ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ

Кафедра ВС

Отчет

по лабораторной работе №1

Выполнил:

Студент группы МГ-172

Суходоева Н. Н.

Проверил:

д.т.н., профессор

Родионов А.С.

Новосибирск 2018

Постановка задачи

Даны два датчика случайных чисел. Для первого первый элемент равен «e», второй «pi», следующее значение считается как сумма предыдущего и предпредыдущего по модулю 4. Для второго датчика первый элемент подбираем сами, следующее значение равно квадрату числа состоящего из битов, которые взяли из середины предыдущего.

Нужно определить какой из датчиков лучше с помощью хи-квадрат и автоковариации.

Ход решения

Для каждого датчика генерируем миллион значений. Рассчитываем хи-квадрат и автоковариацию, также фиксируем попадания генерируемых датчиками значений в заранее установленные промежутки, чтобы наглядно оценить распределение генерируемых значений.

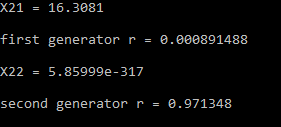


Рисунок 1. Пример работы программы

«X21» и «X22» - это хи-квадрат для первого и второго датчиков соответственно, «r» - значение автоковариации.

Рисунок 2. Гистограмма распределения для первого датчика

Рисунок 3. Гистограмма распределения для второго датчика

Вывод

Первый датчик лучше второго, но при этом оба плохие. Значение хи-квадрата, рассчитанное для первого датчика показывает, что генерируемые значения распределяются близко к равномерному распределению. Для второго хи-квадрат получился таким, так как при данном методе генерации величин, получаемые значения становятся равными нулю почти сразу же. Значение ковариации для первого датчика мало, так как зависимость следующего числа идет не от предыдущего, а двух предыдущих. Для второго же велико, так как полностью зависит от предыдущего числа.

Листинг программы

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <bitset>

#include <iostream>

using namespace std;

#define N 1000000

#define n 20

main()

{

int i, j, k;

long double XY, X, Y, Sx, Sy, r\_first, r\_second, X21, X22, H[n], A;

//первый генератор

X = 0; Y = 0; Sx = 0; Sy = 0; XY = 0;

double \*first\_gen = new double[N];

double \*first\_gen\_f = new double[N];

first\_gen[0] = 2.718282; //e

first\_gen[1] = 3.141593; //pi точки отсчета

first\_gen\_f[0] = first\_gen[1];

for( i = 0; i < n; i++ ) H[i] = 0;

for( i = 2; i < N; i++ ){

first\_gen[i] = ( first\_gen[i-1] + first\_gen[i-2] );

while( first\_gen[i] >= 4 ){

first\_gen[i] = first\_gen[i] - 4;

}

first\_gen\_f[i-1] = first\_gen[i];

A = first\_gen[i];

if( A <= 1 ){

k = 1;

}

else{

if( A <= 2 ){

A = A - 1;

k = 6;

}

else{

if( A <= 3 ){

A = A - 2;

k = 11;

}

else{

if( A <= 4 ){

A = A - 3;

k = 16;

}

}

}

}

if( A <= 0.2 ) k = k - 1;

else{

if( A <= 0.4 ) k = k;

else{

if( A <= 0.6 ) k = k + 1;

else{

if( A <= 0.8 ) k = k + 2;

else{

if( A <= 1 ) k = k + 3;

}

}

}

}

H[k] = H[k] + 1;

}

for( i = 0; i < N - 1; i++ ){

X = X + first\_gen[i];

Y = Y + first\_gen\_f[i];

XY = XY + first\_gen[i] \* first\_gen\_f[i];

Sx = Sx + first\_gen[i] \* first\_gen[i];

Sy = Sy + first\_gen\_f[i] \* first\_gen\_f[i];

}

X = X / ( N - 1 );

Y = Y / ( N - 1 );

XY = XY / ( N - 1 );

Sx = Sx / ( N - 1 ) - X\*X;

Sy = Sy / ( N - 1 ) - Y\*Y;

r\_first = ( XY - X \* Y ) / ( sqrt(Sx) \* sqrt(Sy) );

X21 = 0;

for( i = 0; i < n; i++ ) X21 = X21 + ( H[i] - N / n ) \* ( H[i] - N / n );

X21 = X21 / ( N / n );

cout<<"X21 = "<<X21<<'\n';

for( i = 0; i < n; i++ ) cout<<"H["<<i<<"] = "<<H[i]<<" || ";

cout<<'\n';

cout<<"first generator r = "<<r\_first<<'\n';

//второй генератор

X = 0; Y = 0; Sx = 0; Sy = 0; XY = 0;

for( i = 0; i < n; i++ ) H[i] = 0;

unsigned long \*second\_gen = new unsigned long[N];

unsigned long \*second\_gen\_f = new unsigned long[N];

bitset<sizeof(unsigned long)\*8> bitvec2;

second\_gen[0] = 42949;

for( i = 1; i < N; i++ ){

bitset<sizeof(unsigned long)\*8> bitvec1(second\_gen[i-1]);

for( j = 8; j < 24; j++ ){

bitvec2[j-8] = bitvec1[j];

}

second\_gen\_f[i-1] = second\_gen[i] = bitvec2.to\_ulong() \* bitvec2.to\_ulong();

bitvec1.reset();

bitvec2.reset();

A = second\_gen[i];

while( A >= 4 ){

A = A - 4;

}

if( A <= 1 ){

k = 1;

}

else{

if( A <= 2 ){

A = A - 1;

k = 6;

}

else{

if( A <= 3 ){

A = A - 2;

k = 11;

}

else{

if( A <= 4 ){

A = A - 3;

k = 16;

}

}

}

}

if( A <= 0.2 ) k = k - 1;

else{

if( A <= 0.4 ) k = k;

else{

if( A <= 0.6 ) k = k + 1;

else{

if( A <= 0.8 ) k = k + 2;

else{

if( A <= 1 ) k = k + 3;

}

}

}

}

H[k] = H[k] + 1;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

for( i = 0; i < N - 1; i++ ){

X = X + second\_gen[i];

Y = Y + second\_gen\_f[i];

XY = XY + second\_gen[i] \* second\_gen\_f[i];

Sx = Sx + second\_gen[i] \* second\_gen[i];

Sy = Sy + second\_gen\_f[i] \* second\_gen\_f[i];

//cout<<second\_gen[i]<<"||"<<'\n';

}

X = X / ( N - 1 );

Y = Y / ( N - 1 );

XY = XY / ( N - 1 );

Sx = Sx / ( N - 1 ) - X\*X;

Sy = Sy / ( N - 1 ) - Y\*Y;

r\_second = ( XY - X \* Y ) / ( sqrt(Sx) \* sqrt(Sy) );

X22 = 0;

for( i = 0; i < n; i++ ) X22 = X22 + ( H[i] - N / n ) \* ( H[i] - N / n );

X22 = X22 / ( N / n );

printf("\nX22 = %g\n",X22);

for( i = 0; i < n; i++ ) cout<<"H["<<i<<"] = "<<H[i]<<" || ";

cout<<'\n';

cout<<"second generator r = "<<r\_second<<'\n';

}