Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)
Кафедра вычислительных систем (ВС)

Лабораторная работа №1
по дисциплине «Моделирование»

Выполнил:
студент гр. ИВ-622
Лейка К.Э

Работу проверила:
Ассистент кафедры ВС
Петухова Я.В.

Новосибирск 2020

Оглавление

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.

Теоретические сведения

Метод отбраковки

Законы распределения вероятности могут быть заданы различными функциями. Поэтому надо уметь превращать равномерный ГСЧ в такой генератор случайных чисел, который задан произвольным законом распределения. Один из методов для моделирования непрерывной случайной величины является метод отбраковки. Суть метода заключается в том, что функция плотности распределения вероятностей случайных величин $f_\eta(x)$ вписывается в прямоугольник $(a, b) \times (0, c)$, такой что a и b соответствует границам диапазона изменения случайных величин η, а c - максимальному значению функции плотности её распределения. Тогда очередная реализация случайных величин определяется по следующему алгоритму:

- выбирается функция плотности распределения случайных величин;
- получить два независимых случайных числа ξ_1 и ξ_2;
- если $f_\eta(a + (b - a) * \xi_1) > c * \xi_2$ то выдать $a + (b - a) * \xi_1$ в качестве результата, иначе повторить предыдущий шаг.

Дискретное распределение

Дискретные распределения используются для описания событий с недифференцируемыми характеристиками, определёнными в изолированных точках. Проще говоря, для событий, исход которых может быть отнесён к некоторой дискретной категории: успех или неудача, целое число (например, игра в рулетку, в кости), орёл или решка и т.д.

Описывается дискретное распределение вероятностью наступления каждого из возможных исходов события. Как и для любого распределения (в том числе непрерывного) для дискретных событий определены понятия математическое ожидания и дисперсии. Однако, следует понимать, что математическое ожидание для дискретного случайного события — величина в общем случае нереализуемая как исход одиночного случайного события, а скорее как величина, к которой будет стремиться среднее арифметическое исходов событий при увеличении их количества.

В моделировании дискретных случайных событий важную роль играет комбинаторика, так как вероятность исхода события можно определить как отношение количества комбинаций, дающих требуемый исход к общему количеству комбинаций.

Дискретное распределение с возвратом

Случайная величина *X* называется дискретной, если она может принимать дискретное множество значений ($x_1, x_2, x_3, \dots, x_n$). Дискретная случайная величина описывается с помощью таблицы (распределение случайной величины *X*) или в виде аналитической формулы.

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & \cdots & x_n \\ p_1 & p_2 & p_3 & \cdots & p_n \end{pmatrix}, \text{где}$$

$(x_1, x_2, x_3, \dots, x_n)$ – возможные значения величины *X*;

$(p_1, p_2, p_3, \dots, p_n)$ – соответствующие им вероятности:

$$P(X = x_i) = p_i$$

Числа могут быть любыми, а вероятности должны удовлетворять двум условиям:

- $p_i > 0$;
- $\sum_{i=1}^{n} p_i = 1$.

Функция распределения принимает вид:

$$F(x) = \sum_{x_k < x} P(X = x_k)$$

Наиболее общий вид построения генератора дискретных случайных величин основывается на следующем алгоритме. Пусть имеется таблица пар (x_i, p_i). Тогда сумму можно представить интервалом [0,1), разбитым на

полуинтервалы $[v_{i-1}, v_i), v_0 = 0, v_n = 1$ длины p_i. Случайная величина X с неизбежностью принадлежит одному из этих полуинтервалов, тем самым определяя индекс дискретного значения. Номера полуинтервала определяется как:

$$min\{i|X < v_i\} = min\left\{ i \middle| X < \sum_{j=1}^{i} p_i \right\}$$

Дискретное распределение без возврата

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

Универсальный алгоритм для выборки без повторения.

Входные данные: n – число индексов; m – длина выборки. Выходные данные: $L[1..m]$ – результирующий вектор индексов.

