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

Лабораторная работа №0
по дисциплине «Моделирование»

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

Работу проверила:
Ассистент кафедры ВС
Петухова Я.В.

Новосибирск 2020

Оглавление

Постановка задачи

Взять готовую реализацию генератора псевдослучайных чисел на отрезке [0,1] и убедится в его равномерном распределении , используя такие параметры как «хи-квадрат» и автокорреляции.

Теоретические сведения

Критерий согласия Пирсона - наиболее часто употребляемый для проверки гипотезы о принадлежности некоторой выборки теоретическому закону распределения.

В задачах обычно используется следующий алгоритм:

1. Выбор теоретического закона распределения (обычно задан заранее, если не задан - анализируем выборку, например, с помощью гистограммы относительных частот, которая имитирует плотность распределения);
2. Оцениваем параметры распределения по выборке (для этого вычисляется математическое ожидание и дисперсия): a, σ для нормального, a, b - для равномерного, λ - для распределения Пуассона и т.д;
3. Вычисляются теоретические значения частот (через теоретические вероятности попадания в интервал) и сравниваются с исходными (выборочными) данными;
4. Анализируется значение статистики χ^2 и делается вывод о соответствии (или нет) теоретическому закону распределения.

Процедура проверки осуществляется с помощью критерия «хи-квадрат» χ^2:

1. Отрезок [0,1] разбивается на k равных интервалов;
2. N раз генерируется случайное число, при это должно выполнятся условие что $\frac{N}{k} > 5$;

3. Подсчитывается количество случайных сгенерированных чисел попавших в каждый из интервалов;

4. Рассчитываем критерий «хи-квадрат» χ^2 по формуле:

$$\chi^2 = \frac{(n_1 - p_1 N)^2}{p_1 N} + \frac{(n_2 - p_2 N)^2}{p_2 N} + \cdots + \frac{(n_k - p_k N)^2}{p_k N} = \sum_{i=1}^{k} \frac{(n_i - p_i N)^2}{p_i N}, \text{где}$$

- $p_i = \frac{1}{k}$ - теоретическая вероятность попадания чисел в i-ый интервал;
- k - количество интервалов;
- N - общее количество сгенерированных чисел;
- n_i - количество попавших чисел в интервал;
- χ^2- критерий, который позволяет определить , удовлетворяет ли генератор случайных чисел требованиям равномерного распределения или нет;

5. Если $\chi^2_{\text{эксп}} \leq \chi^2_{\text{таб}}$, то гипотеза H_0 не противоречит опытным данным, иначе H_0 отвергается.

Условие применения критерия Пирсона является наличие в каждом интервале не менее пяти наблюдений. И следует помнить, что для равномерного распределение все значения вероятности одинаковы. Далее надо провести корреляционный анализ. Корреляционный анализ – популярный метод статистического исследования, который используется для выявления степени зависимости одного показателя от другого.

Предназначение корреляционного анализа сводится к выявлению наличия зависимости между различными факторами. То есть, определяется, влияет ли уменьшение или увеличение одного показателя на изменение другого.

Если зависимость установлена, то определяется коэффициент корреляции. В отличие от регрессионного анализа, это единственный показатель, который рассчитывает данный метод статистического исследования. Коэффициент корреляции варьируется в диапазоне от +1 до -1. При наличии положительной корреляции увеличение одного показателя

способствует увеличению второго. При отрицательной корреляции увеличение одного показателя влечет за собой уменьшение другого. Чем больше модуль коэффициента корреляции, тем заметнее изменение одного показателя отражается на изменении второго. При коэффициенте равном 0 зависимость между ними отсутствует полностью.

Значение (по модулю)	Интерпретация
до 0,2	очень слабая корреляция
до 0,5	слабая корреляция
до 0,7	средняя корреляция
до 0,9	высокая корреляция
свыше 0,9	очень высокая корреляция

Автокорреляционная функция – предназначена для оценки корреляции между смещенными копиями последовательностей.

$$\hat{a}(\tau) = \frac{\sum_{i=1}^{n}(x_i - \bar{x}) * (x_{i+\tau} - \bar{x})}{(\mathrm{N} - \tau) * S^2(x)}, \text{где}$$

$\bar{x}$ - математическое ожидание — среднее значение случайной величины при стремлении количества выборок или количества её измерений (иногда говорят — количества испытаний) к бесконечности:

$$\acute{x} \equiv M(x) \equiv E(x) = \frac{1}{N}\sum_{i=1}^{N} x_i ;$$

$S^2(x)$ - выборочная дисперсия случайной величины — это оценка теоретической дисперсии распределения, рассчитанная на основе данных выборки:

$$S^2(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \acute{x})^2 = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - (\acute{x})^2;$$

$x_{i+\tau}$- множество значений другой случайной величины (полученной из значений прошлой случайной величины, но с некоторым смещением);

n - мощность множества случайных величин;

τ- смещение последовательности.

