МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«**УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**»
Кафедра «Вычислительная техника»
Дисциплина «Моделирование»

Лабораторная работа № 1
По теме «Анализ и генерация случайных чисел. Основы имитационного моделирования»

Вариант 1

Выполнил: студент группы ИВТВМбд-31
Албутов Д. В.
Проверила: доцент, к.т.н.
Валюх В.В.

Ульяновск 2020

Цель работы:

Изучение основных характеристик случайных величин на базе теории вероятностей и математической статистики; изучение и программирование способов получения псевдослучайных чисел.

Выполнение работы:

Для выполнения лабораторной работы был использован язык C# и класс Random. Данный класс использует алгоритм с вычитанием двух предыдущих чисел последовательности, описанный Дональдом Кнутом во втором томе его книги «Искусство программирования». Числа будут генерироваться от 0 до 45340. Последнее число получено извлечением корня из максимального числа, которое вмещает int, округлённым по нижней границе.

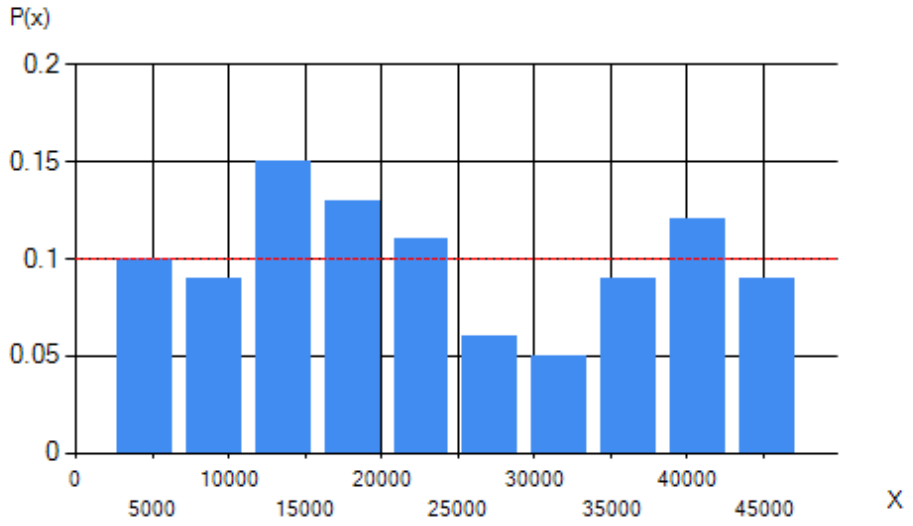
Сгенерируем с помощью стандартного генератора последовательности из 100, 1000 и 10000 чисел и посчитаем для них математическое ожидание, дисперсию и среднеквадратическое отклонение:

Количество элементов	Математическое ожидание	Дисперсия ($*10^8$)	Среднеквадратическое отклонение
100	21823.19	1.791544	13384.8559200314
1000	23380.79	1.887713	13739.4055184349
10000	23013.55	1.830876	13530.9884339615

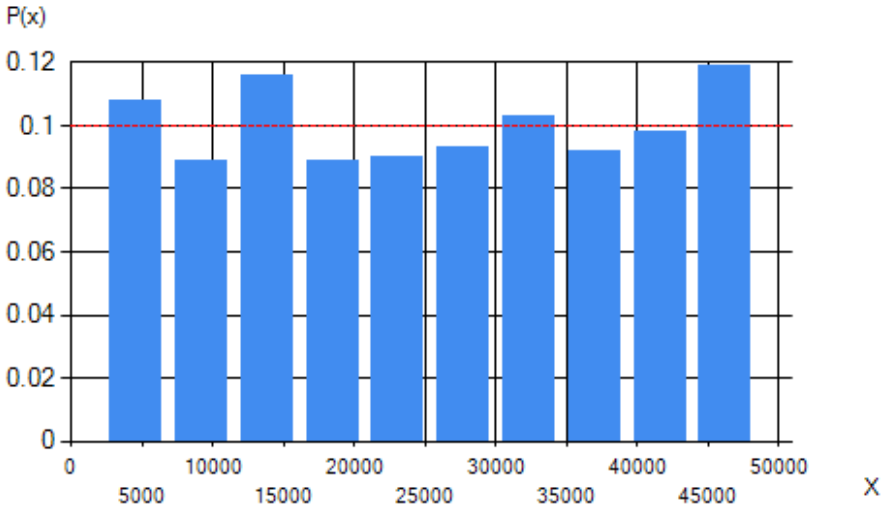
Далее проверим частотность генератора. Для это разделим наборы чисел на десять интервалов и оценим вероятность попадания чисел в данные интервала. Теоретическая вероятность попадания в интервал для идеального генератора равна единице, делённой на число интервалов, то есть 0.1 для данного случая.

