**МИНОБРНАУКИ РОССИИ**

**ФГБОУ ВО «ТУЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт прикладной математики и компьютерных наук

Кафедра вычислительной техники

Отчет по лабораторной работе № 2

**«МАШИННАЯ ИМИТАЦИЯ СЛУЧАЙНЫХ**

**ПОСЛЕДОВАТЕЛЬНОСТЕЙ ЧИСЕЛ»**

по дисциплине «Основы моделирования систем»

Выполнил студент группы 220681:

Шайхаттаров Д.В.

Проверил:

доц. Семенчев Е.А.

Тула 2020

# ЦЕЛЬ РАБОТЫ

Изучение функционирования программных датчиков псевдослучайных чисел. Практическая проверка качества генераторов случайных чисел.

# ЗАДАНИЕ

1. Нарисовать схему алгоритма вычисления распределения случайных чисел по К интервалам.
2. Нарисовать полную схему алгоритма проверки качества датчиков случайных чисел.
3. Написать полную программу проверки качества датчиков случайных чисел.

# ХОД РАБОТЫ

В работе используются два датчика случайных чисел, равномерно распределённых на интервале [(0,1]. Генерируется N случайных чисел для последующего анализа каждого датчика. При этом весь интервал [0,1] разбивается на k меньших интервалов одинаковой длины.

Для выполнения данной работы используется два генератора псевдослучайных чисел, а именно: линейный и мультипликативный.

Оба метода образования последовательности псевдослучайных чисел можно выразить следующими формулами.

Последовательность, получаемая с помощью линейного метода:

Последовательность, получаемая с помощью мультипликативного метода:

## Распределение чисел по интервалам

Алгоритм распределения чисел по интервалам и работы ГСПЧ выглядит следующим образом (Рисунок 1)

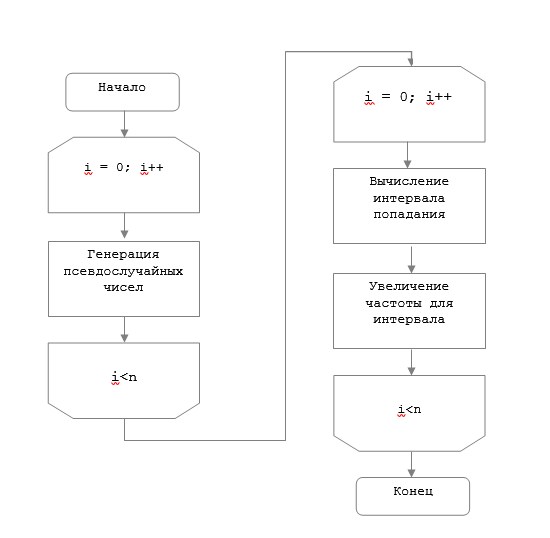


Рисунок 1 – Алгоритм распределения чисел

## Проверка качества генераторов

Качество работы ГПСЧ оценивается с помощью критерия хи-квадрат:

где вычисленная частота, ожидаемая частота для конкретного интервала.

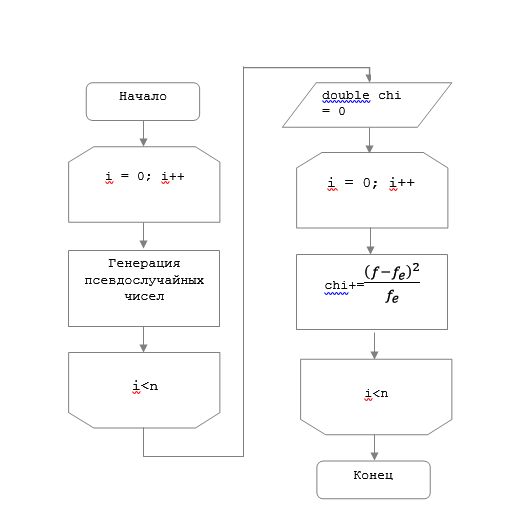


Рисунок 2 – Алгоритм проверки качества псевдослучайной последовательности

# Тестирование

В результате тестирования были получены следующие результаты (Рисунок 3)

Реализацию программы см. в «Приложение».



Рисунок 3 – Результат выполнения программы

В результате были получены следующий гистограммы. По данным гистограммам можно судить о том, является ли реализованные ГПСЧ равномерными.