В качестве генераторов случайных чисел были взяты:

- стандартный генератор для языка C/C++, но с не большой доработкой для генерации вещественных чисел;

```
float get_float_rand(float l,float r){
int l_ = l*1000000;
int r_ = r * 1000000;
return (float) (rand() % (r_- l_)+ l_)
/ 1000000;
}
```

- генератор случайных чисел на основе алгоритма «Вихря Марсенна». Вихрь Мерсе́нна (англ. Mersenne twister, MT) — генератор псевдослучайных чисел (ГПСЧ), разработанный в 1997 году японскими учёными Макото Мацумото (яп. 松本眞) и Такудзи Нисимура (яп. 西村拓士). Вихрь Мерсенна основывается на свойствах простых чисел Мерсенна (отсюда название) и обеспечивает быструю генерацию высококачественных по критерию случайности псевдослучайных чисел. Вихрь Мерсенна лишён многих недостатков, присущих другим ГПСЧ, таких как малый период, предсказуемость, легко выявляемые статистические закономерности. Тем не менее, этот генератор не является криптостойким, что ограничивает его использование в криптографии. Существуют по меньшей мере два общих варианта алгоритма, различающихся только величиной используемого простого числа Мерсенна, наиболее распространённым из которых является алгоритм *MT19937*, период которого составляет $2^{19937} - 1$ (приблизительно $4{,}3 \cdot 10^{6001}$);

```
unsigned long long genrand64_int64(void)
{
int i;
unsigned long long x;
static unsigned long long mag01[2] = { 0ULL, MATRIX_A };

if (mti >= NN)
{ /* generate NN words at one time */

/* if init_genrand64() has not been called, */
```

```
/* a default initial seed is used     */
if (mti == NN + 1)
init_genrand64(5489ULL);

for (i = 0; i < NN - MM; i++)
{
x = (mt[i] & UM) | (mt[i + 1] & LM);
mt[i] = mt[i + MM] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];
}
for (; i < NN - 1; i++) {
x = (mt[i] & UM) | (mt[i + 1] & LM);
mt[i] = mt[i + (MM - NN)] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];
}
x = (mt[NN - 1] & UM) | (mt[0] & LM);
mt[NN - 1] = mt[MM - 1] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];

mti = 0;
}

x = mt[mti++];

x ^= (x >> 29) & 0x5555555555555555ULL;
x ^= (x << 17) & 0x71D67FFFEDA60000ULL;
x ^= (x << 37) & 0xFFF7EEE000000000ULL;
x ^= (x >> 43);

return x;
}

double genrand64_real1(void)
{
return (genrand64_int64() >> 11)* (1.0 / 9007199254740991.0);
}
```

- цифровой генератор случайных чисел Intel (DRNG). Intel® Secure Key с кодовым названием Bull Mountain Technology - это название Intel для инструкций по архитектуре Intel® 64 и IA-32 RDRAND и RDSEED и базовой аппаратной реализации Цифрового генератора случайных чисел (DRNG). Помимо прочего, DRNG, использующий инструкцию RDRAND, полезен для генерации высококачественных ключей для криптографических протоколов, а инструкция RSEED предназначена для заполнения программных генераторов псевдослучайных чисел (PRNG).Цифровой генератор случайных

чисел, использующий инструкцию RDRAND, представляет собой инновационный аппаратный подход к высококачественной, высокопроизводительной энтропии и генерации случайных чисел. При компиляции gcc дополнительно указывать опцию `-mrdrnd`.

```
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;
}
```

Выполнение и оценка результатов эксперимента

При запуске программы пользователю выведется сообщение о том, что надо ввести с клавиатуры целое число, которое указывает на количество генерируемых чисел. Далее, появится сообщение о вводе количества интервалов на отрезок [0,1]. После ввода чисел в программу обязательно проверятся числа на соотношение $\frac{N}{k} > 5$,если условие не выполнится, то потребуется ввести числа заново.

