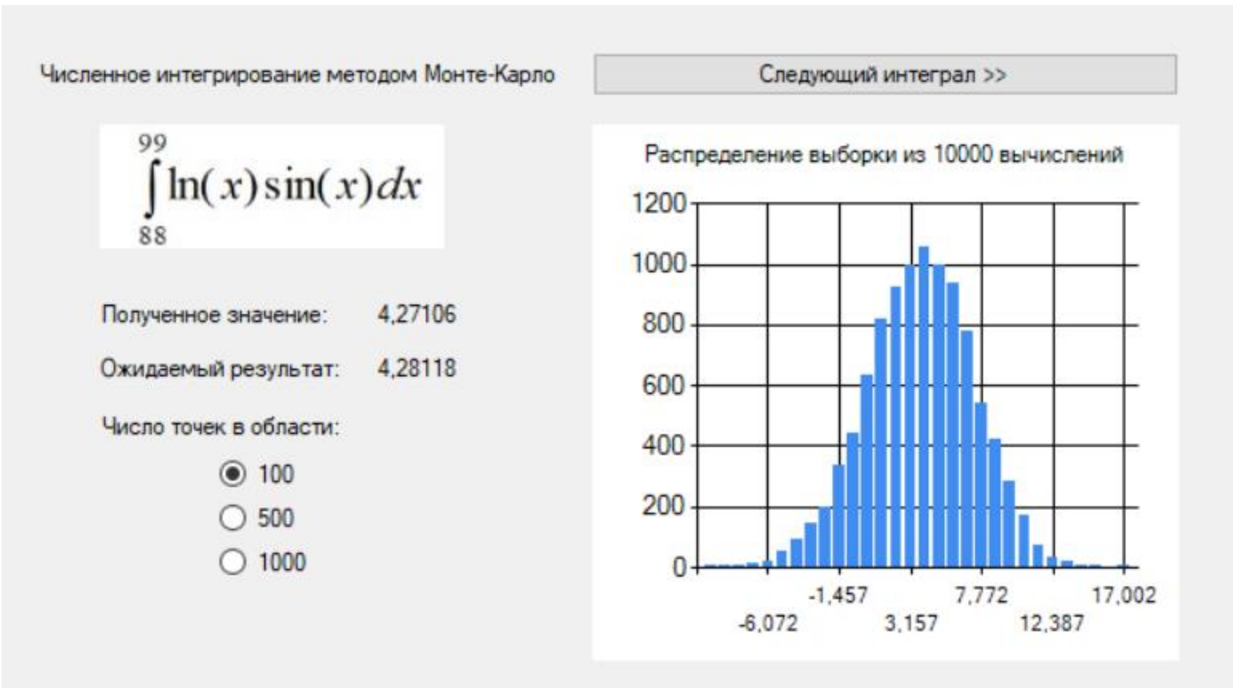
Данный интеграл точно вычислить не представляется возможным. При помощи онлайн-сервиса WolframAlfa получим приближённое численное значение, которое будем считать эталонным.

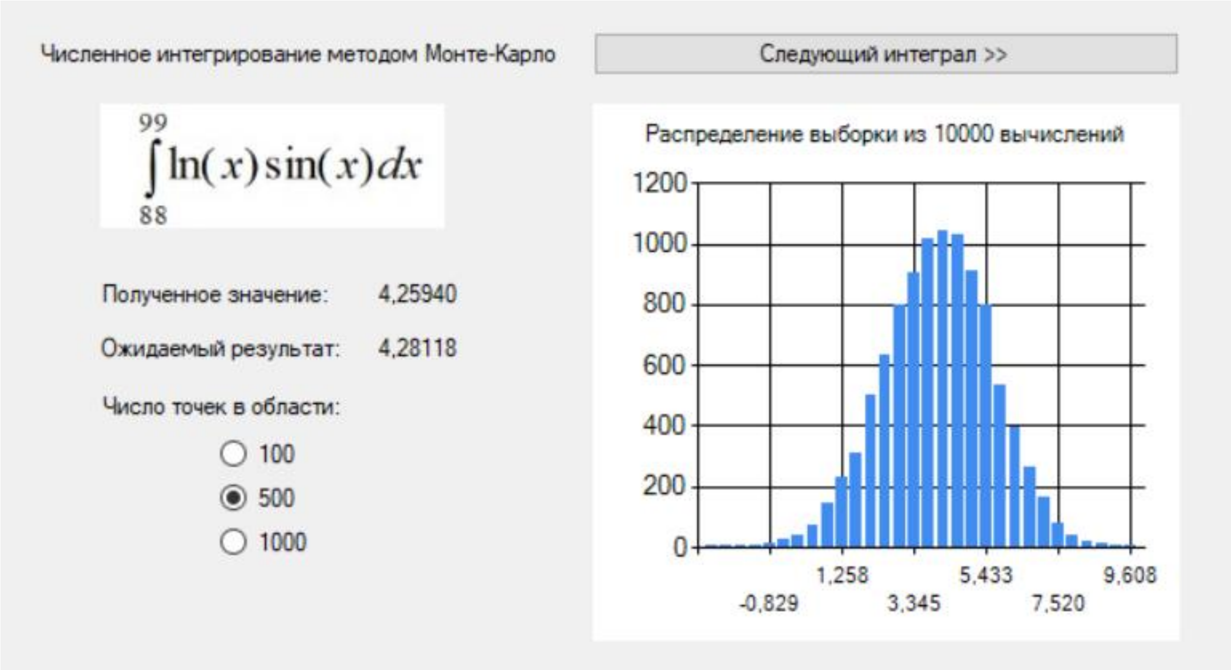
Код программы:

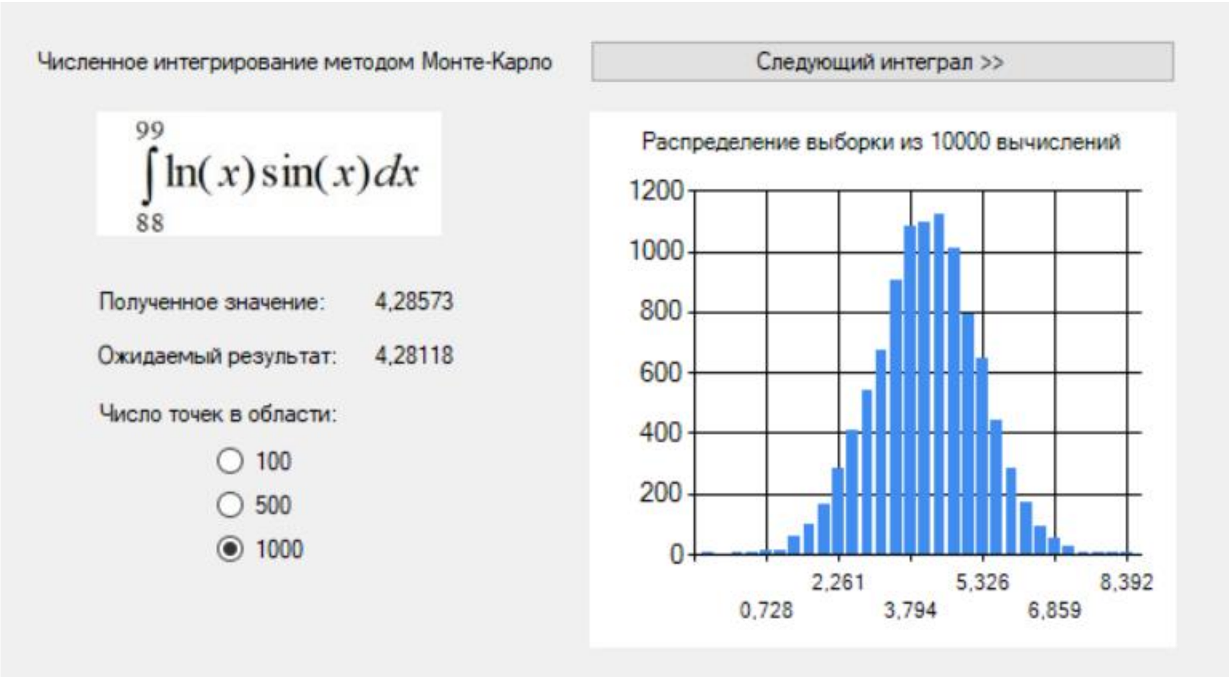
```
public class MCIntegrator : IIntegrator
    {
        //Класс, вычисляющий одномерный интеграл.

        private Random random;

        private double upperBound;
        private double lowerBound;

        private Func<double, double> func;

        //Принимает на вход генератор БСВ, подынтегральную функцию и пределы
                                                            интегрирования.
        public MCIntegrator(Random random,
            Func<double, double> integrand,
            double upperBound, double lowerBound)
        {
            func = integrand;
            this.random = random;

            this.upperBound = upperBound;
            this.lowerBound = lowerBound;
        }

        //Принимает число n точек для генерации.

        public double Integrate(int pointsCount)
        {
            double delta = upperBound - lowerBound;
            double sum = 0;

            for (int i = 0; i < pointsCount; i++)
```

```
            {
                double x = random.NextDouble() * delta + lowerBound;
                sum += func(x);
            }

            return sum * delta / pointsCount;
        }
    }

...

//Пример вызова вычислений интеграла согласно варианту в головной программе.

int pointsCount = 1000;

Func<double, double> firstIntegrand =
                new Func<double, double>((double x) =>
                {
                    return Math.Log(x) * Math.Sin(x);
                });

IIntegrator integral = new MCIntegrator(
                new Random(), firstIntegrand, 99.0, 88.0);

double result = integral.Integrate(pointsCount);
```

Результат работы программы:







В данном случае программа выполняет 10000 вычислений интеграла при помощи указанного метода класса и выводит среднее полученное значение.

На гистограмме можем наблюдать распределение полученных значений численного интегрирования. Данное распределение приближается к нормальному с увеличением числа точек для вычисления усреднённого интеграла и/или увеличением числа повторений вычисления. Отсюда можно сделать вывод, что точность вычисления методом Монте-Карло зависит от числа генерируемых СВ, что также было показано в примере в пособии.