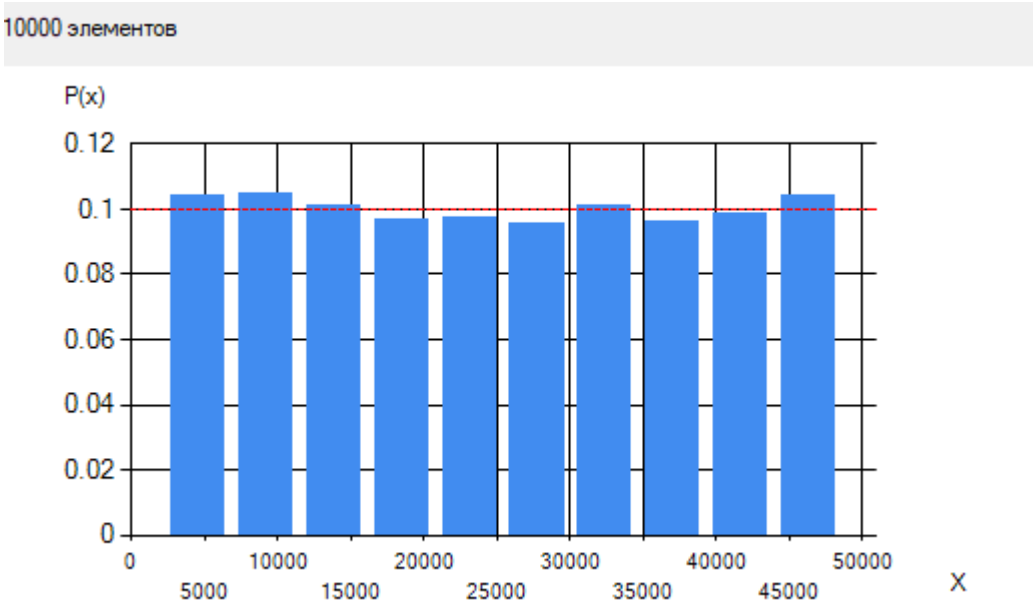
По этим данным составим графики:

100 элементов



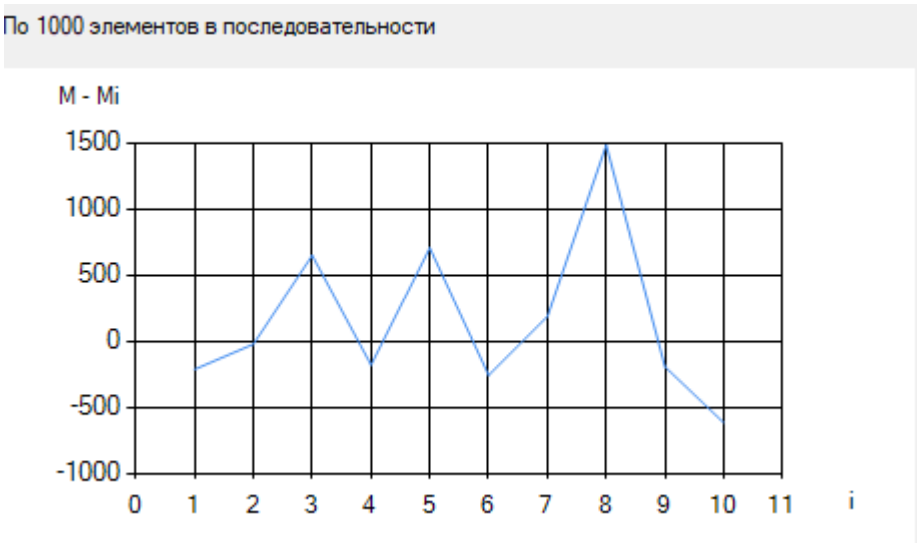
1000 элементов





Далее сгенерируем два набора чисел, каждый из которых включает десять последовательностей из чисел. Количество элементов в первом наборе статично и равно тысяче, для второго набора количество элементов определяется по формуле I * 1000, где I - это номер последовательности.