Затем происходит генерация чисел, их подсчет количеств попаданий на интервалы и запись данных в два файла (это файл в котором случайно сгенерированные числа и файл с количеством попаданий на интервалы).

После это происходит подсчет критерия Пирсона «хи-квадрат и автокорреляционной функции с различным смещением - τ.

```
leika@leika:~/Modelirovanie/laba0$ ./laba
Enter the number of points to generate : N = 1000000
How many intervals to divide a segment from 0 to 1, provided that the number of points must be
 5 once more than the number of intervals : k = 1000
X^2=983.441772
```

Рисунок 1 - Запуск программы с использованием стандартного для C/C++ ГСЧ функция `rand`, при N = 1 000 000 и k = 1 000

```
a(1)=-0.009483
a(2)=-0.035451
a(3)=-0.016877
a(4)=-0.045830
a(5)=-0.005712
a(6)=-0.045720
a(7)=-0.008900
a(8)=0.035326
a(9)=0.002381
a(10)=-0.011078
a(11)=0.051055
a(12)=0.013269
a(13)=-0.031051
a(14)=0.036472
a(15)=0.018755
a(16)=0.002186
a(17)=-0.043808
a(18)=0.025268
a(19)=0.001874
a(20)=0.019848
```

Рисунок 2- Результат расчета коэффициента автокорреляции генератором `rand` , при N = 1 000 000 k = 1 000 и $\boldsymbol{\tau}$ от $\mathbf{1}$ до $\boldsymbol{k/2}$

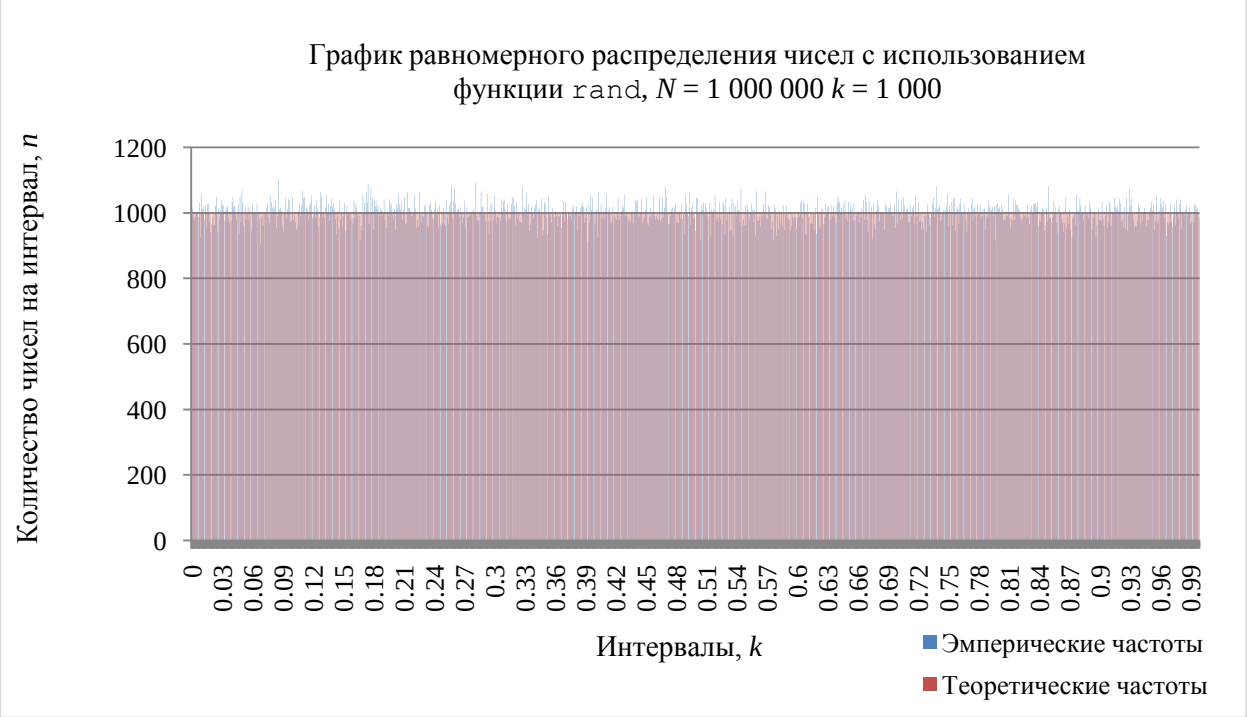

Рисунок 3 - График равномерного распределения чисел с использованием функции `rand`, N = 1 000 000 k = 1 000

```
leika@leika:~/Modelirovanie/laba0$ ./laba
Enter the number of points to generate : N = 1000000
How many intervals to divide a segment from 0 to 1, provided that the number of points must be
 5 once more than the number of intervals : k = 1000
X^2=1032.973267
```

