Задание №3, 17 вариант

**«Проверка встроенного ГСЧ и создание на его базе собственного»**

1. **Проверка встроенного ГСЧ**

При помощи библиотечного ГСЧ Random на языке Java получила 100000 равномерно распределённых чисел на промежутке [0;1].

Для этих чисел подсчитала мат.ожидание и СКО. Сравнила их с теоретическими:

|  |  |  |
| --- | --- | --- |
|  | Теоретические: | Подсчитанные мной: |
|  | 0.5 | 0.5001269969962386 |
|  | 0.083 | 0.0831489604378391 |
|  | 0.288 | 0.288355614541904 |

Вывод: Подсчитанные мат.ожидание, дисперсия и СКО полностью соответствуют теоретическим данным. Таким образом генератор случайных чисел, который используется в стандартной библиотеке Java, генерирует равномерно распределённые числа.

Частотная диаграмма:

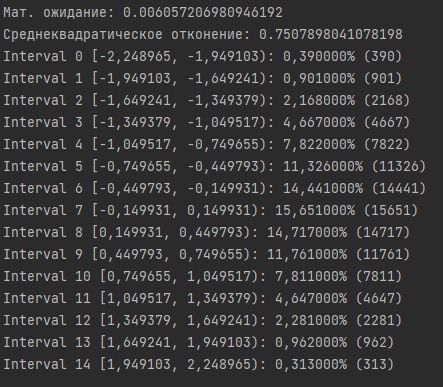
1. **Создание собственного ГСЧ**

На базе библиотечного ГСЧ создала свой, генерирующий случайные числа по нормальному закону с параметрами (m=0, σ=y). Для этого воспользовалась двумя методами:  
**1) Метод генерации по ЦПТ:**

Описание общего алгоритма работы для метода генерации случайных чисел по ЦПТ:

1. Генерируем 12 случайных чисел от 0 до 1 и суммирую их. Полученная сумма V будет иметь нормальное распределение с параметрами =n/2 и =sqrt(n/12).
2. Затем нормализую полученную сумму, с помощью формулы: Таким образом, получаем ряд Z со стандартным нормальным распределением N (=0, =1).
3. Затем преобразуем полученный ряд Z в нужный ряд X с параметрами (=0, =0.749655) при помощи формулы
4. Вычисление параметра Δy: Δy=0,05\*14,9931 = 0,749655

Получившиеся значения мат. ожидания, СКО и значения гистограммы:



Вывод: При помощи созданного собственного генератора случайных чисел методом ЦПТ и на основе заданного закона получила последовательность случайных чисел, вычислила их реальные значения математического ожидания и среднеквадратического отклонения, а затем построила частотную диаграмму для проверки распределения. Получившиеся значения полностью совпали с теоретическими.

**2) Метод Мюллера:**

Описание общего алгоритма работы для метода генерации случайных чисел по методу Мюллера:

1) Генерируем 2 случайных числа r1 и r2 от 0 до 1.

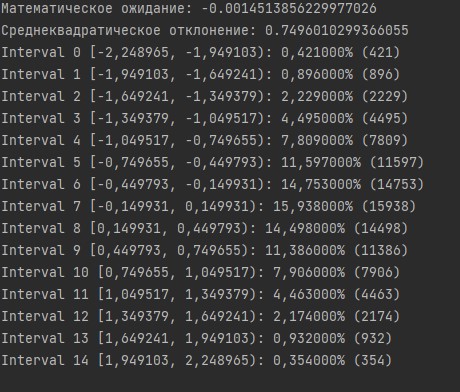
2) Вычисляем z1 и z2 по формуле и получаем 2 случайных числа со стандартным нормальным распределением N (m=0, σ=1).

z1 = cos(2π \* r1) \* sqrt(-2 \* ln(r2))

z2 = sin(2π \* r1) \* sqrt(-2 \* ln(r2))

3) Затем преобразуем полученный ряд Z в нужный ряд с параметрами (m=0, σ=Δy=0.749655) при помощи формулы x=Z⋅σ +m

Получившиеся значения мат. ожидания, СКО и значения гистограммы:



Вывод: при помощи созданного собственного генератора случайных чисел методом Мюллера и на основе заданного закона получила последовательность случайных чисел, вычислила их реальные значения математического ожидания и среднеквадратического отклонения, а затем построила частотную диаграмму для проверки распределения. Получившиеся значения полностью совпали с теоретическими.