Для каждой последовательности чисел посчитаем её математическое ожидание и найдём разницу с теоретическим значениям, которое равно сумме минимального и максимального значений последовательности, делённой на 2, то есть 22670 для данных последовательностей. На основании этих разностей построим график зависимости разницы математических ожиданий от номера последовательности:

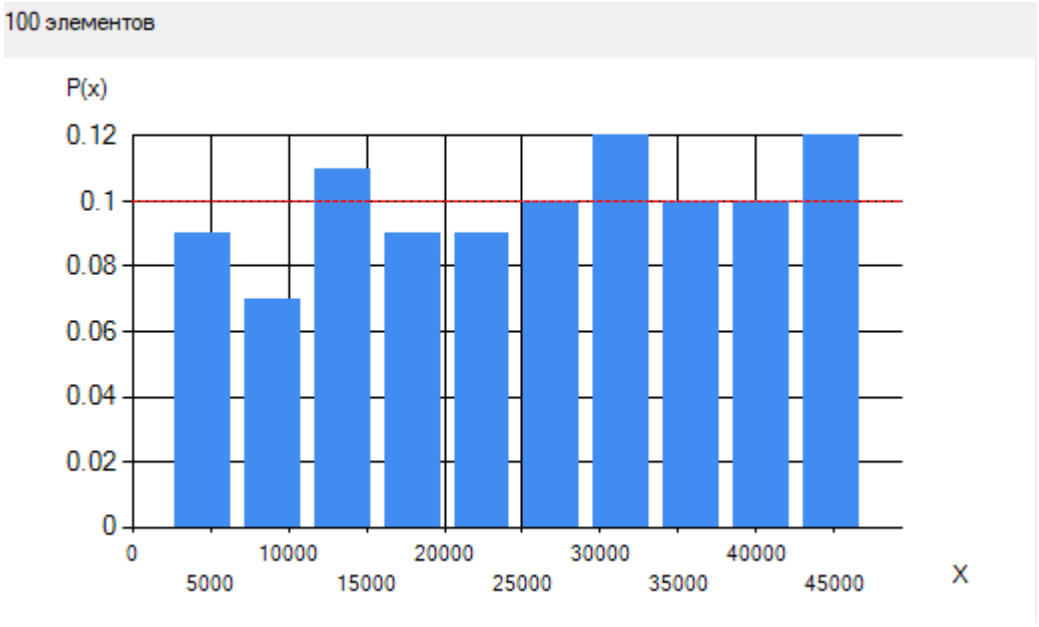


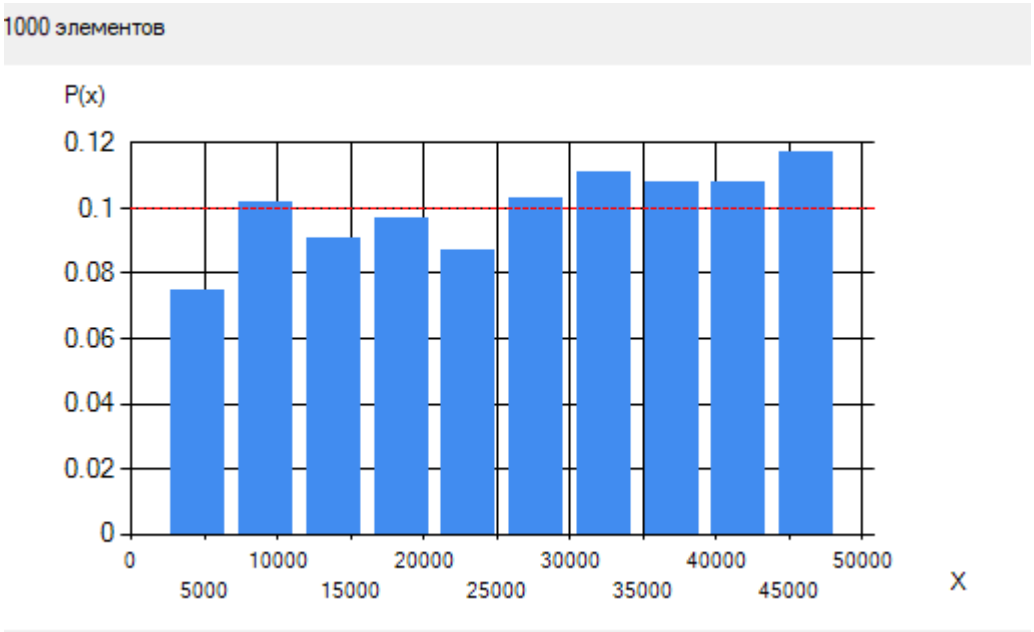
Теперь напишем собственный генератор псевдослучайных чисел. За основу возьмём линейный конгруэнтный метод. Данный генератор является одним из простейших и описывается формулой $X_{n+1} = (a*X_n + b) \bmod m$, где a, b и m - статические параметры, выбранные заранее. Выберем параметры, используемые в Microsoft Visual/Quick C/C++, с a = 214013, b = 2531011, $m = 2^{32}$. А зерном выберем количество времени в миллисекундах, прошедшее с запуска системы.