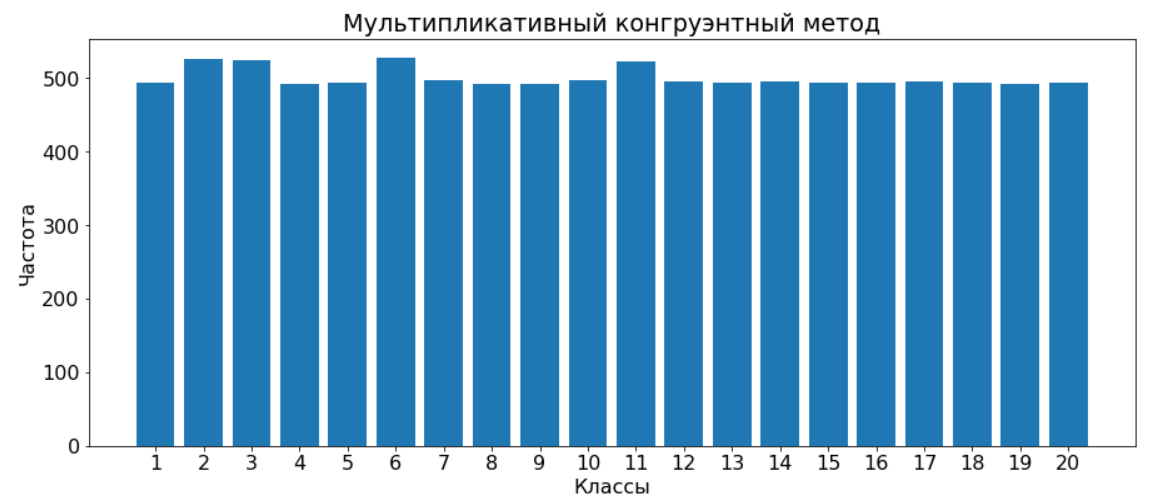


Рисунок 4 – Мультипликативный метод

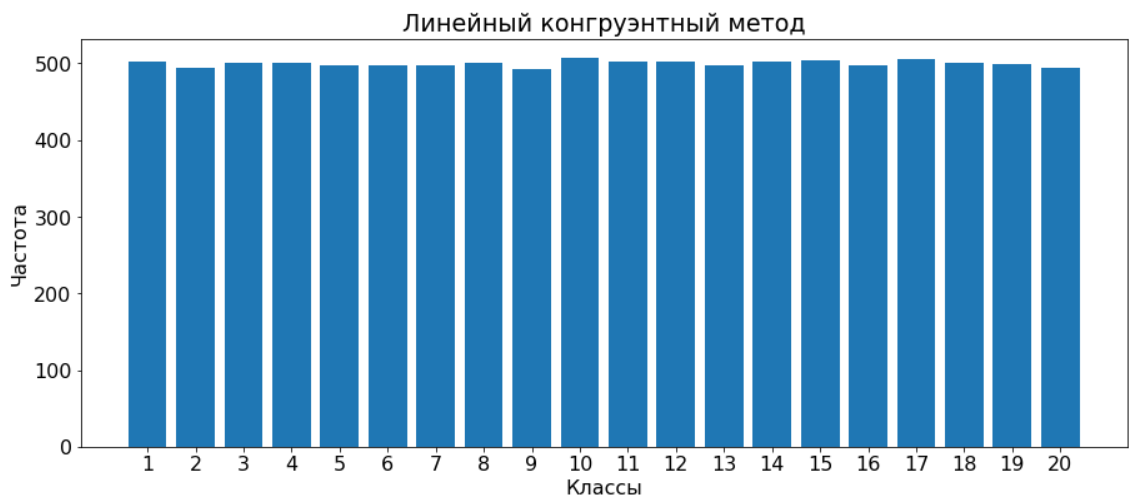


Рисунок 5 – Линейный метод

Определим число степеней свободы каждого из ГПСЧ. Для мультипликативного метода: 20 – 1 – 2 = 17, а эталонное значение хи-квадрат должно быть равно 27.6. Для линейного: 20 – 1 – 3 = 16, а эталонное значение хи-квадрат – 26.3.