Выполнение и оценка результатов эксперимента

Выбрана плотность распределения:

$$f(x) = \begin{cases} 0, x \le 1; \\ A(x-1), 1 < x \le 3; \\ 0, x > 3. \end{cases}$$

Найдем параметр A из условия нормировки $\int_{-\infty}^{\infty} f(x)dx = 1$. Получаем:

$$\int_{-\infty}^{\infty} f(x)dx = \int_{1}^{3} A(x-1)dx = A\int_{1}^{3}(x-1)dx = A\left(\frac{1}{2}x^2 - x\right)\Big|_1^3 =$$

$$= A\left(\frac{1}{2} * 9 - 3 - \frac{1}{2} + 1\right) = 2A = 1, \qquad \text{откуда } A = {}^1\!/_2.$$

Тогда функция плотности примет вид:

$$f(x) = \begin{cases} 0, \text{при } x \le 1; \\ \dfrac{x-1}{2}, \text{при } 1 < x \le 3; \\ 0, x > 3. \end{cases}$$

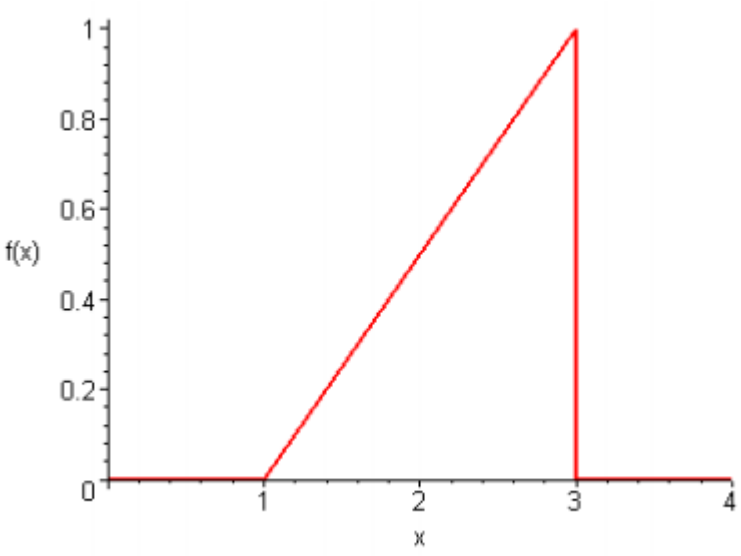

Рисунок 1 - График плотности распределения

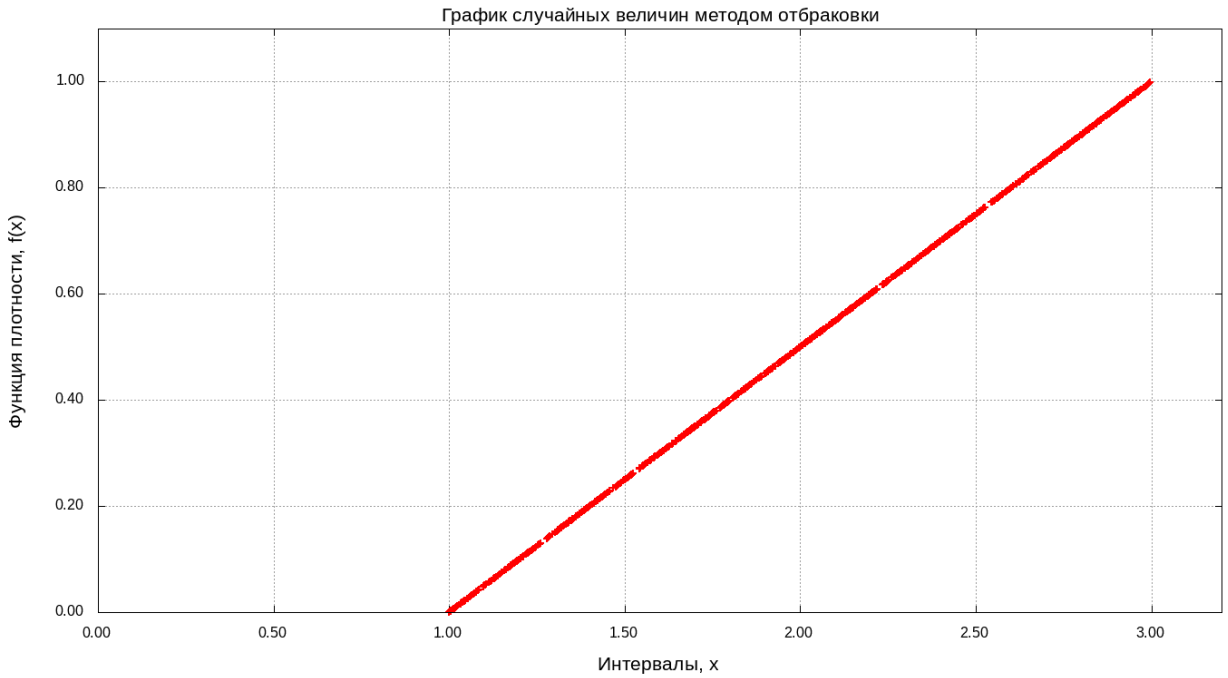

Рисунок 2 - График непрерывного распределения отработки метода отбраковки

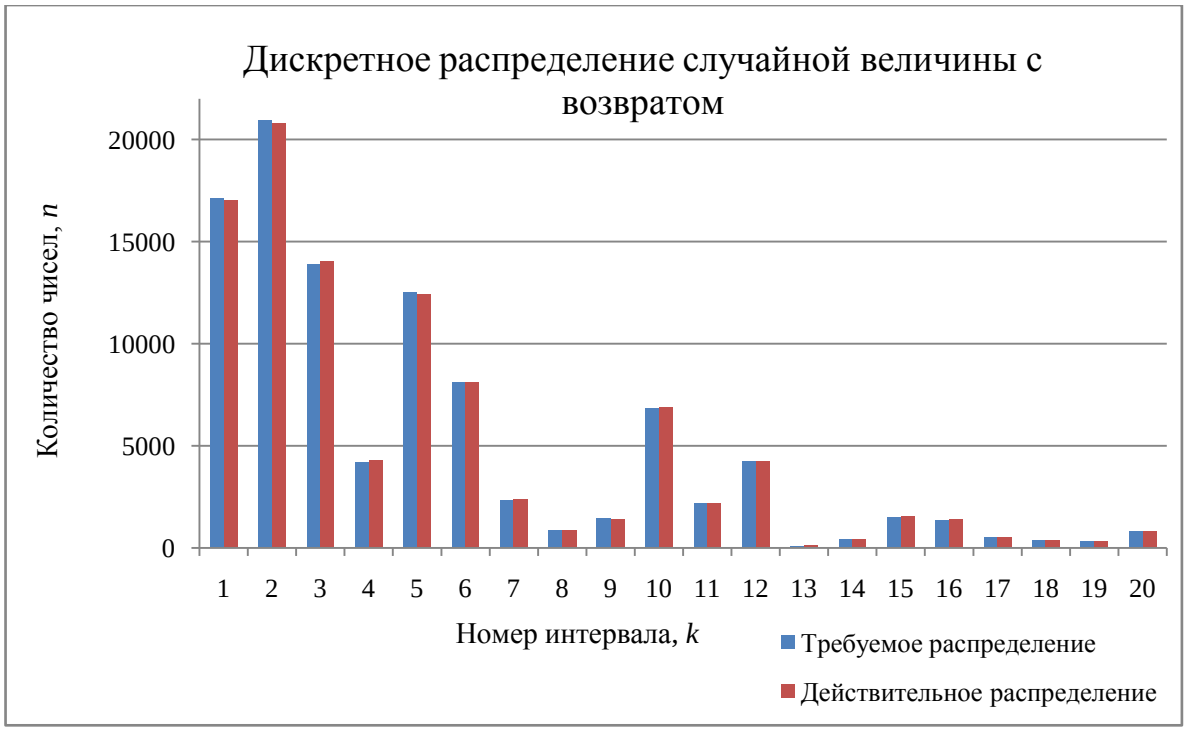

Рисунок 3 - График моделирования дискретного распределения случайной величины с возвратом