Т.к. значения интеграла в результате 10000 повторных вычислений распределились нормально, можно утверждать, что полученное решение будет находиться в 3σ окрестности от истинного значения с вероятностью не менее 8/9.

Лабораторная работа 4.2.

Условие:

Вычислить интеграл методом Монте-Карло:

$$\iint\limits_{|x|+|y|<1} \frac{x^3+3xy}{e^{-y}}\,dx\,dy$$

Теория:

Теоретический минимум был описан в предыдущем пункте. Для случая двойного интеграла определим следующие изменения:

Чтобы упростить ход решения и сделать возможным использование равномерного распределения, впишем область интегрирования, представляющую ромб с вершинами в точках (-1,1) (1,1) (-1,-1) (1,-1), в квадрат с центром в (0, 0) и стороной 2.

В данном случае будем использовать СВ η, равномерно распределённую на области $[-1,1]^2$, и индикаторную функцию $\mathbb{1}(x, y)$ вида:

$$\begin{cases} 1, & |x|+|y|<1 \\ 0, & \text{иначе} \end{cases}$$

определяющую, попала ли сгенерированная точка в ромб.

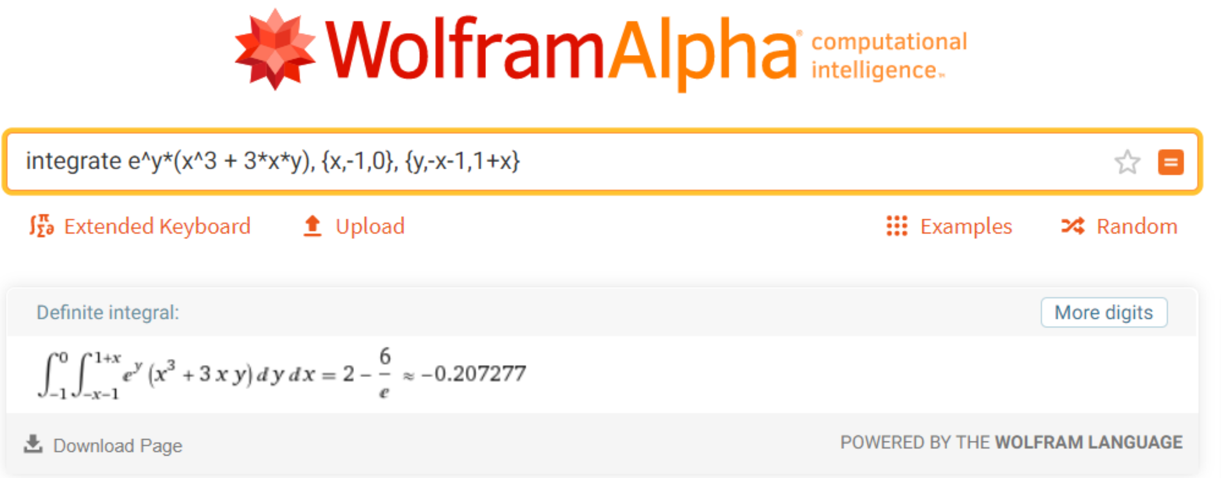
Таким образом, т.к. площадь ромба соотносится с площадью квадрата, в котором происходит генерация, как 1:2, можем утверждать, что в среднем половина из сгенерированных нами точек окажется в ромбе. Чтобы устранить данную вероятность будем генерировать в 2 раза больше точек для отыскания значения интеграла.

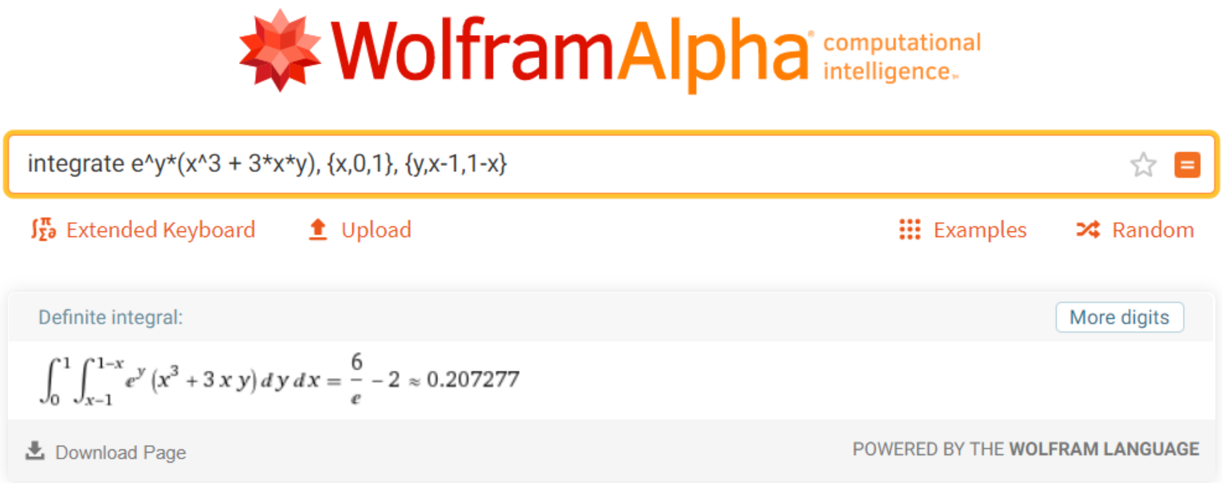
Ожидаемый результат вычисления:

