Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

ОТЧËТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине «Моделирование»

Выполнил: Студент гр. ИВ-622 Тимофеев Д.А.

Проверила: Ассистент Кафедры ВС

Петухова Я.В.

Новосибирск 2020

ИЗУЧЕНИЕ И РЕАЛИЗАЦИЯ НЕПРЕРЫВНОГО РАСПРЕДЕЛЕНИЯ РАЗЛИЧНЫМИ МЕТОДАМИ

ИВ-622 Тимофеев Д.А.

СОДЕРЖАНИЕ:

[СОДЕРЖАНИЕ: 1](#_Toc39608697)

[1. Постановка задачи 1](#_Toc39608698)

[2. Теория 1](#_Toc39608699)

[2.1. Непрерывные распределения 1](#_Toc39608700)

[2.2. Метод отбраковки 1](#_Toc39608701)

[2.3. Дискретное распределение с возвратом 1](#_Toc39608702)

[3. Выполнение работы 2](#_Toc39608703)

[4. Результат работы 3](#_Toc39608704)

[4.1. Результат работы метода отбраковки 3](#_Toc39608705)

[4.2. Моделирование дискретного распределения случайной величины с возвратом 3](#_Toc39608706)

[4.3. Моделирование дискретного распределения случайной величины без возврата 4](#_Toc39608707)

[5. Вывод 4](#_Toc39608708)

[6. Листинг 5](#_Toc39608709)

1. Постановка задачи

Реализовать:

1. Непрерывное распределение методом отбраковки
2. Дискретное распределение с возвратом
3. Дискретное распределение без воврата
4. Теория
   1. Непрерывные распределения

Существует несколько методов генерации независимых случайных величин с заданным законом распределения. Наиболее точные из них основаны на преобразовании случайных величин. Так, большое количество датчиков получается исходя из известного результата о равномерном на [0, 1) распределении функции , где η — произвольная непрерывная случайная величина с функцией распределения .

* 1. Метод отбраковки

В некоторых случаях требуется точное соответствие заданному закону распределения при отсутствии эффективных методов генерации. В такой ситуации для ограниченных случайных величин можно использовать следующий метод. Функция плотности распределения вероятностей случайных величин вписывается в прямоугольник *(a, b) × (0, c)*, такой, что *a* и *b* соответствуют границам диапазона изменения случайных величин *η*, *а c* – максимальному значению функции плотности еѐ распределения. Тогда очередная реализация случайных величин определяется по следующему алгоритму:

Шаги выполнения

1. Получить два независимых случайных числа и .
2. Если , то выдать в качестве результата. Иначе повторить Шаг 1.
   1. Дискретное распределение с возвратом

Есть n случайных величин с одинаковой вероятностью (при следующих выборках вероятность распределяется поровну между величинами), мы выбираем следующих величин без повторений, проделываем это большое количество раз и считаем частоты этих значений.