Рисунок 4 - Запуск программы с использованием алгоритма генерации чисел "Вихрь Мерсенна", N = 1 000 000 и k = 1 000

```
a(1)=-0.025541
a(2)=-0.066667
a(3)=-0.001147
a(4)=-0.061755
a(5)=0.018272
a(6)=-0.048992
a(7)=-0.019374
a(8)=0.025509
a(9)=-0.031598
a(10)=-0.031597
a(11)=0.040035
a(12)=0.034763
a(13)=0.015655
a(14)=-0.076322
a(15)=-0.056190
a(16)=-0.054458
a(17)=-0.011244
a(18)=0.023380
a(19)=-0.026574
a(20)=-0.004156
```

Рисунок 5- Результат расчета коэффициента автокорреляции генератором "Вихрь Марсенна" , при N = 1 000 000 k = 1 000 и τ от 1 до k/2

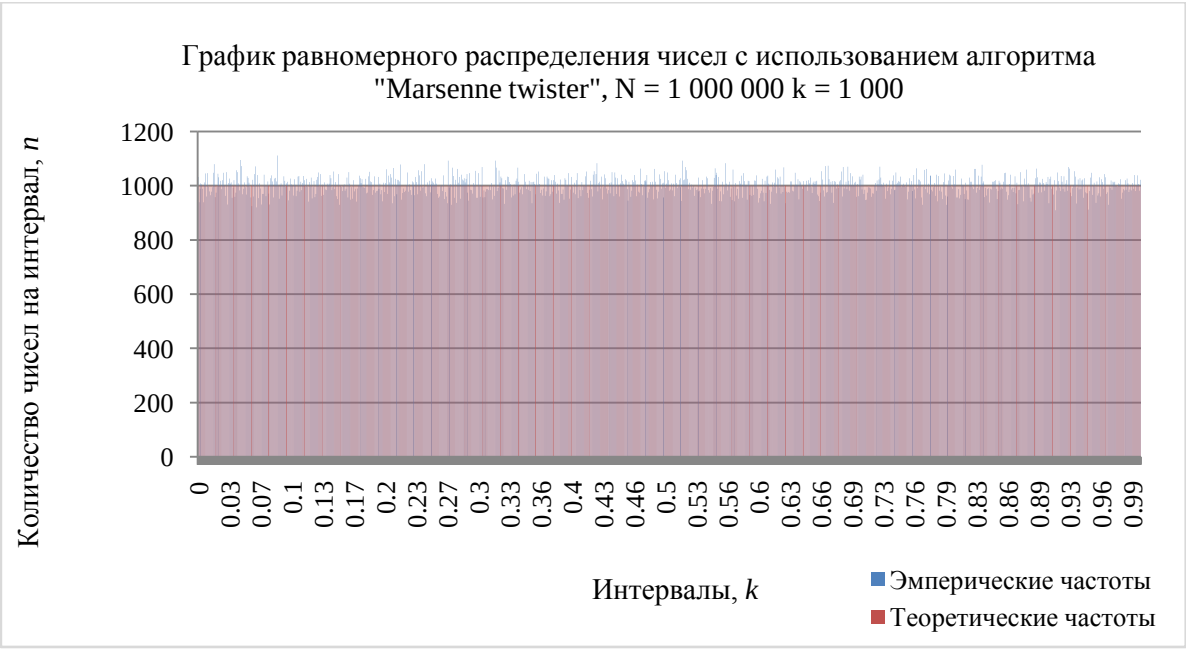

Рисунок 6 - График равномерного распределения чисел с использованием алгоритма "Marsenne twister", N = 1 000 000 k = 1 000

```
leika@leika:~/Modelirovanie/laba0$ ./laba
Enter the number of points to generate : N = 1000000
How many intervals to divide a segment from 0 to 1, provided that the number of points must be
 5 once more than the number of intervals : k = 1000
X^2=999.329407
```

