ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

Пояснительная записка

к первой лабораторной работе по дисциплине

“Моделирование”

Выполнил:
студент гр. ИВ-621
Сенченко А.П.
Проверила:
ассистент кафедры ВС
Петухова Я.В.

Новосибирск – 2020 г.

Оглавление

1 Постановка задачи

Генерация независимых, одинаково распределенных случайных величин

1) Непрерывное распределение с помощью метода отбраковки
2) Дискретное распределение с возвратом
3) Дискретное распределение без возврата

2 ГСЧ

Эксперименты будут проводиться при помощи генератора псевдослучайных чисел, основанный на линейно конгруэнтном методе.

3 Теоретические сведения

1) Непрерывное распределение с помощью метода отбраковки

В некоторых случаях требуется точное соответствие заданному закону распределения при отсутствии эффективных методов генерации. В такой ситуации для ограниченных с.в. можно использовать следующий метод. Функция плотности распределения вероятностей с.в. $f_\eta(x)$ вписывается в прямоугольник $(\mathrm{a}, \mathrm{b}) \times (0, \mathrm{c})$, такой, что a и b соответствуют границам диапазона изменения с.в. η, а c – максимальному значению функции плотности её распределения. Тогда очередная реализация с.в. определяется по следующему алгоритму:

1. Построить функцию плотности распределения вероятностей
2. Получить два независимых случайных числа
3. Если $f_\eta(a + (b-a)\xi_1) > c\xi_2$ то выдать $a + (b-a)\xi_1$ в качестве результата. Иначе повторить шаг 2

2) Дискретное распределение с возвратом

Если x – дискретная случайная величина, принимающая значения $x_1 < x_2 < \cdots < x_i < \cdots$ с вероятностями $p_1 < p_2 < \cdots <$

$p_i < \cdots$, то Таблица *1* называется распределением дискретной случайной величины.

x_1	x_2	...	x_i	...
p_1	p_2	...	p_i	...

Таблица 1

Функция распределения случайной величины, с таким распределением имеет вид:

$$F(x) = \begin{cases} 0, & \text{при } x < x_1, \\ p_1, & \text{при } x_1 \leq x < x_2, \\ p_1 + p_2, & \text{при } x_2 \leq x < x_3, \\ \cdots & \\ p_1 + p_2 + \cdots p_{n-1}, & \text{при } x_{n-1} \leq x < x_n, \\ 1, & \text{при } x_n \leq x. \end{cases}$$

3) Дискретное распределение без возврата

Есть n случайных величин с одинаковой вероятностью (при следующих выборках вероятность распределяется поровну между величинами), мы выбираем $3/4\ n$ следующих величин без повторений, проделываем это большое количество раз и считаем частоты этих величин.

4 Ход работы

Функция плотности:

$$f(x) = \begin{cases} 0, & \text{если } x < 1 \\ \alpha\sqrt{x}, & \text{если } 1 \leq x \leq 4 \\ 0, & \text{если } 4 < x \end{cases}$$

Несобственный интеграл от плотности это истинное событие:

$$\int_{-\infty}^{\infty} f(x)dx = P\{-\infty < X < \infty\} = P\{\Omega\} = 1$$

поэтому найдем α:

$$\int_{a}^{b} f(x)dx = \int_{1}^{4} \alpha\sqrt{x}\,dx = 1$$

$$\int_{\infty}^{1} 0dx + \int_{1}^{4} \alpha\sqrt{x}dx + \int_{4}^{\infty} 0dx = 0 + \int_{1}^{4} \alpha\sqrt{x}dx + 0 = 1$$

$$\left(\frac{2}{3}\alpha\sqrt{x}\right)\Big|_{1}^{4} = \frac{2}{3}\alpha * 4^{3/2} - \left(\frac{2}{3}\alpha * 1^{3/2}\right) = \frac{14}{3}\alpha - 1 = 1$$

$$\alpha = \frac{3}{14}$$

Итоговая плотность:

$$f(x) = \begin{cases} 0, & \text{если } x < 1 \\ \frac{3}{14}\sqrt{x}, & \text{если } 1 \le x \le 4 \\ 0, & \text{если } 4 < x \end{cases}$$