2)

**Код:**

1. **Проверка встроенного ГСЧ**

import java.util.Random;

public class RandomNumbersTest {

public static void main(String[] args) {

int n = 100000;

int numOfIntervals = 10;

double[] count = new double[numOfIntervals];

double sum = 0, sumOfSquares = 0;

double mTheor = 0.5;

Random random = new Random();

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

double randValue = random.nextDouble();

sum += randValue;

sumOfSquares += Math.pow(randValue - mTheor, 2);

int intervalIndex = (int) (randValue \* numOfIntervals);

count[intervalIndex]++;

}

double m = sum / n;

double D = sumOfSquares / n;

double sigma = Math.sqrt(D);

System.out.println("Мат. ожидание: " + m);

System.out.println("Дисперсия : " + D);

System.out.println("Среднеквадратическое отклонение: " + sigma);

for (int i = 0; i < numOfIntervals; i++) {

double percentage = (count[i] / n) \* 100;

System.out.printf("Interval %d [%f, %f): %f%% (%d)%n", i, (double) i / numOfIntervals,

(double) (i + 1) / numOfIntervals,percentage, (int)count[i]);

}

}

1. **По методу ЦПТ:**

import java.util.Random;

public class CustomRandomGenerator {

private static final int RANDOM\_NUMS = 12;

private Random random;

public CustomRandomGenerator() {

random = new Random();

}

public double generateNextRandom() {

double v = 0;

for (int i = 0; i < RANDOM\_NUMS; i++) {

v += random.nextDouble();

}

double m = RANDOM\_NUMS / 2.0;

double sigma = Math.sqrt(RANDOM\_NUMS / 12.0);

double z = (v - m) / sigma;

return 0.749655 \* z;

}

}

public class CustomRandomTest {

public static void main(String[] args) {

int n = 100000;

int numOfIntervals = 16;

double[] intervals = new double[numOfIntervals];

double mx = 0.0;

double s = 0.749655;

double MIN\_VALUE = mx - 3 \* s;// интервал +- 3 сигмы

double MAX\_VALUE = mx + 3 \* s;

CustomRandomGenerator generator = new CustomRandomGenerator();

double sum = 0;

double sumOfSquares = 0;

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

double randValue = generator.generateNextRandom();

sum += randValue;

sumOfSquares += randValue \* randValue;

int intervalIndex = (int) ((randValue - MIN\_VALUE) / (MAX\_VALUE - MIN\_VALUE) \* numOfIntervals);

if (intervalIndex >= 0 && intervalIndex < numOfIntervals) {

intervals[intervalIndex]++;

}

}

double m = sum / n;

double D = (sumOfSquares / n) - (m \* m);

double sigma = Math.sqrt(D);

System.out.println("Мат. ожидание: " + m);

System.out.println("Среднеквадратическое отконение: " + sigma);

for (int i = 0; i < numOfIntervals; i++) {

double intervalStart = MIN\_VALUE + i \* (MAX\_VALUE - MIN\_VALUE) / numOfIntervals;

double intervalEnd = MIN\_VALUE + (i + 1) \* (MAX\_VALUE - MIN\_VALUE) / numOfIntervals;

double percentage = (intervals[i] / n) \* 100;

System.out.printf("Interval %d [%f, %f): %f%% (%d)%n",

i, intervalStart, intervalEnd,

percentage, (int)intervals[i]);

}

}

}

1. **По методу Мюллера:**

import java.util.Random;

public class CustomRandomGenerator {

private Random random;

public CustomRandomGenerator() {

random = new Random();

}

public double[] generateRandomNumberMuller() {

double r1 = random.nextDouble();

double r2 = random.nextDouble();

double z1 = Math.cos(2 \* Math.PI \* r1) \* Math.sqrt(-2 \* Math.log(r2));

double z2 = Math.sin(2 \* Math.PI \* r1) \* Math.sqrt(-2 \* Math.log(r2));

return new double[]{0.749655 \* z1, 0.749655 \* z2};

}

}

public class CustomRandomTest {

public static void main(String[] args) {

int n = 100000;

int numOfIntervals = 15;

double[] intervals = new double[numOfIntervals];

double mx = 0.0;

double s = 0.749655;

double MIN\_VALUE = mx - 3 \* s;

double MAX\_VALUE = mx + 3 \* s;

CustomRandomGenerator generator = new CustomRandomGenerator();

double sum = 0;

double sumOfSquares = 0;

for (int i = 0; i < n / 2; i++) {

double[] randValues = generator.generateRandomNumberMuller();

for (double randValue : randValues) {

sum += randValue;

sumOfSquares += randValue \* randValue;

int intervalIndex = (int) ((randValue - MIN\_VALUE) / (MAX\_VALUE - MIN\_VALUE) \* numOfIntervals);

if (intervalIndex >= 0 && intervalIndex < numOfIntervals) {

intervals[intervalIndex]++;

}

}

}

double m = sum / n;

double D = (sumOfSquares / n ) - (m \* m);

double sigma = Math.sqrt(D);

System.out.println("Математическое ожидание: " + m);

System.out.println("Среднеквадратическое отклонение: " + sigma);

for (int i = 0; i < numOfIntervals; i++) {

double intervalStart = MIN\_VALUE + i \* (MAX\_VALUE - MIN\_VALUE) / numOfIntervals;

double intervalEnd = MIN\_VALUE + (i + 1) \* (MAX\_VALUE - MIN\_VALUE) / numOfIntervals;

double percentage = (intervals[i] / n ) \* 100;

System.out.printf("Interval %d [%f, %f): %f%% (%d)%n",

i, intervalStart, intervalEnd,

percentage, (int) intervals[i]);

}

}

}