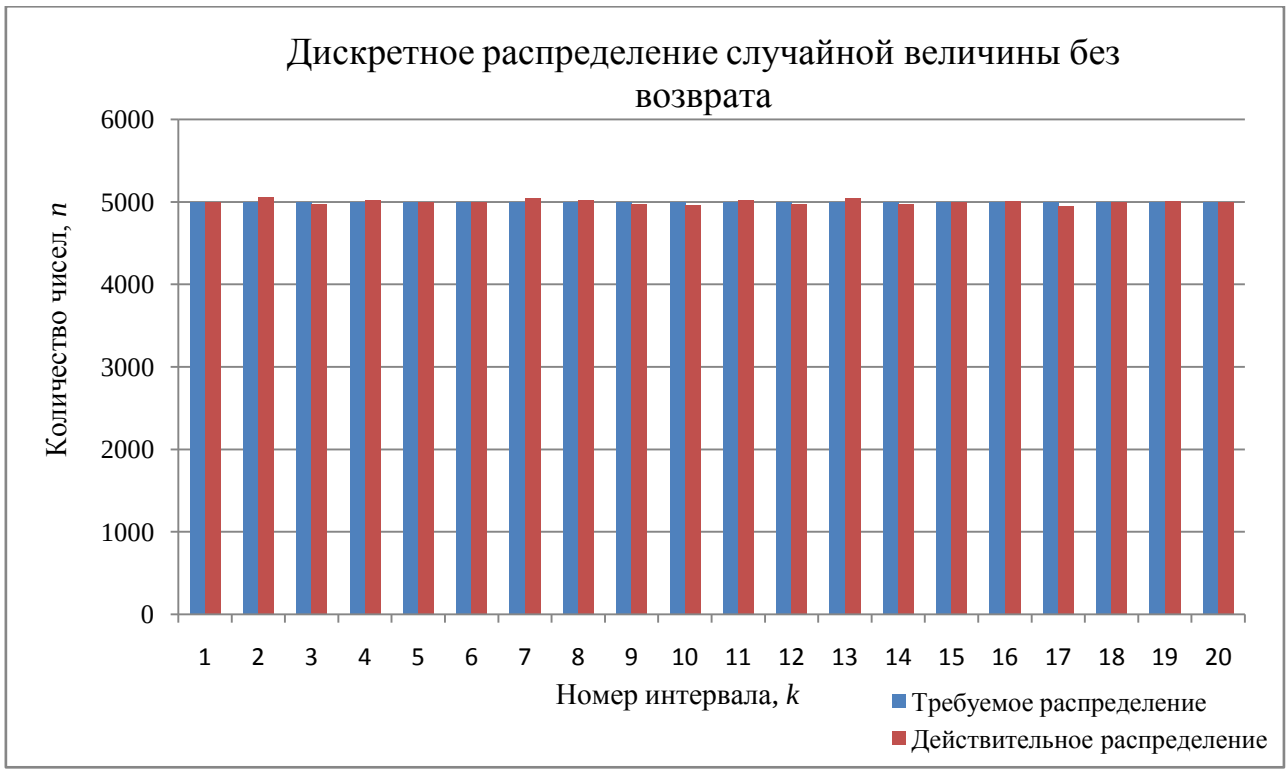

Рисунок 4 - График моделирования дискретного распределения случайной величины без возврата

Заключение

В данной работе были изучены методы моделирования непрерывных случайных величин методом отбраковки и два вида моделирования дискретных распределения (с возвратом и без него). Все результаты экспериментов были изображены на соответствующих графиках и гистограммах. В качестве генератора случайных чисел был взят `DRNG` (цифровой генератор случайных чисел) от компании Intel, на котором были написаны исследуемые методы моделирования случайных величин.

По итогу работы метода отбраковки можно сделать вывод, что данный метод смог смоделировать выданную ему функцию плотности достаточно точно. Если сравнить результат работы метода с графиком плотности, то можно заметить идентичность графиков. Но имеются несколько недостатков данного метода:

- при вычислении точек, которые не попали на линию функции плотности распределения вероятности, отбрасываются. Но это сказывается на общем времени работы алгоритма, так как эти точки генерируются и рассчитываются;
- данный метод применятся для аналитических функций. Он неэффективен для распределений с длинными «хвостами», поскольку в этом случае имеются частые повторные испытания.

По результатам исследования дискретных случайных величин двумя методами (это с возвратом и без него) можно сказать, что цифровой генератор случайных чисел от компании Intel выдает близкое к равномерному распределению, это можно увидеть на его результатах выполнения с возвратом и без него. По данным результатам действительное распределение приблизительно равно требуемому распределению.