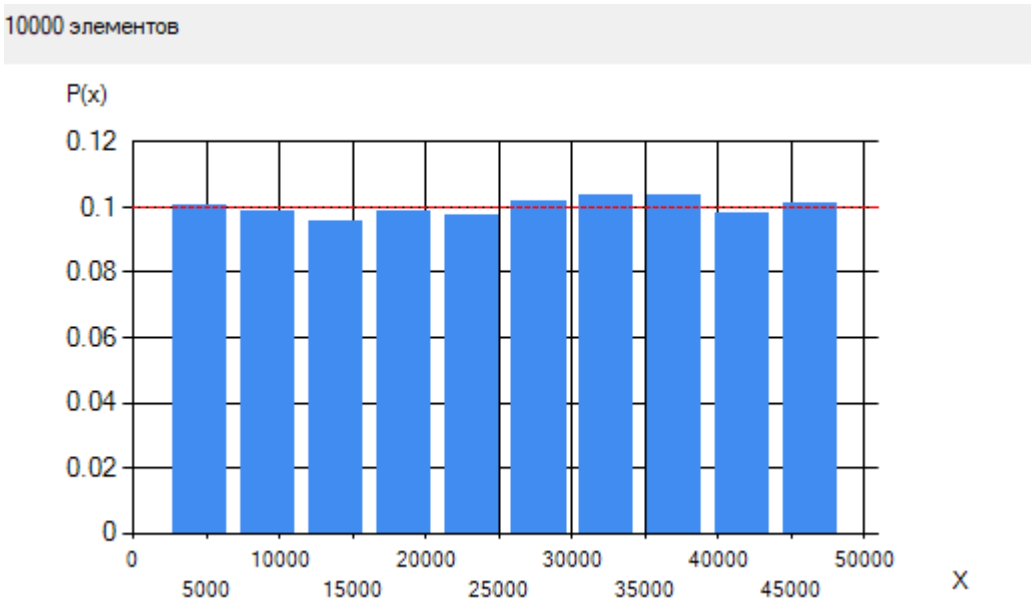
Теперь посчитаем математическое ожидание, дисперсию и среднеквадратическое отклонение для данного генератора:

Количество элементов	Математическое ожидание	Дисперсия ($*10^8$)	Среднеквадратическое отклонение
100	23741.74	1.692498	13009.6038371658
1000	24420.79	1.75757	13257.3372892146
10000	23318.35	1.779649	13340.3496206059

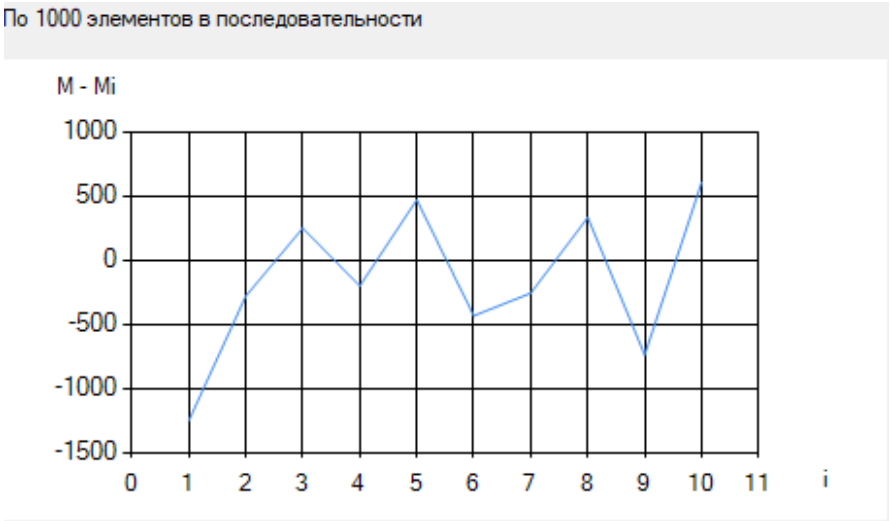
Далее составим графики частотности генератора:







Также проверим генератор на равномерность:



Вывод:

В ходе выполнения лабораторной работы мы познакомились с несколькими алгоритмами генерации псевдослучайных чисел. Провели тесты частот и равномерности генераторов данных чисел. Изучили основные характеристики случайных величин на базе теории вероятностей и математической статистики.

Исходный код:

```
    static class Program
    {
        private const int NUM_OF_INTERVALS = 10;
        private const int MAX_VALUE = 46340; //floor(sqrt(int.maxValue))

        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }

        public enum GeneratorType
        {
            DEFAULT,
            CUSTOM
        }

        public static List<int> GenerateRandomArray(int size, GeneratorType type,
Random randomizer = null)
        {
            var generatedItems = new List<int>();
            Random random;

            if (randomizer != null)
                random = randomizer;
            else if (type == GeneratorType.DEFAULT)
                random = new Random();
```

```
            else
                random = new Randomizer();

            for (int i = 0; i < size; i++)
                generatedItems.Add(random.Next(MAX_VALUE));

            return generatedItems;
        }

        public static float CalculateExpectedValue(List<int> array)
        {
            var sum = 0.0f;
            var probability = 1.0f / array.Count;

            foreach (var item in array)
                sum += item * probability;

            return sum;
        }

        public static float CalculateDispersion(List<int> array, float
expectedVal)
        {
            var squareArray = new List<int>(array.Count);

            foreach (var item in array)
                squareArray.Add((int)Math.Pow(item, 2));

            return CalculateExpectedValue(squareArray) -
(float)Math.Pow(expectedVal, 2);
        }

        public static Dictionary<int, float> CalculateFrequencies(List<int>
array)
        {
            var intervalWidth =  (array.Max() - array.Min()) / NUM_OF_INTERVALS;
            var frequencies = new Dictionary<int, float>(NUM_OF_INTERVALS);
            var upperBound = intervalWidth;
            var count = 0;

            array.Sort();

            for (int i = 0; i < array.Count; i++)
            {
```

```
                if (array[i] >= upperBound)
                {
                    frequencies.Add(upperBound, (float) count / array.Count);
                    upperBound += intervalWidth;
                    count = 0;
                }

                count++;
            }

            return frequencies;
        }

        public static Dictionary<int, float>[]
ExecuteUnifirmityTest(GeneratorType type)
        {
            var points = new Dictionary<int, float>[]
            {
                new Dictionary<int, float>(),
                new Dictionary<int, float>()
            };
            var expectedVal = (float) MAX_VALUE / 2;
            Random random;

            if (type == GeneratorType.DEFAULT)
                random = new Random();
            else
                random = new Randomizer();

            for (int i = 1; i <= 10; i++)
            {
                var generatedArray = GenerateRandomArray(1000, type, random);
                var realVal = CalculateExpectedValue(generatedArray);
                points[0].Add(i, expectedVal - realVal);
            }

            for (int i = 1; i <= 10; i++)
            {
                var generatedArray = GenerateRandomArray(i * 1000, type);
                var realVal = CalculateExpectedValue(generatedArray);
                points[1].Add(i, expectedVal - realVal);
            }

            return points;
```

```
        }

        public static int GetNumberOfIntervals() => NUM_OF_INTERVALS;
    }
```