Данный интеграл можно точно вычислить путём разбиения на сумму двух повторных интегралов вида:

$$\iint\limits_{|x|+|y|<1} \frac{x^3+3xy}{e^{-y}}\,dx\,dy = \int\limits_{-1}^{0} dx \int\limits_{-x-1}^{x+1} \frac{x^3+3xy}{e^{-y}}dy + \int\limits_{0}^{1} dx \int\limits_{x-1}^{1-x} \frac{x^3+3xy}{e^{-y}}dy$$

При помощи онлайн-сервиса WolframAlfa получим численные значения данных повторных интегралом, суммируем их, а полученное значение будем считать эталонным.





Итого ожидаемое значение 0.0.

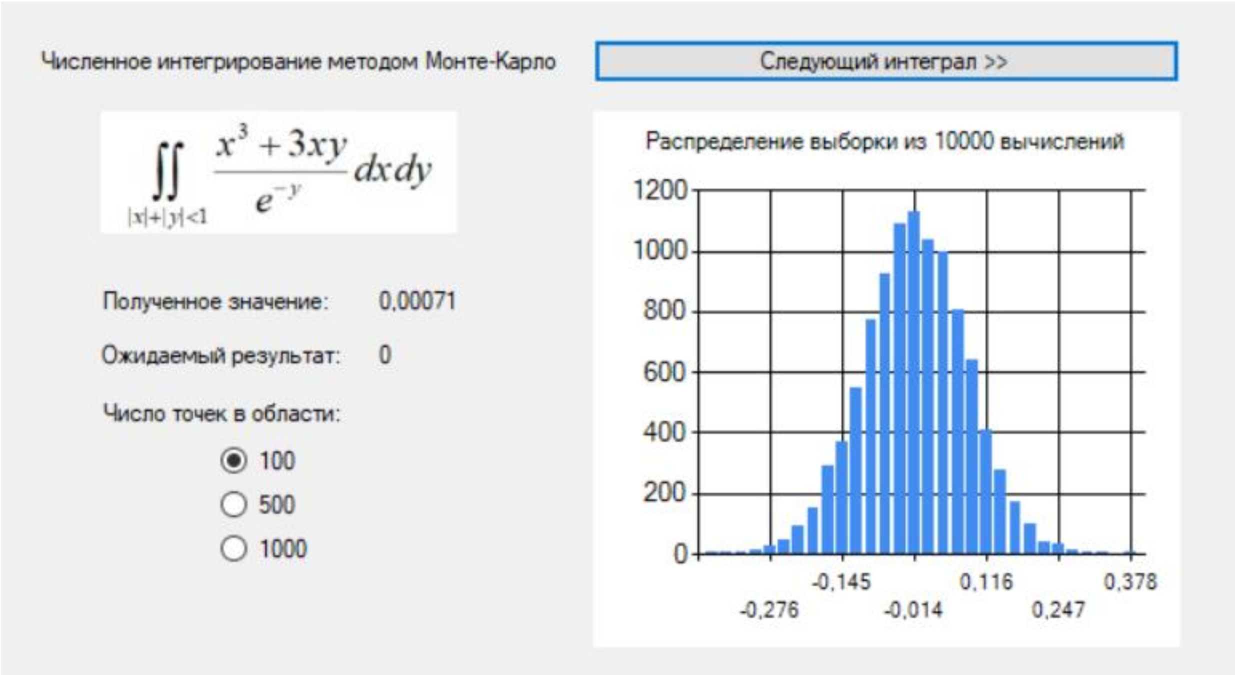
Код программы:

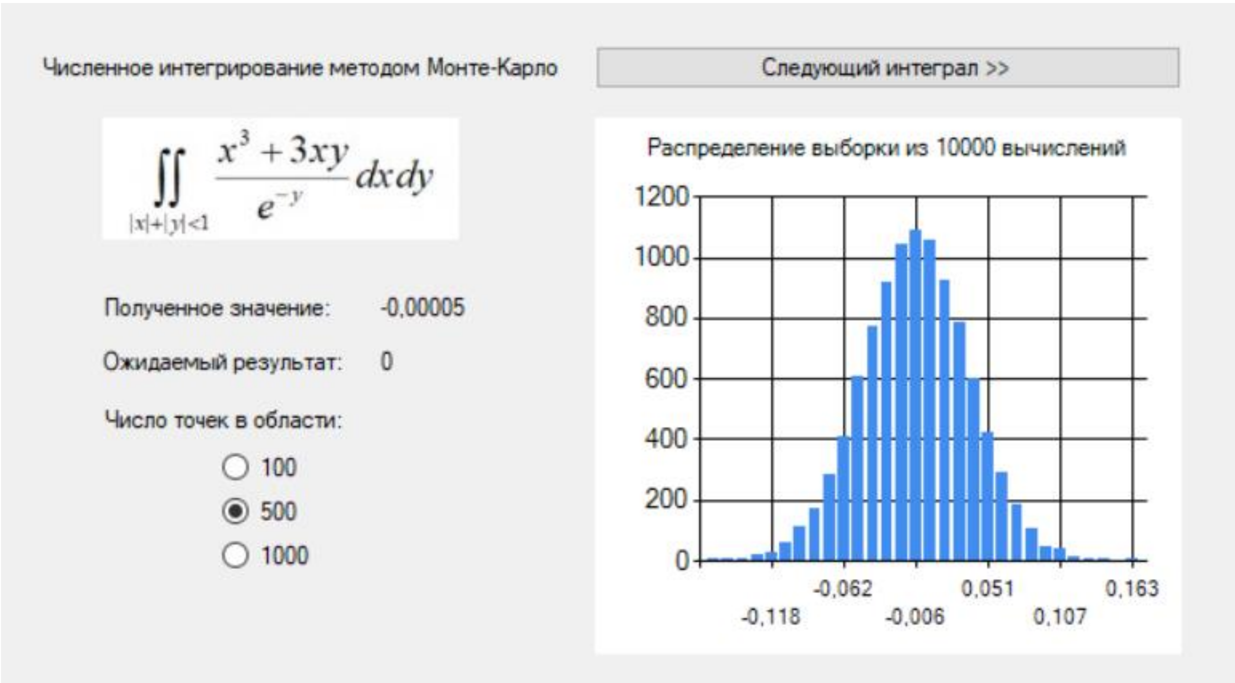
```
public class MCTwoDimensionalIntegrator : IIntegrator
    {
        //Класс, вычисляющий двойной интеграл.

        private Random random;

        private double upperXBound;
        private double upperYBound;

        private double lowerXBound;
        private double lowerYBound;

        private Func<double, double, double> func;
        private Func<double, double, double> indicator;

        private double square;

        //Принимает генератор БСВ, площадь фигуры, индикаторную функцию, подынтегральную функцию и пределы для начертания квадрата на плоскости, в который нужная вписана фигура.
```

```
        public MCTwoDimensionalIntegrator(
            Random random, double square,
            Func<double, double, double> indicator,
            Func<double, double, double> integrand,
            double upperXBound, double upperYBound,
            double lowerXBound, double lowerYBound)
        {
            this.square = square;

            func = integrand;
            this.indicator = indicator;
            this.random = random;

            this.upperXBound = upperXBound;
            this.upperYBound = upperYBound;

            this.lowerXBound = lowerXBound;
            this.lowerYBound = lowerYBound;
        }

        //Принимает количество точек для вычисления, при вычислении домножает результат
подынтегральной функции на индикаторную функцию.

        public double Integrate(int pointsCount)
        {
            double deltaX = upperXBound - lowerXBound;
            double deltaY = upperYBound - lowerYBound;

            double sum = 0;

            for (int i = 0; i < pointsCount * 2; i++)
            {
                double x = random.NextDouble() * deltaX + lowerXBound;
                double y = random.NextDouble() * deltaY + lowerYBound;

                sum += func(x, y) * indicator(x, y);
            }

            return square * sum / pointsCount;
        }
    }
...
//Пример вычисления интеграла согласно варианту работы.


int pointsCount = 1000;


Func<double, double, double> secondIntegrand =
                new Func<double, double, double>((double x, double y) =>
                {
                    double result = Math.Pow(x, 3) + 3 * x * y;
                    return result * Math.Pow(Math.E, y);
                });


Func<double, double, double> indicator =
                new Func<double, double, double>((double x, double y) =>
                {
                    return Math.Abs(x) + Math.Abs(y) < 1 ? 1.0 : 0.0;
                });


IIntegrator integral = new MCTwoDimensionalIntegrator(
```

```
                new Random(), 2.0, indicator, secondIntegrand, 1, 1, -1, -1);

double result = integral.Integrate(pointsCount);
```

Результат работы программы:





Данный результат получен аналогично результату для первого интеграла и демонстрирует нам, что выводы, полученные при построении выборки из 10000 вычислений одномерного интеграла методом Монте-Карло, также применимы для случая двойного интеграла, что подтверждает приведённые выше утверждения о распределении полученных значений и справедливость использования самого метода Монте-Карло.

Лабораторная работа 5.

Условие:

Решить систему линейных уравнений, используя метод Монте-Карло.

1. Решить систему линейных алгебраических уравнений методом Монте-Карло.

2. Сравнить с решением данного уравнения, полученным в произвольном математическом пакете.

3. Построить график зависимости точности решения от длины цепи маркова и числа смоделированных цепей маркова.

$$7)\ A=\begin{pmatrix}1.2 & -0.4 & 0.3\\ 0.1 & 0.7 & -0.2\\ -0.4 & 0.0 & 1.4\end{pmatrix}, f=\begin{pmatrix}1\\ 2\\ -2\end{pmatrix};$$

Теория:

Метод Монте-Карло приближенного решения системы линейных алгебраических уравнений:

Необходимо решить систему, представленную в виде $x = Ax + f$, где $x = (x_1,...,x_n)^T$, $f = (f_1,..., f_n)^T$, $A = (a_{ij}),\ i, j = \overline{1,n}$, собственные значения A по модулю меньше 1.