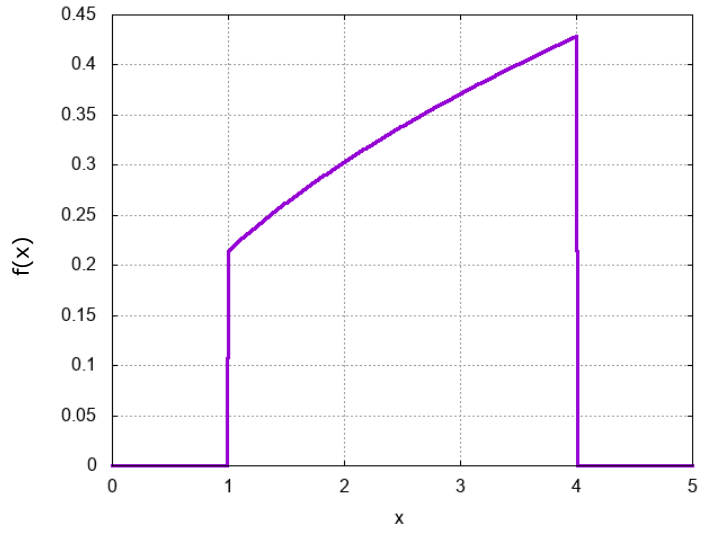

Рисунок 1. График функции плотности

5 Результаты работы программы

1) Частота распределения случайной величины

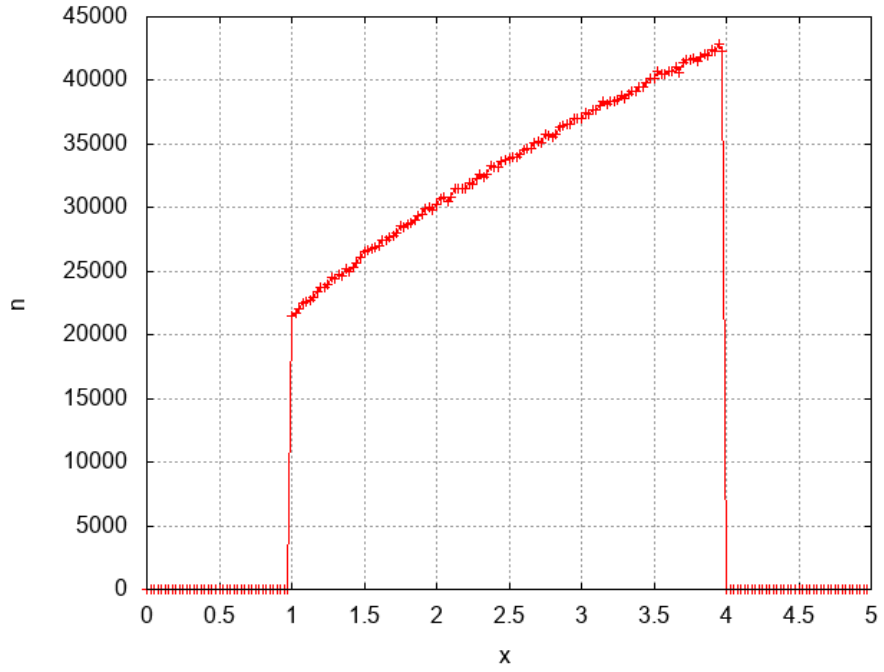

Рисунок 1. Результат работы метода отбраковки.