Рисунок 7 - Запуск программы с использованием алгоритма цифрового генератора случайных чисел от Intel, N = 1 000 000 и k = 1 000

```
a(1)=0.003194
a(2)=0.018226
a(3)=-0.000981
a(4)=-0.012088
a(5)=0.007361
a(6)=-0.031808
a(7)=-0.067186
a(8)=-0.029503
a(9)=-0.002998
a(10)=0.039600
a(11)=-0.008219
a(12)=0.017785
a(13)=-0.013001
a(14)=-0.049454
a(15)=0.012613
a(16)=-0.019637
a(17)=0.045115
a(18)=-0.026093
a(19)=-0.032750
a(20)=-0.001349
```

Рисунок 8 - Результат расчета коэффициента автокорреляции генератором DRNG от Intel, при N = 1 000 000 k = 1 000 и τ от 1 до k/2

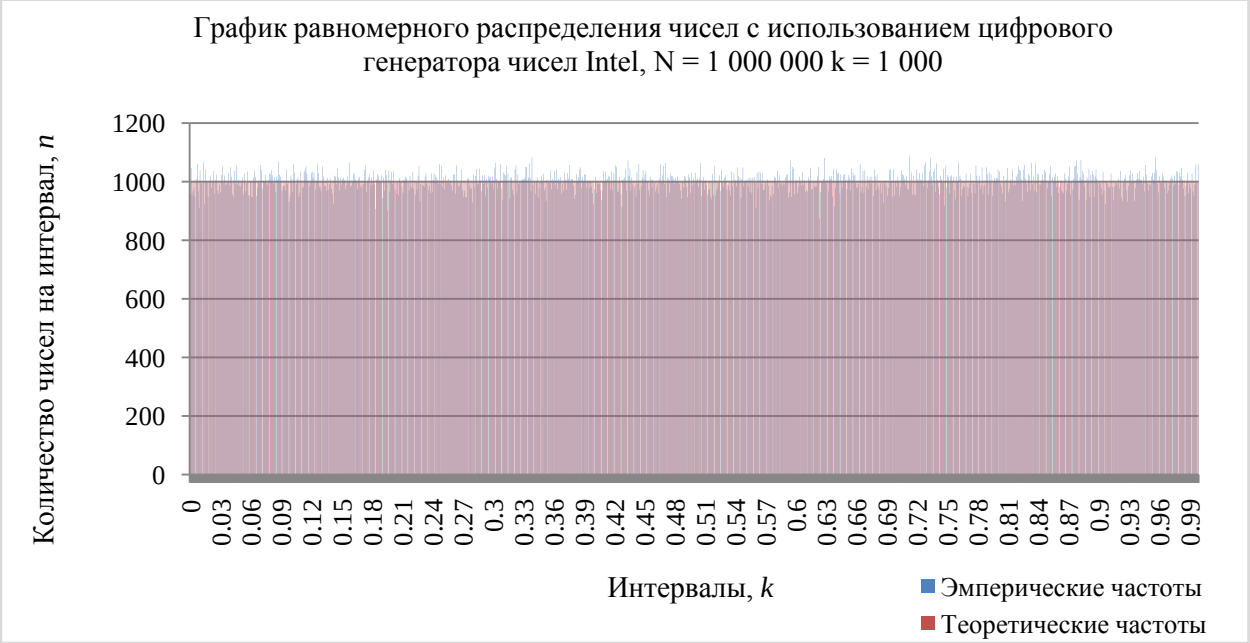

Рисунок 9 - График равномерного распределения чисел с использованием цифрового генератора чисел Intel, N = 1 000 000 k = 1 000

N - количество чисел, k—количество интервалов ГСЧ	N = 10 000 k = 1 000	N = 1 000 000 k = 1 000	N = 1 000 000 k = 100
rand	$\chi^2_{\text{эксп},rand}$ = 1047.79	$\chi^2_{\text{эксп},rand}$ = 983.442	$\chi^2_{\text{эксп},rand}$ = 88.467
Marsenne twister	$\chi^2_{\text{эксп},marsenne}$ =1003.6	$\chi^2_{\text{эксп},marsenne}$ = 1032.973	$\chi^2_{\text{эксп},marsenne}$ =74.412
DRND	$\chi^2_{\text{эксп},DRNG}$=979.399	$\chi^2_{\text{эксп},DRNG}$= 999.329	$\chi^2_{\text{эксп},DRNG}$=82.112

Заключение

Входе данной лабораторной работе были проведены ряд экспериментов по исследованию равномерного распределения в трех независимых генераторов псевдослучайных чисел в языке программирования C/C++.