Наша цель – вычислить скалярное произведение вектора решения $x = (x_1,..., x_n)^T$ с некоторым вектором $h = (h_1,..., h_n)^T$.

Рассмотрим цепь Маркова с параметрами $\pi = (\pi_1,...,\pi_n)^T$, $P = (p_{ij})$, такими что

$$\pi_i \geq 0,\ \sum_{i=1}^{n} \pi_i = 1;$$

$$p_{ij} \geq 0,\ \sum_{j=1}^{n} p_{ij} = 1;$$

$$\pi_i > 0,\ \text{если}\ h_i \neq 0;$$

$$p_{ij} > 0,\ \text{если}\ a_{ij} \neq 0.$$

Положим

$$g_i^{(0)} = \begin{cases} h_i / \pi_i,\ \pi_i > 0\\ 0,\ \pi_i = 0\end{cases},\ g_i^{(k)} = \begin{cases} a_{ij} / p_{ij},\ p_{ij} > 0\\ 0,\ p_{ij} = 0\end{cases}.$$

Выберем некоторое натуральное N и рассмотрим случайную величину

$$\xi_N = \sum_{m=0}^{N} Q_m f_{i_m},$$

Где $i_0 \rightarrow i_1 \rightarrow \ldots \rightarrow i_m$ - траектория цепи Маркова.

Q_m определяется как:

$$Q_0 = g_{i_0}^{(0)},\ Q_m = Q_{m-1} g_{i_{m-1} i_m}^{(m)}.$$

Тогда скалярное произведение вектором h и x приблизительно равно $E\{\xi_N\}$.

Можем найти x, скалярно умножая его на векторы h у которых в одной позиции стоит 1, а в остальных – 0.

Код программы:

```
class Program
    {
        public static double[,] A = new double[3, 3]
        {
            { 1.2, -0.4, 0.3 },
            { 0.1, 0.7, -0.2 },
            { -0.4, 0.0, 1.4 }
        };

        public static double[] f = new double[3]
        {
            1.0,
            2.0,
            -2.0
        };

        public static double[] exactSolution = new double[]
        {
            1.83800623052959,
            2.33644859813084,
            -0.903426791277259
        };

        public static void Main(String[] args)
        {
            MonteCarloSolver solver = new MonteCarloSolver(3, A, f);
            solver.Solve(1, 1, 10, exactSolution);

            Console.ReadKey();
            Console.ReadKey();
        }
      }


public class MonteCarloSolver
    {
        private int n;

        private Random random;

        private double[,] A;
        private double[,] h;
        private double[,] P;

        private double[] Af;
        private double[] f;
        private double[] X;

        public MonteCarloSolver(int size, double[,] A, double[] f)
        {
            n = size;
            random = new Random();
```

```
            this.A = new double[size, size];
            this.h = new double[size, size];
            this.P = new double[size, size];

            this.Af = new double[size * (size + 1)];
            this.f = f.Clone() as double[];
            this.X = new double[size];

            int k = 0;

            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                    Af[k++] = A[i, j];
            }

            for (int i = 0; i < size; i++)
                Af[k++] = f[i];

            InitMatrices(size, this.A, this.f);
        }

        private void InitMatrices(int size, double[,] A, double[] f)
        {
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    A[i, j] = Af[j + i * n];
                    h[i, j] = 0.0;
                    P[i, j] = 1.0 / n;
                }

                f[i] = Af[n * n + i];

                //Приводим систему к нужному виду
                for (int j = 0; j < n; j++)
                {
                    if (i == j)
                    {
                        A[i, j] = 1 - A[i, j];
                        h[i, j] = 1;
                    }
                    else
                        A[i, j] *= -1.0;
                }
            }
        }

        public void Solve(int chainLength, int chainCount, int steps, double[]
exactSolution)
        {
            Console.Write("\n\nТочное решение:");

            for (int i = 0; i < n; i++)
                Console.Write("  {0:f5}", exactSolution[i]);

            Console.WriteLine("\n\n         Результат                 Отклонение
Цепи Маркова\n" +
                        "    X1       X2       X3        (среднее)          Число
Длина\n\n");

            for (int j = 0; j < steps; j++)
            {
                for (int i = 0; i < n; i++)
```

```
                X[i] = CalcX(A, i, f, P, chainLength, chainCount);

            for (int i = 0; i < n - 1; i++)
                Console.Write("{0:f5}  ", X[i]);

            Console.Write("{0:f5}", X[n - 1]);

            double diff = 0;
            for (int i = 0; i < n; i++)
                diff += exactSolution[i] - X[i];

            Console.Write("       {0:f5}", Math.Abs(diff / n));
            Console.Write("      {0,9}    {1,9}\n", chainCount, chainLength);

            chainLength *= 2;
            chainCount *= 4;

            Console.WriteLine();
        }
    }

    private double CalcX(double[,] matrix, int idx, double[] f,
        double[,] P, int chainLength, int chainCount)
    {
        double Qprev, Qcurr;
        int Mprev, Mcurr;
        double X = 0.0;

        //вычисляем веса для состояний цепи Маркова
        //chainCount - номер реализации цепи Маркова
        for (int i = 0; i < chainCount; i++)
        {
            Mprev = idx;
            Qprev = 1.0;

            //g = a/p, p > 0
            //g = 0, p = 0
            for (int j = 1; j < chainLength; j++)
            {
                Mcurr = MarkovChain(P, Mprev);

                if (P[Mprev, Mcurr] > 0)
                    Qcurr = Qprev * matrix[Mprev, Mcurr] / P[Mprev, Mcurr];
                else
                    Qcurr = 0.0;

                X += Qcurr * f[Mcurr]; //случайная величина
                Qprev = Qcurr;
                Mprev = Mcurr;
            }
        }

        return (f[idx] + X / chainCount);
    }

    //моделируем вспомогательную цепь Маркова
    private int MarkovChain(double[,] P, int curr)
    {
        double rd = random.NextDouble();

        for (int i = 0; i < n; i++)
        {
            rd -= P[curr, i];

            if (rd < 0)
                return i;
```

```
            }

            return (n - 1);
        }
    }
```