2) Распределения с возвратом

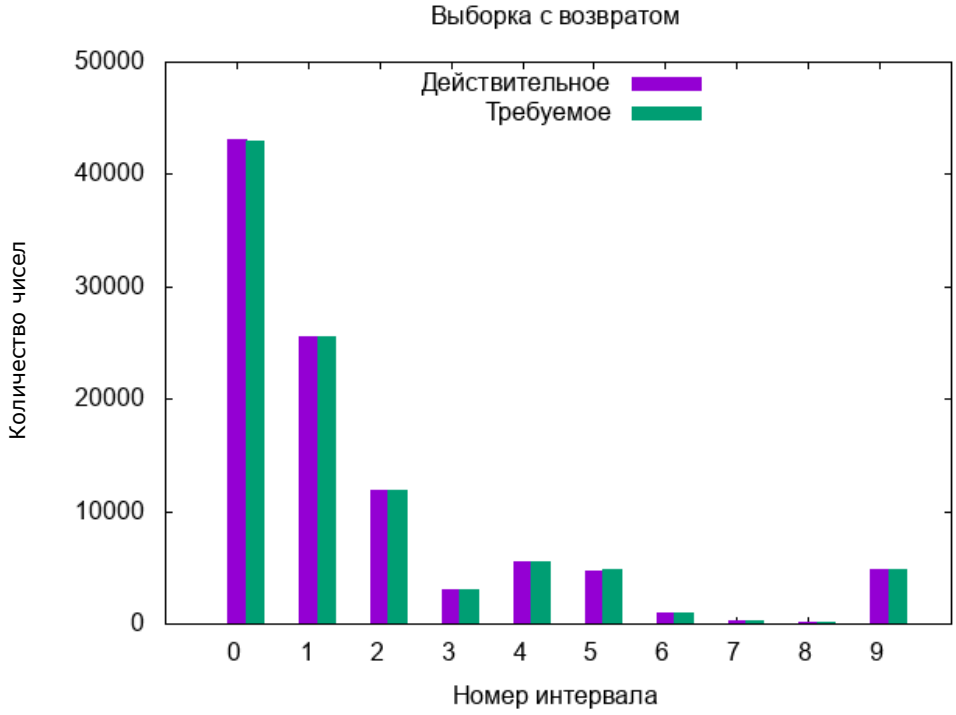

Рисунок 2. Моделирование дискретной случайно величины с возвратом.

3) Распределение без возврата

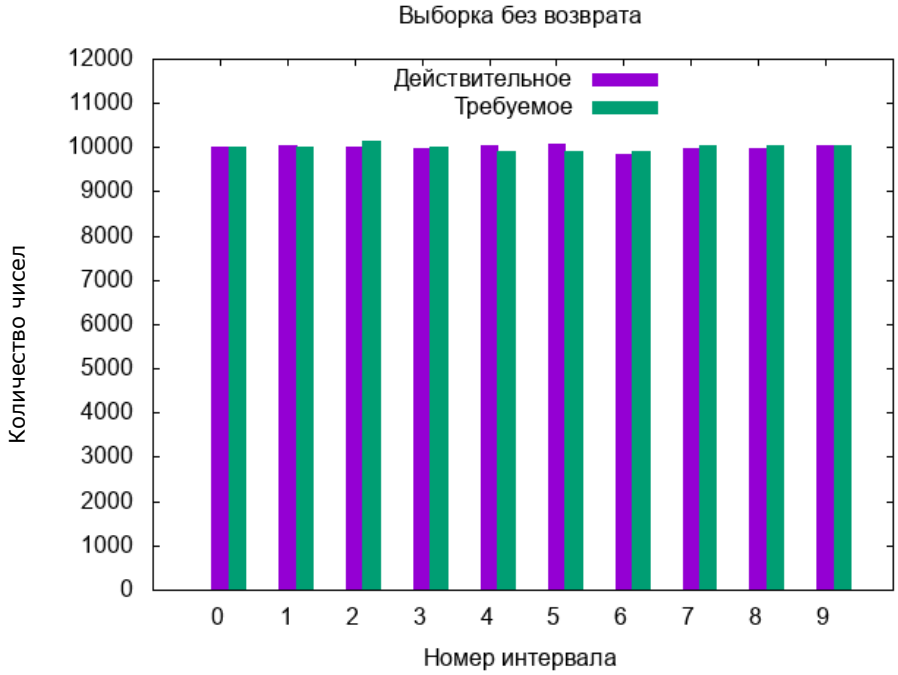

Рисунок 3. Моделирование дискретной случайной величины без возврата.