По результатам экспериментов видно, что критерии «хи-квадрат» равны $\chi^2_{\text{эксп},rand}$ = 983.442, $\chi^2_{\text{эксп},marsenne}$ = 1032.973, $\chi^2_{\text{эксп},DRNG}$ = 999.329 , в тоже время «хи-квадрат» табличного равен $\chi^2_{\text{таб}}$=1142.848. Отсюда следует, что $\chi^2_{\text{эксп}} < \chi^2_{\text{таб}}$ и можно сделать вывод о том, что гипотезы о равновероятном распределении в генераторах случайных чисел принимается. Если бы $\chi^2_{\text{эксп}}$ значение попало в критическую область (была бы ровна или больше чем $\chi^2_{\text{таб}}$), то гипотеза была бы отклонена. Следовательно, для всех трех различных генераторов гипотеза о равномерном распределении принимается.

В каждом из наших проведенных экспериментов автокорреляционная функция при изменении параметра τ (смещение в последовательности от 1 до половины наших интервалов) приближена к нулю, что говорит нам об очень слабой силе корреляции. Она показывает зависимости между данными крайне мала и также можно отметить, что числа генерируются случайным образом.

Листинг программы

Использование стандартной функции `rand`.

RND_RAND.c

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <time.h>
    #include <math.h>
    #include <limits.h>
    float get_float_rand(float l,float r){
    int l_ = l*1000*1000;
    int r_ = r*1000*1000;
    return (float) (rand()%(r_-l_)+l_)/1000000;
    }
    int main(){
    long int N,n;
    int k;
    FILE *Data,*Data1,*Data2;
    Data = fopen("1000000_1000_3.dat", "w"); Data1 = fopen("intervals_1000_3.dat", "w");
    while(1){
    printf("Enter the number of points to generate : N = ");
    scanf("%d",&N);
    printf("How many intervals to divide a segment from 0 to 1, provided that the number of
points must be 5 once more than the number of intervals : k = ");
    scanf("%d",&k);
    if(N/k<5){
    printf("Error!!! N/k<5\n");
    }else{
    break;
    }
    }
    int kk[k];
    float rr[N];
    for (int i = 0; i < k;i++)
    kk[i] = 0;
    for (int i = 0; i < N;i++)
    rr[i] = 0;
    float g = 1.0 / (k * 1.0);
    srand(time(NULL));
    for (long int i = 0; i < N; i++){
    float r = get_float_rand(0.0,1.0);
    rr[i] = r;
    fprintf(Data,"%f\n",r);
    int inter = (int)(r/g);
    kk[inter]++;
    }
    for (int i = 0; i < k;i++)
    fprintf(Data1,"%d\n",kk[i]);
    fclose(Data);
    fclose(Data1);
    //////
```

```
float chi2 = 0.0;
for (int i = 0; i < k;i++){
chi2 += pow(kk[i] - (N/k),2) /(N/k);
}
printf("X^2=%f\n",chi2);
Data2 = fopen("autocorr.dat", "w");
for(int offset = 1;offset <= k/2;offset++){
float ExKvX = 0.0;float matX = 0.0;float dis = 0.0;float ExKvY = 0.0;float matY = 0.0;
for (int i = 0; i < k-offset;i++){
matX = matX + kk[i];
ExKvX = ExKvX + kk[i] * kk[i];
}
matX = matX / (k-offset);
ExKvX = (ExKvX / (k-offset))-(matX*matX);
for (int i = offset; i < k;i++){
matY = matY + kk[i];
ExKvY = ExKvY + kk[i] * kk[i];
}
matY = matY / (k-offset);
ExKvY = (ExKvY / (k-offset))-(matY*matY);
for (int i = 0; i < k-offset;i++)
dis = dis + (kk[i]*kk[i+offset]);
dis=dis/ (k-offset);
float autoR = 0.0;
autoR=(dis-(matX*matY))/(sqrt(ExKvX)*sqrt(ExKvY));