Класс Form1:

```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            DrawHorizontalLines();
        }

        private HorizontalLineAnnotation GetLineForChart(Chart chart)
        {
            return new HorizontalLineAnnotation
            {
                IsInfinitive = true,
                LineColor = Color.Red,
                LineDashStyle = ChartDashStyle.Dash,
                Y = 1.0 / Program.GetNumberOfIntervals(),

                AxisX = chart.ChartAreas[0].AxisX,
                AxisY = chart.ChartAreas[0].AxisY,
                ClipToChartArea = chart.ChartAreas[0].Name
            };
        }

        private void DrawHorizontalLines()
        {
            chart1.Annotations.Add(GetLineForChart(chart1));
            chart2.Annotations.Add(GetLineForChart(chart2));
            chart3.Annotations.Add(GetLineForChart(chart3));
        }

        private void Button2_Click(object sender, EventArgs e)
        {
            var type = Program.GeneratorType.CUSTOM;
            var hundreds = Program.GenerateRandomArray(100, type);
            var thousands = Program.GenerateRandomArray(1000, type);
            var tensOfThousands = Program.GenerateRandomArray(10000, type);

            CompleteLab1(hundreds, thousands, tensOfThousands);
```

```
    var expValDifferences = Program.ExecuteUnifirmityTest(type);

    foreach (KeyValuePair<int, float> point in expValDifferences[0])
        chart4.Series["Series1"].Points.AddXY(point.Key, point.Value);

    foreach (KeyValuePair<int, float> point in expValDifferences[1])
        chart5.Series["Series1"].Points.AddXY(point.Key, point.Value);
}

private void Button3_Click(object sender, EventArgs e)
{
    var type = Program.GeneratorType.DEFAULT;
    var hundreds = Program.GenerateRandomArray(100, type);
    var thousands = Program.GenerateRandomArray(1000, type);
    var tensOfThousands = Program.GenerateRandomArray(10000, type);

    CompleteLab1(hundreds, thousands, tensOfThousands);

    var expValDifferences = Program.ExecuteUnifirmityTest(type);

    foreach (KeyValuePair<int, float> point in expValDifferences[0])
        chart4.Series["Series1"].Points.AddXY(point.Key, point.Value);

    foreach (KeyValuePair<int, float> point in expValDifferences[1])
        chart5.Series["Series1"].Points.AddXY(point.Key, point.Value);
}

private void ClearCharts()
{
    foreach (var series in chart1.Series)
        series.Points.Clear();

    foreach (var series in chart2.Series)
        series.Points.Clear();

    foreach (var series in chart3.Series)
        series.Points.Clear();

    foreach (var series in chart4.Series)
        series.Points.Clear();

    foreach (var series in chart5.Series)
        series.Points.Clear();
```

```
        }

        private void CompleteLab1(
            List<int> hundreds,
            List<int> thousands,
            List<int> tensOfThousands)
        {
            var expVal1 = Program.CalculateExpectedValue(hundreds);
            var expVal2 = Program.CalculateExpectedValue(thousands);
            var expVal3 = Program.CalculateExpectedValue(tensOfThousands);

            expValues.Text = expVal1 + "\r\n" + expVal2 + "\r\n" + expVal3;

            var dispersion1 = Program.CalculateDispersion(hundreds, expVal1);
            var dispersion2 = Program.CalculateDispersion(thousands, expVal2);
            var dispersion3 = Program.CalculateDispersion(tensOfThousands,
expVal3);

            dispersions.Text = dispersion1 +
                "\r\n" + dispersion2 + "\r\n" + dispersion3;
            standDeviations.Text = Math.Sqrt(dispersion1) +
                "\r\n" + Math.Sqrt(dispersion2) + "\r\n" +
Math.Sqrt(dispersion3);

            var frequency1 = Program.CalculateFrequencies(hundreds);
            var frequency2 = Program.CalculateFrequencies(thousands);
            var frequency3 = Program.CalculateFrequencies(tensOfThousands);

            ClearCharts();

            foreach (KeyValuePair<int, float> sector in frequency1)
                chart1.Series["Series1"].Points.AddXY(sector.Key,
sector.Value);

            foreach (KeyValuePair<int, float> sector in frequency2)
                chart2.Series["Series1"].Points.AddXY(sector.Key,
sector.Value);

            foreach (KeyValuePair<int, float> sector in frequency3)
                chart3.Series["Series1"].Points.AddXY(sector.Key,
sector.Value);
        }
    }
```

Класс Randomizer:

```
class Randomizer : Random
{
    private const int A = 214013;
    private const int B = 2531011;
    private const int M = int.MaxValue;

    private long seed;

    public Randomizer() : this(Environment.TickCount) {}

    public Randomizer(int seed) => this.seed = seed;

    public override int Next()
    {
        seed = (A * seed  + B) % M;
        return (int) seed;
    }

    public override int Next(int maxVal) => Next() % maxVal;
}
```