Результат:

Точное решение данной системы уравнений, полученное в Jupyter Notebook при помощи библиотеки символьных вычислений SymPy для языка Python.

```
In [3]:  import sympy as sp

In [4]:  A = sp.Matrix([[1.2, -0.4, 0.3], [0.1, 0.7, -0.2], [-0.4, 0.0, 1.4]])
         b = sp.Matrix([1.0, 2.0, -2.0])

In [5]:  A
```

Out[5]:
$$\begin{bmatrix} 1.2 & -0.4 & 0.3 \\ 0.1 & 0.7 & -0.2 \\ -0.4 & 0.0 & 1.4 \end{bmatrix}$$

```
In [6]:  b
```

Out[6]:
$$\begin{bmatrix} 1.0 \\ 2.0 \\ -2.0 \end{bmatrix}$$

```
In [7]:  result = A.solve(b)
         result
```

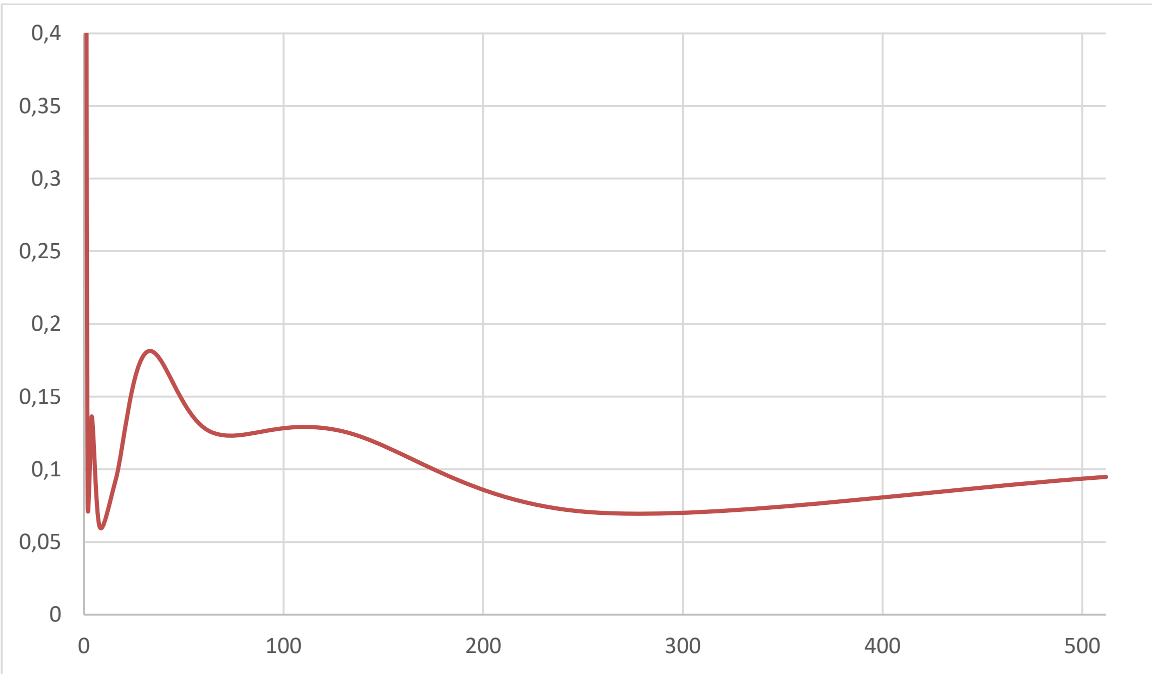
Out[7]:
$$\begin{bmatrix} 1.83800623052959 \\ 2.33644859813084 \\ -0.903426791277259 \end{bmatrix}$$

Результат работы программы для различных параметров цепей Маркова.