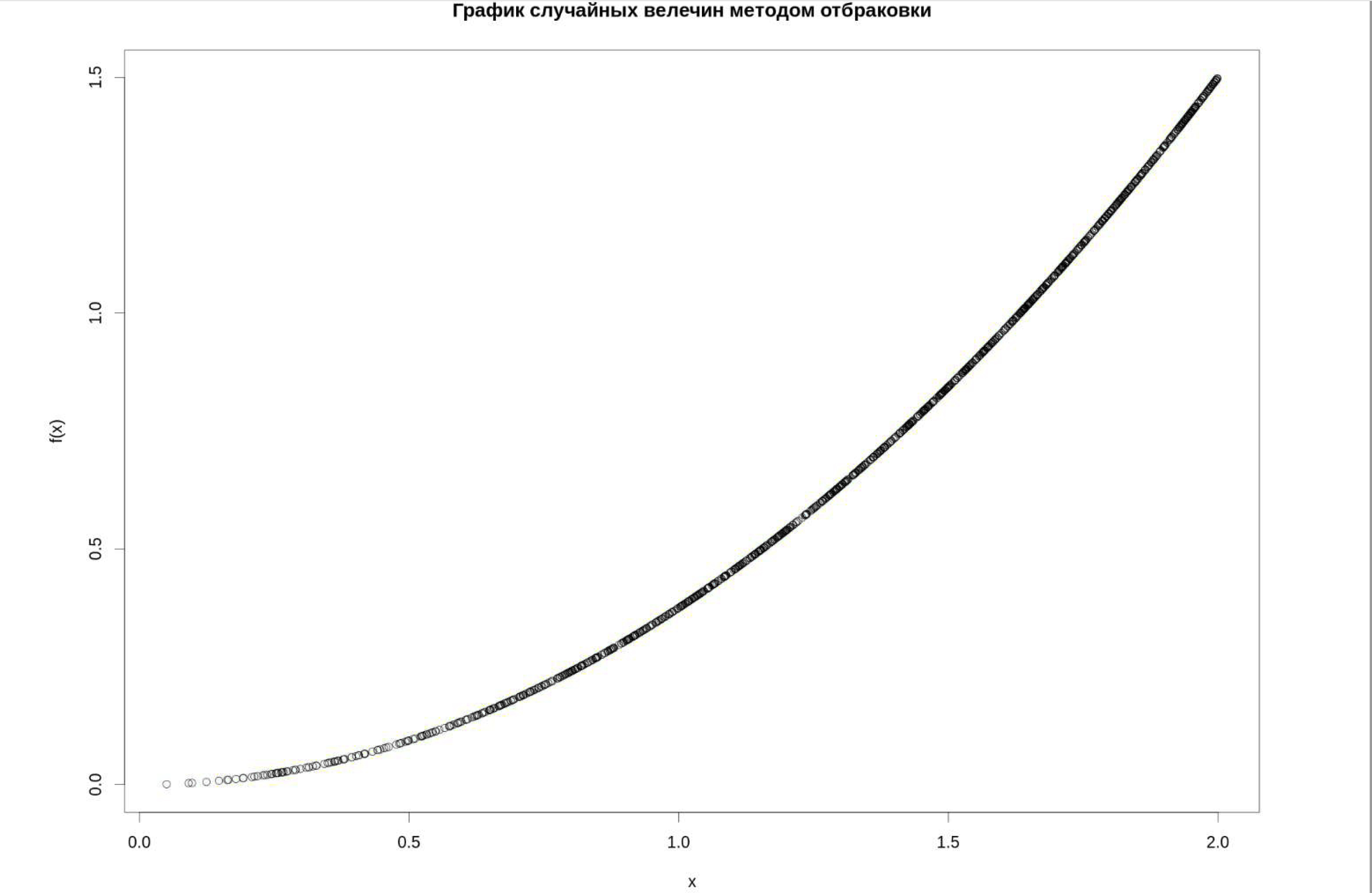
1. Выполнение работы

Пусть случайная величина Х задана плотностью вероятности

Известно, что несобственный интеграл от плотности вероятности есть вероятность достоверного события (условие нормировки):

Тогда функция плотности распределения имеет вид:

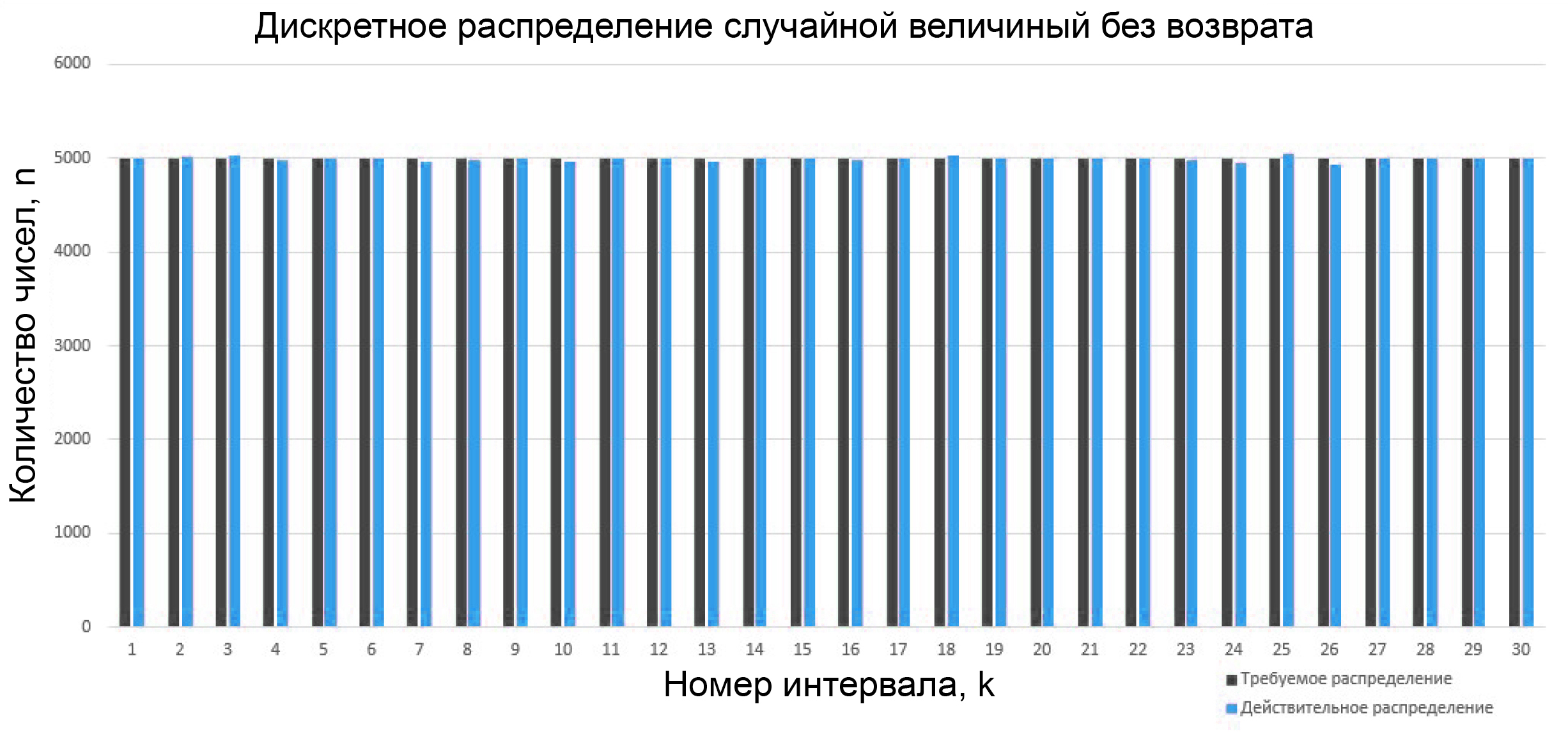
1. Результат работы
   1. Результат работы метода отбраковки



* 1. Моделирование дискретного распределения случайной величины с возвратом



* 1. Моделирование дискретного распределения случайной величины без возврата



1. Вывод

В данной лабораторной работе мы изучили и реализовали непрерывное распределение методом отбраковки, дискретное распределение с возвратом, дискретное распределение без возврата. В качестве генератора случайных чисел был выбран генератор SplittableRandom языка Java из библиотеки java.util.SplittableRandom.

По результатам работы метода отбраковки для заданной функции плотности можно сделать вывод, что с его помощью удалось смоделировать эту функцию, графики для обоих случаев получились идентичны (при построении по заданной функции и при построении с помощью метода отбраковки). К недостаткам данного метода можно отнести то, что точки, не попавшие в интервал, отбрасываются, несмотря на то что было потрачено время на их генерацию. Также данный метод не эффективен для распределений с длинными «хвостами», поскольку в этом случае имеются

частые повторные испытания.

По результатам моделирования дискретных случайных величин с реализацией выборки с возвратом и без возврата, можно заметить, что исследуемый генератор случайных чисел выдает распределение близкое к требуемому, из чего можно сделать вывод, что генератор имеет равномерное распределение с малой долей погрешности.