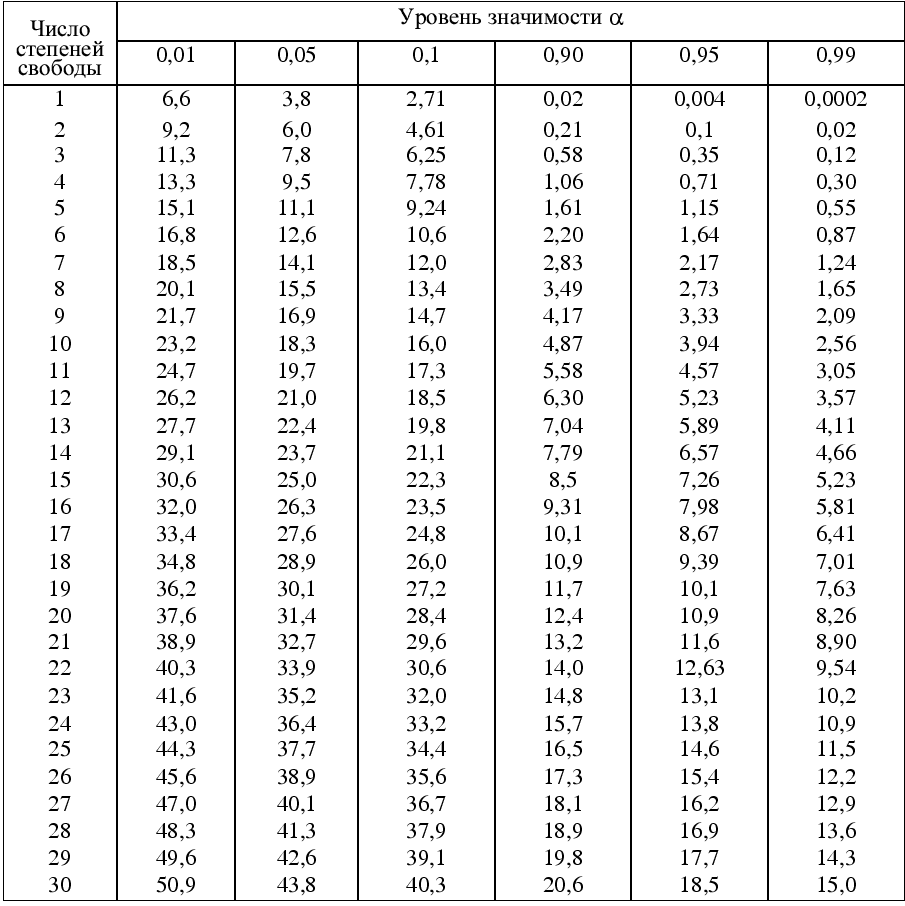


Рисунок 6 – Распределение хи-квадрат

# ВЫВОД

По отношению к табличным значениям оба реализованных ГПСЧ дают равномерное распределене значений, но лучшим из них является линейный конгруэнтный метод, т.к. он имеет критерий хи-квадрат меньше, чем мультипликативный.

# Приложение

## IRandom.cs

namespace NumberGenerator

{

public interface IRandom

{

double GenerateNumber();

}

}

## LinearCongruentMethod.cs

namespace NumberGenerator

{

public class LinearCongruentMethod : IRandom

{

private int a;

private int c;

private int m;

private double xn;

public LinearCongruentMethod(int x0=5, int a=211, int c=1663, int m=7875)

{

xn = x0;

this.a = a;

this.c = c;

this.m = m;

}

public double GenerateNumber()

{

xn = (a \* xn + c) % m;

return xn / m;

}

}

}

## MultiplicativeCongruentMethod.cs

namespace NumberGenerator

{

public class MultiplicativeCongruentMethod : IRandom

{

private int a;

private int m;

private double xn;

public MultiplicativeCongruentMethod(int x0=5, int t=43, int m=6075)

{

xn = x0;

a = 8 \* t + 3;

this.m = m;

}

public double GenerateNumber()

{

xn = (a \* xn) % m;

return xn / m;

}

}

}

## Testing.cs

using System;

namespace NumberGenerator

{

public class Testing

{

private IRandom random;

private int[] frequencies;

private int numberIntervals;

private int amountNumbers;

private int expectedFrequency;

public Testing(IRandom random, int numberIntervals=20, int amountNumbers=10000)

{

this.random = random;

this.numberIntervals = numberIntervals;

this.amountNumbers = amountNumbers;

frequencies = new int[numberIntervals];

expectedFrequency = amountNumbers / numberIntervals;

}

public double ChiSquared()

{

FilArrayOfFrequencies();

double chi = 0;

for (int i = 0; i < numberIntervals; i++)

chi += Math.Pow(frequencies[i] - expectedFrequency, 2) / expectedFrequency;

return chi;

}

private void FilArrayOfFrequencies()

{

double[] generatedNumbers = GenerateArrayOfNumbers();

for (int i = 0; i < amountNumbers; i++)

{

int hitInterval = CalculateHitInterval(generatedNumbers[i]);

frequencies[hitInterval]++;

}

}

private double[] GenerateArrayOfNumbers()

{

var array = new double[amountNumbers];

for (int i = 0; i < amountNumbers; i++)

array[i] = random.GenerateNumber();

return array;

}

private int CalculateHitInterval(double num)

=> (int)(numberIntervals \* num);

}

}

## Program.cs

using System;

namespace NumberGenerator

{

class Program

{

static void Main(string[] args)

{

var random1 = new MultiplicativeCongruentMethod();

var random2 = new LinearCongruentMethod();

var testing1 = new Testing(random1);

var testing2 = new Testing(random2);

Console.WriteLine($"Хи-квадрат для мультипликативного конгруэнтного метода = {testing1.ChiSquared()}");

Console.WriteLine($"Хи-квадрат для линейного конгруэнтного метода = {testing2.ChiSquared()}");

}

}

}