Листинг программы

Метод отбраковки:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <limits.h>
float funcplot(float x){
    return (x-1)/2;
}
float reject(float a, float b, float ksi){
    return a+(b-a)*ksi;
}
char randoms(float *randf, float min, float max){
    int retries= 10;
    unsigned long long rand64;

    while(retries--) {
        if (  __builtin_ia32_rdrand64_step(&rand64) ) {
            *randf= (float)rand64/ULONG_MAX*(max - min) + min;
            return 1;
        }
    }
    return 0;
}
int main(int argc, char const *argv[]){
     srand(time(NULL));
     int N;
     float a,b,ksi1,ksi2,c;
     a=1;b=3;c=(b-1)/2;
     FILE *Data1;
     Data1 = fopen("tmp.dat", "w");
     N=1000;
     for (int i =0;i<N;i++){
          randoms(&ksi1,0.0, 1.0);
          randoms(&ksi2,0.0, 1.0);
          float tmp = reject(a,b,ksi1);
          if(tmp > (c*ksi2)){
             fprintf(Data1,"%f %f\n",tmp,fabs(funcplot(tmp)));
         }
         else
             continue;
    }
     fclose(Data1);
     return 0;
}
```

Дискретное распределение с возвратом:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <cmath>
#include <string>
#include <string.h>
#include <fstream>
#include <vector>
#include <climits>
#define MAX_GENERATED_ARRAY 100000
char randoms(float *randf, float min, float max){
    int retries= 10;
    unsigned long long rand64;

    while(retries--) {
        if (  __builtin_ia32_rdrand64_step(&rand64) ) {
            *randf= (float)rand64/ULONG_MAX*(max - min) + min;
            return 1;
        }
    }
    return 0;
}
int main(int argc, char const *argv[]){
    int generated_numbers=20;
    float residue_chance = 1, chance_hit[generated_numbers], chance;
    int hit[generated_numbers];
    memset(hit, 0, sizeof(int) * generated_numbers);
    for (int number = 0; number < generated_numbers - 1; number++){
        float r;randoms(&r,0.0, 1.0);
        chance_hit[number] =abs(remainder(r, residue_chance));
        residue_chance -= chance_hit[number];
    }
    chance_hit[generated_numbers - 1] = residue_chance;
    for (int i = 0; i < MAX_GENERATED_ARRAY; i++){
        randoms(&chance,0.0, 1.0);
        float sum = 0;
        for (int j = 0; j < generated_numbers; j++){
            sum += chance_hit[j];
            if (chance < sum){
                ++hit[j];
                break;
            }
        }
```

```
    }
    FILE *fp = fopen("out_repeat.dat", "w");
    for (int number = 0; number < generated_numbers; number++){
        fprintf(fp,                                          "%d\t%.4f\t%.d\n",number+1,
chance_hit[number]*MAX_GENERATED_ARRAY, hit[number]);
        printf("%d\t%.4f\t%.d\n",number+1,      chance_hit[number]*MAX_GENERATED_ARRAY,
hit[number]);
    }

    fclose(fp);
    return 0;
}
```

Дискретное распределение без возврата:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <cmath>
#include <string>
#include <string.h>
#include <fstream>
#include <vector>
#include <climits>
using namespace std;
#define MAX_GENERATED_ARRAY 100000
char randoms(float *randf, float min, float max){
    int retries= 10;
    unsigned long long rand64;
    while(retries--) {
        if (  __builtin_ia32_rdrand64_step(&rand64) ) {
            *randf= (float)rand64/ULONG_MAX*(max - min) + min;
            return 1;
        }
    }
    return 0;
}
int main(int argc, char const *argv[]){
    vector<int> nums, temp;
    int generated_numbers=20;
    int it, k = 3 * generated_numbers / 4;
    int hit[generated_numbers], part = MAX_GENERATED_ARRAY / k + 1;
    float p;
    memset(hit, 0, sizeof(int) * generated_numbers);
```

```
    for (int number = 0; number < generated_numbers; number++)
        temp.push_back(number);

    for (int j = 0; j < part; j++){
        nums = temp;
        if (j == part - 1)
            k = MAX_GENERATED_ARRAY % k;
        for (int i = 0; i < k; i++){
            p = 1.0 / (generated_numbers - i);
            float tmp;
            randoms(&tmp,0.0,1.0);
            it = tmp / p;
            ++hit[nums[it]];
            nums.erase(nums.begin() + it);
        }
    }
    FILE *fp = fopen("out_no_repeat.dat", "w");
    for (int number = 0; number < generated_numbers; number++){
        fprintf(fp, "%d\t%.d\n", number+1, hit[number]);
        printf("%d \t%.d\n", number+1, hit[number]);
    }
    fclose(fp);
    return 0;
}
```