6 Выводы

1) Метод отбраковки показал свою эффективность. Благодаря ему числа были сгенерированы в соответствии с заданным законом распределением. По экспериментально полученному графику видно, что он схож с теоретическим графиком плотности распределения $f(x)$. Стоит заметить, что при методе отбраковки присутствует вероятность, что сгенерированная точка попадет выше требуемой области, из-за этого придется генерировать точку до тех пока она не окажется ниже. Данный факт может заметно увеличить время работы алгоритма.
2) На основе проведенных экспериментам, дискретное распределение с возвратом и без возврата, можно увидеть, что отклонение действительного от требуемого несильно отличается. Из этого можно сделать вывод, что используемый генератор псевдослучайных чисел имеет равномерное распределение с малой долей погрешности.

Список литературы

1) Математическое ожидание непрерывной случайной величины. Примеры решения. [Электронный ресурс]. URL: https://math.semestr.ru/math/example-expectation-continuous.php (Дата обращения 22.03.2020).
2) Распределение вероятностей. [Электронный ресурс]. URL: http://statistica.ru/theory/raspredeleniya-veroyatnostey/ (Дата обращения 22.03.2020).
3) Распределение вероятностей. [Электронный ресурс]. URL: http://termist.com/gloss/6221/p_n/p_n_1.htm (Дата обращения 22.03.2020).

Листинг программы

main.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <cmath>
#include <string>
#include <string.h>
#include <fstream>
#include <vector>
#include <chrono>
#include <random>

using namespace std;

#define MAX_EPOCH 100000

double getRand(double a, double b) {
  long seed =
chrono::system_clock::now().time_since_epoch().count();

  default_random_engine rand_generator(seed);
  uniform_real_distribution<double> distribution(a, b);

  return distribution(rand_generator);
}

double get_rand_congruent() {
  return ((double) rand()) / RAND_MAX;
}

int get_rand_congruent_int() {
  return rand();
}
```

```
double get_rand_mersenne() {
  static random_device rd;
  static mt19937_64 mersenne(rd());
  return ((double) mersenne()) / RAND_MAX;
}

double f(double x) {
  double y, a = 3.0 / 14.0;

  if (1.0 <= x && x <= 4.0) {
    y = a * sqrt(x);
  } else {
    y = 0.0;
  }

  return y;
}

double rejection_method(double a, double b, double c,
                        double (*f)(double)) {
  double x1, x2;

  do {
#if DEF
    x1 = getRand(0.0, 1.0);
    x2 = getRand(0.0, 1.0);
#elif CONG
    x1 = get_rand_congruent();
    x2 = get_rand_congruent();
#elif MER
    x1 = get_rand_mersenne();
    x2 = get_rand_mersenne();
#endif
  } while (f(a + (b - a) * x1) <= c * x2);
```

```
  return a + (b - a) * x1;
}

vector<double> get_started_distribution(uint64_t size) {
  vector<double> distribution(size);
  double residue_chance = 1, rand_num;

  for (uint64_t i = 0; i < size - 1; i++) {
#if DEF
    rand_num = getRand(0.0, 1.0);
#elif CONG
    rand_num = get_rand_congruent();
#elif MER
    rand_num = get_rand_mersenne();
#endif
    double probability = abs(remainder(rand_num,
residue_chance));
    distribution[i] = probability;
    residue_chance -= probability;
  }

  distribution[size - 1] = residue_chance;

  double sum = 0.0;
  for (int i = 0; i < size; i++)
    sum += distribution[i];

  return distribution;
}

void repeat(int generated_numbers) {
  vector<double> distribution =
get_started_distribution(generated_numbers);
  double chance;
```