printf("R=%f,offset = %d\n",autoR,offset);
fprintf(Data2,"%f\n",autoR);
}
fclose(Data2);
return 0;
}
```

Алгоритм «Mersenna twister».

RND_ Mersenna_twister.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <limits.h>
#include "gen.c"

int main(){
long int N,n;
int k;
FILE *Data,*Data1,*Data2;
Data = fopen("1000000_1000_3.dat", "w"); Data1 = fopen("intervals_1000_3.dat", "w");
while(1){
printf("Enter the number of points to generate : N = ");
scanf("%d",&N);
```

```
        printf("How many intervals to divide a segment from 0 to 1, provided that the number of
points must be 5 once more than the number of intervals : k = ");
        scanf("%d",&k);
        if(N/k<5){
        printf("Error!!! N/k<5\n");
        }else{
        break;
        }
        }
        int kk[k];
        float rr[N];
        for (int i = 0; i < k;i++)
        kk[i] = 0;
        for (int i = 0; i < N;i++)
        rr[i] = 0;
        float g = 1.0 / (k * 1.0);
        srand(time(NULL));
        for (long int i = 0; i < N; i++){
        float r=genrand64_real1();
        rr[i] = r;
        fprintf(Data,"%f\n",r);
        int inter = (int)(r/g);
        kk[inter]++;
        }
        for (int i = 0; i < k;i++)
        fprintf(Data1,"%d\n",kk[i]);
        fclose(Data);
        fclose(Data1);
        float chi2 = 0.0;
        for (int i = 0; i < k;i++){
        chi2 += pow(kk[i] - (N/k),2) /(N/k);
        }
        printf("X^2=%f\n",chi2);
        Data2 = fopen("autocorr.dat", "w");
        for(int offset = 1;offset <= k/2;offset++){
        float ExKvX = 0.0;float matX = 0.0;float dis = 0.0;float ExKvY = 0.0;float matY = 0.0;
        for (int i = 0; i < k-offset;i++){
        matX = matX + kk[i];
        ExKvX = ExKvX + kk[i] * kk[i];
        }
        matX = matX / (k-offset);
        ExKvX = (ExKvX / (k-offset))-(matX*matX);
        for (int i = offset; i < k;i++){
        matY = matY + kk[i];
        ExKvY = ExKvY + kk[i] * kk[i];
}
matY = matY / (k-offset);
ExKvY = (ExKvY / (k-offset))-(matY*matY);
for (int i = 0; i < k-offset;i++)
dis = dis + (kk[i]*kk[i+offset]);
dis=dis/ (k-offset);
float autoR = 0.0;
autoR=(dis-(matX*matY))/(sqrt(ExKvX)*sqrt(ExKvY));
```

```
printf("R=%f,offset = %d\n",autoR,offset);
fprintf(Data2,"%f\n",autoR);
}
fclose(Data2);
return 0;
}
```