Точное решение: 1,83801 2,33645 -0,90343

Результат			Отклонение	Цепи Маркова	
X1	X2	X3	(среднее)	Число	Длина
1,00000	2,00000	-2,00000	0,75701	100	1
1,99600	2,23400	-0,72800	0,07699	100	2
2,12368	2,18672	-0,63008	0,13643	100	4
2,20219	2,38568	-1,13562	0,06041	100	8
1,73120	2,44943	-1,18751	0,09264	100	16
1,92746	2,27775	-0,39108	0,18103	100	32
1,57055	2,20812	-0,88438	0,12558	100	64
1,34028	2,29763	-0,74691	0,12667	100	128
1,85350	2,07703	-0,87040	0,07030	100	256
1,79893	2,21612	-1,02839	0,09479	100	512

На основании полученных результатов построим график зависимости среднего отклонения от длины цепи Маркова в методе Монте Карло:

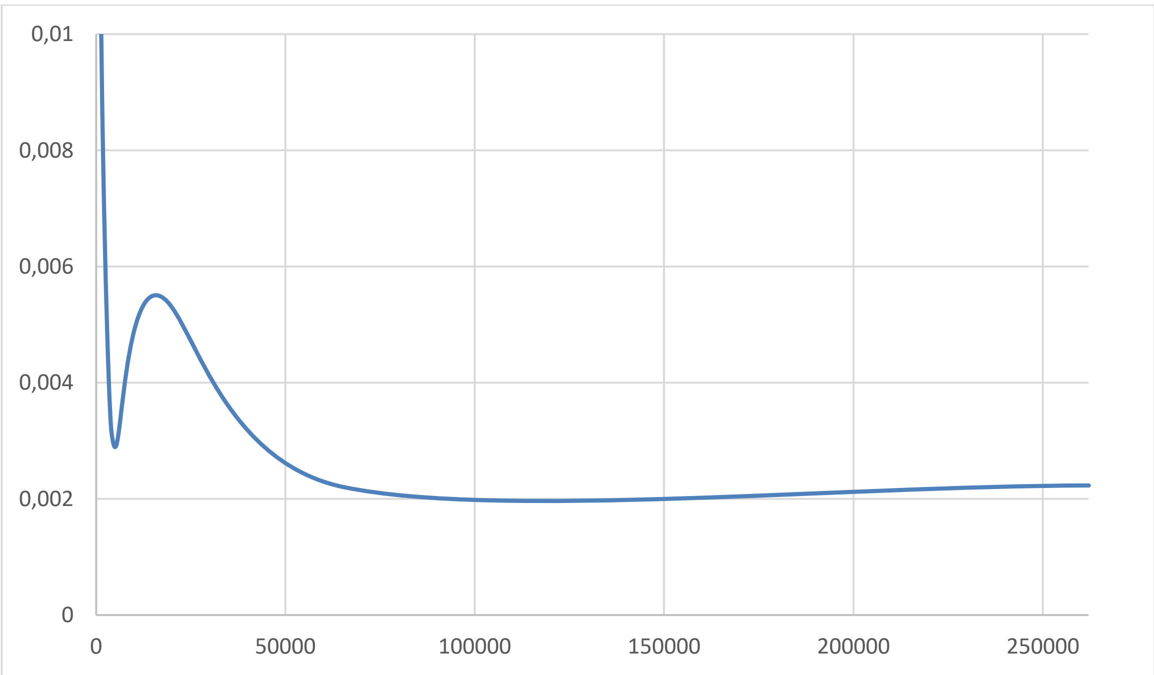


Можем сделать вывод о том, что увеличение длины цепей положительно сказывается на точности решения, однако т.к. алгоритм построения решения случайный, такое моделирование не может показать нам явную зависимоть на матрицах малых размеров.

Точное решение: 1,83801 2,33645 -0,90343

Результат X1	Результат X2	Результат X3	Отклонение (среднее)	Цепи Маркова Число	Цепи Маркова Длина
0,10820	4,01600	-3,44185	0,86289	1	1000
-0,06060	2,69386	-1,68320	0,77366	4	1000
1,41201	1,84793	-0,43525	0,14878	16	1000
1,73175	2,54188	-1,21861	0,07201	64	1000
1,75594	2,25092	-1,02452	0,09623	256	1000
1,83197	2,33076	-0,93103	0,01311	1024	1000
1,90017	2,31653	-0,93626	0,00314	4096	1000
1,84219	2,34571	-0,90037	0,00550	16384	1000
1,83421	2,33738	-0,89395	0,00220	65536	1000
1,84036	2,33029	-0,89294	0,00223	262144	1000

Постоим также график зависимости среднего отклонения от числа цепей для моделирования решения:



В данном случае можем сделать аналогичный вывод о том, что увеличение числа цепей для моделирования положительно сказывается на точности, но т.к. алгоритм основан на случайных числах, нельзя проследить явную зависимость на малых данных.

В целом, исходя из полученных данных, имеет смысл моделирование решений для систем уравнений большой размерности, а также наращивание количества и длины цепей в пределах 100000 цепей длиной в 256 элементов, т.к. наращивание данных параметров до бесконечности даёт также и сильный прирост в трудоёмкости таких вычислений.