```
  vector<int> hit(generated_numbers);

  for (int i = 0; i < MAX_EPOCH; i++) {
#if DEF
    chance = getRand(0.0, 1.0);
#elif CONG
    chance = get_rand_congruent();
#elif MER
    chance = get_rand_mersenne();
#endif
    double sum = 0;

    for (int j = 0; j < generated_numbers; j++) {
      sum += distribution[j];

      if (chance < sum) {
        hit[j]++;
        break;
      }
    }
  }

  ofstream output("repeat");
  output << "Number\tДействительное\tТребуемое" << endl;
  float step = ((float) MAX_EPOCH) / generated_numbers;
  float sum = 0.0;
  for (int i = 0; i < generated_numbers; i++)
    output << i << "\t" << hit[i] << "\t"
           << distribution[i] * MAX_EPOCH
           << endl;
  output.close();
  system("gnuplot repeat.plt");
}

vector<int> get_array_random_numbers(int N, int max) {
```

```
  vector<int> array(N);
  for (int i = 0; i < N; i++)
    array[i] = get_rand_congruent_int() % max;
  return array;
}

vector<int> get_counts_unique(int N, int max) {
  vector<int> counts(max);
  vector<int> random_nambers = get_array_random_nambers(N,
max);
  for (int elemet : random_nambers)
    counts[elemet]++;
  return counts;
}

vector<double> get_probablity(int N, int size) {
  vector<double> probablity(size);
  vector<int> counts = get_counts_unique(N, size);
  for (int i = 0; i < size; i++)
    probablity[i] = ((double) counts[i]) / N;
  return probablity;
}

void no_repeat(int generated_numbers) {
  vector<int> nums, temp, hit(generated_numbers);
  int k = 3 * generated_numbers / 4;
  int part = MAX_EPOCH / k + 1;
  for (int i = 0; i < generated_numbers; i++)
    temp.push_back(i);
  for (int j = 0; j < part; j++) {
    nums = temp;
    if (j == part - 1)
      k = MAX_EPOCH % k;
    for (int i = 0; i < k; i++) {
```

```
      double distribution_unit = 1.0 / (generated_numbers -
i);
      double probability = get_rand_congruent();
      double ratio = probability / distribution_unit;
      int it = static_cast<int>(truncl(ratio));
      hit[nums[it]]++;
      nums.erase(nums.begin() + it);
    }
  }
  ofstream output("no_repeat");
  float step = ((float) MAX_EPOCH) / generated_numbers;
  vector<int> counts = get_counts_unique(MAX_EPOCH, 10);
  output << "Number\tДействительное\tТребуемое" << endl;
  for (int i = 0; i < generated_numbers; i++)
    output << i << "\t" << hit[i] << "\t" << counts[i]
           << endl;
  output.close();
  system("gnuplot no_repeat.plt");
}

void _discrete_distribution() {
  repeat(10);
  no_repeat(10);
}

void continuous_distribution() {
  int size = 5000;
  vector<uint64_t> hit(size);
  for (int i = 0; i < MAX_EPOCH * 1000; i++) {
    double p = rejection_method(0.0, 5.0, 1.0, f);
    auto it = static_cast<uint64_t>(truncl(p * 1000));
    hit[it]++;
  }
  ofstream output("reject");
  for (int i = 0; i < size; i += 25) {
```

```
    output << static_cast<double>(i) / 1000.0
           << "\t" << hit[i] << endl;
  }
  output.close();
  system("gnuplot reject.plt");
}

void distribution_density() {
  ofstream output("density");
  for (double i = 0.0; i < 5.0; i += 0.01)
    output << i << " " << f(i) << endl;
  output.close();
  system("gnuplot density.plt");
}

int main(int argc, char const *argv[]) {
#if CONG
  srand(time(0));
#endif
  distribution_density();
  continuous_distribution();
  _discrete_distribution();
  return 0;
}
```