gen.c

```
#include <stdio.h>
#define NN 312
#define MM 156
#define MATRIX_A 0xB5026F5AA96619E9ULL
#define UM 0xFFFFFFFF80000000ULL /* Most significant 33 bits */
#define LM 0x7FFFFFFFULL /* Least significant 31 bits */


/* The array for the state vector */
static unsigned long long mt[NN];
/* mti==NN+1 means mt[NN] is not initialized */
static int mti = NN + 1;

/* initializes mt[NN] with a seed */
void init_genrand64(unsigned long long seed)
{
mt[0] = seed;
for (mti = 1; mti < NN; mti++)
mt[mti] = (6364136223846793005ULL * (mt[mti - 1] ^ (mt[mti - 1] >> 62)) + mti);
}

/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
void init_by_array64(unsigned long long init_key[],
unsigned long long key_length)
{
unsigned long long i, j, k;
init_genrand64(19650218ULL);
i = 1; j = 0;
k = (NN > key_length ? NN : key_length);
for (; k; k--)
{
mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 62)) * 3935559000370003845ULL))
+ init_key[j] + j; /* non linear */
i++; j++;
if (i >= NN) { mt[0] = mt[NN - 1]; i = 1; }
if (j >= key_length) j = 0;
}
for (k = NN - 1; k; k--) {
mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 62)) * 2862933555777941757ULL))
- i; /* non linear */
i++;
if (i >= NN) { mt[0] = mt[NN - 1]; i = 1; }
```

```
}

mt[0] = 1ULL << 63; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0, 2^64-1]-interval */
unsigned long long genrand64_int64(void)
{
int i;
unsigned long long x;
static unsigned long long mag01[2] = { 0ULL, MATRIX_A };

if (mti >= NN)
{ /* generate NN words at one time */

/* if init_genrand64() has not been called, */
/* a default initial seed is used     */
if (mti == NN + 1)
init_genrand64(5489ULL);

for (i = 0; i < NN - MM; i++)
{
x = (mt[i] & UM) | (mt[i + 1] & LM);
mt[i] = mt[i + MM] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];
}
for (; i < NN - 1; i++) {
x = (mt[i] & UM) | (mt[i + 1] & LM);
mt[i] = mt[i + (MM - NN)] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];
}
x = (mt[NN - 1] & UM) | (mt[0] & LM);
mt[NN - 1] = mt[MM - 1] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];

mti = 0;
}

x = mt[mti++];

x ^= (x >> 29) & 0x5555555555555555ULL;
x ^= (x << 17) & 0x71D67FFFEDA60000ULL;
x ^= (x << 37) & 0xFFF7EEE000000000ULL;
x ^= (x >> 43);

return x;
}

/* generates a random number on [0, 2^63-1]-interval */
long long genrand64_int63(void)
{
return (long long)(genrand64_int64() >> 1);
}

/* generates a random number on [0,1]-real-interval */
double genrand64_real1(void)
```

```
{
return (genrand64_int64() >> 11)* (1.0 / 9007199254740991.0);
}

/* generates a random number on [0,1)-real-interval */
double genrand64_real2(void)
{
return (genrand64_int64() >> 11)* (1.0 / 9007199254740992.0);
}

/* generates a random number on (0,1)-real-interval */
double genrand64_real3(void)
{
return ((genrand64_int64() >> 12) + 0.5)* (1.0 / 4503599627370496.0);
}
```

Цифровой генератор случайных чисел Intel.

RND_DRND.c

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <limits.h>



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(){

        long int N,n;

        int k;

        FILE *Data,*Data1,*Data2;

        Data = fopen("1000000_1000_3.dat", "w"); Data1 = fopen("intervals_1000_3.dat", "w");

        while(1){

        printf("Enter the number of points to generate : N = ");

        scanf("%d",&N);

        printf("How many intervals to divide a segment from 0 to 1, provided that the number of
points must be 5 once more than the number of intervals : k = ");

        scanf("%d",&k);

        if(N/k<5){

        printf("Error!!! N/k<5\n");

        }else{

        break;

        }

        }

        int kk[k];

        float rr[N];

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

        kk[i] = 0;

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

        rr[i] = 0;

        float g = 1.0 / (k * 1.0);

        srand(time(NULL));

        for (long int i = 0; i < N; i++){

        float r;

        randoms(&r,0.0, 1.0);

        rr[i] = r;
```

```
fprintf(Data,"%f\n",r);

int inter = (int)(r/g);

kk[inter]++;

}

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

fprintf(Data1,"%d\n",kk[i]);

fclose(Data);

fclose(Data1);

float chi2 = 0.0;

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

chi2 += pow(kk[i] - (N/k),2) /(N/k);

}

printf("X^2=%f\n",chi2);

Data2 = fopen("autocorr.dat", "w");

for(int offset = 1;offset <= k/2;offset++){

float ExKvX = 0.0;float matX = 0.0;float dis = 0.0;float ExKvY = 0.0;float matY = 0.0;

for (int i = 0; i < k-offset;i++){

matX = matX + kk[i];

ExKvX = ExKvX + kk[i] * kk[i];

}

matX = matX / (k-offset);

ExKvX = (ExKvX / (k-offset))-(matX*matX);

for (int i = offset; i < k;i++){

matY = matY + kk[i];

ExKvY = ExKvY + kk[i] * kk[i];

}

matY = matY / (k-offset);

ExKvY = (ExKvY / (k-offset))-(matY*matY);

for (int i = 0; i < k-offset;i++)

dis = dis + (kk[i]*kk[i+offset]);
```

```
        dis=dis/ (k-offset);

        float autoR = 0.0;

        autoR=(dis-(matX*matY))/(sqrt(ExKvX)*sqrt(ExKvY));

        printf("R=%f,offset = %d\n",autoR,offset);

        fprintf(Data2,"%f\n",autoR);

        }

        fclose(Data2);

        return 0;

        }
```