1. Листинг

Main.java

|  |
| --- |
| **import java.io.\*; import java.util.ArrayList; import java.util.List; import java.util.SplittableRandom; import java.util.Vector;  import static java.lang.Math.\*;  public class Main {  public static double f(double** x**) {  if (**x **>= 1 &&** x **<= 3) return (**x **\*** x **\*** x **- 1) / 18;  else return 0;  }   public static void rejection(int** max\_n**) throws IOException {  double a = 1, b = 3, c = f(b), xsi1, xsi2;  String** str **= null;  FileWriter file = new FileWriter("file1.txt");  for (int** i **= 0;** i **<** max\_n**;** i**++) {  xsi1 = new SplittableRandom().nextDouble(0,** max\_n**);  xsi2 = new SplittableRandom().nextDouble(0,** max\_n**);  double def = a + ((b - a) \* xsi1);  if (def > (c \* xsi2)) {  double fabs\_fun = abs(f(def));** str **+= def + " " + fabs\_fun;  file.write(**str**);  }  }  file.close();  }   public static void with\_return(int** max\_n**, int** n**) throws IOException {  double[] probability = new double[**n**];  double[] hit\_to\_int = new double[**n**];  double** chance\_to\_minus **= 1;  for (int** i **= 0;** i **<** n**;** i**++) probability[**i**] = 1 /** n**;  for (int** i **= 0;** i **<** n**;** i**++) {  double rand\_num1 = new SplittableRandom().nextDouble(0,** max\_n**);  probability[**i**] = abs((rand\_num1 % 1));** chance\_to\_minus **-= probability[**i**];  }  probability[**n **- 1] =** chance\_to\_minus**;  for (int** i **= 0;** i **<** n**;** i**++) hit\_to\_int[**i**] = 0;  for (int** i **= 0;** i **<** max\_n **\* 100;** i**++) {  double rand\_num2 = new SplittableRandom().nextDouble(0,** max\_n**);  double** summa **= 0.0;  for (int** j **= 0;** j **<** n**;** j**++) {** summa **+= probability[**j**];  if (rand\_num2 <** summa**) {  hit\_to\_int[**j**] += 1;  break;  }  }  }  String** str **= null;  FileWriter file = new FileWriter("file2.txt");  for (int** i **= 0;** i **<** n**;** i**++) {** str **+=** i **+ 1 + " " +** i **+ 1.5 +** max\_n **\* 100 \* probability[**i**] + " " + hit\_to\_int[**i**];  file.write(**str**);  }  file.close();  }   public static void without\_return(int** max\_n**, int** n**) throws IOException {  List<Integer>** array\_num1 **= new ArrayList<Integer>();  List<Integer> array\_num2 = new ArrayList<Integer>();  int** k **= 3 \*** n **/ 4, max\_n\_to\_def = (**max\_n **\* 100 /** k**) + 1;  double[] hit\_to\_int = new double[**n**];  for (int** i **= 0;** i **<** n**;** i**++) {  hit\_to\_int[**i**] = 0;  }  for (int** i **= 0;** i **<** n**;** i**++) {  array\_num2.add(**i**);  }  for (int** i **= 0;** i **< max\_n\_to\_def;** i**++) {** array\_num1 **= array\_num2;  if (**i **== max\_n\_to\_def - 1)** k **= (**max\_n **\* 100) %** k**;  for (int** j **= 0;** j **<** k**;** j**++) {  float p = (float) (1.0 / (**n **-** j**));  float num\_rand = (float) new SplittableRandom().nextDouble(0,** max\_n **\* 100);  int num\_rand\_to = (int) (num\_rand / p);  hit\_to\_int[**array\_num1**.get(num\_rand\_to)] += 1;** array\_num1**.remove(num\_rand\_to);  }  }  String** str **= null;  FileWriter file = new FileWriter("file3.txt");  for (int** i **= 0;** i **<** n**;** i**++) {** str **+=** i **+ 1 + " " +** i **+ 1.5 + " " + hit\_to\_int[**i**] + " " +** max\_n **\* 100 / 20;  file.write(**str**);  }  file.close();  }   public static void main(String[]** args**) throws IOException {  int max\_n = 5000, n = 10;  rejection(max\_n);  with\_return(max\_n, n);  without\_return(max\_n, n);  } }** |