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

Кафедра вычислительных систем

Пояснительная записка

к нулевой 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

Убедиться в равномерности распределения генератора псевдослучайных чисел, используя параметр χ^2 и автокорреляция.

2 Теоретические сведения

1) Используемые генераторы

1. Линейно конгруэнтный метод (linear congruential) – применяется в простых случаях и не обладает криптографической стойкостью. Суть метода заключается в вычислении последовательности случайных чисел X_n, полагая:

   $X_{n+1} = (aX_n + c)\ mod\ m$,

   где m – модуль($m \geq 2$), a – множитель($0 \leq a < m$), c – приращение($0 \leq c \leq m$), X_0 – начальное значение($0 \leq X_0 < m$).

2. Вихрь мерсенна (mersenne twister) – генератор псевдослучайных чисел Вихрь Мерсенна основывается на свойствах простых чисел Мерсенна и обеспечивает быструю генерацию высококачественных по критерию случайности псевдослучайных чисел.

3. Вычет с переносом (subtract with carry) – алгоритм, используемый этим механизмом, является генератором с запаздыванием Фибоначчи, с последовательностью состояний из r целых элементов плюс одно значение переноса.

2) Проверка по критерию хи-квадрат

p_i – теоретическая вероятность попадания чисел в i-ый интервал (всего этих интервалов k) равна $p_i = 1/k$

N – общее количество сгенерированных чисел

n_i – попадание чисел в каждый интервал

χ^2 – критерий, который позволяет определить, удовлетворяет ли генератор случайных чисел требованиям равномерного распределения или нет

Процедура проверки имеет следующий вид:

1. Диапазон от 0 до 1 разбивается на k равных интервалов
2. Запускается генератор случайных чисел N раз (N должно быть велико, например, $N/k > 5$)
3. Определяется количество случайных чисел, попавших в каждый интервал
4. Вычисляется экспериментальное значение χ^2 по следующей формуле:

$$\chi^2_{\text{эксп.}} = \frac{(n_1 - p_1 * N)^2}{p_1 * N} + \frac{(n_2 - p_2 * N)^2}{p_2 * N} + \cdots + \frac{(n_k - p_k * N)^2}{p_2 * N}$$

$$\chi^2_{\text{эксп.}} = \sum_{i=1}^{k} \frac{(n_i - p_i * N)^2}{p_i * N} = \frac{1}{N} \sum_{i=1}^{k} \left(\frac{n_i^2}{p_i}\right) - N$$

3) Проверка по критерию автокорреляции

$Ex = \frac{1}{N} \sum_{i=1}^{N} x_i = \bar{x}$ – математическое ожидание

$S^2 = Ex^2 - (Ex)^2$ – дисперсия

$\hat{a}(\tau) = \frac{1}{(N-\tau)*S^2} \sum_{i=1}^{N-\tau} (x_i - E_x)(x_{i+\tau} - E_x)$ – автокорреляция

3 Результат работы программы

1) Хи-квадрат

```
Generating N random numbers
N = 1000000
k = 20
Engine: linear congruential

[0 ; 0.05) 50370
[0.05 ; 0.1) 50211
[0.1 ; 0.15) 50298
[0.15 ; 0.2) 50103
[0.2 ; 0.25) 50044
[0.25 ; 0.3) 49908
[0.3 ; 0.35) 49783
[0.35 ; 0.4) 49907
[0.4 ; 0.45) 50178
[0.45 ; 0.5) 49276
[0.5 ; 0.55) 50032
[0.55 ; 0.6) 49647
[0.6 ; 0.65) 50083
[0.65 ; 0.7) 50138
[0.7 ; 0.75) 50241
[0.75 ; 0.8) 50336
[0.8 ; 0.85) 49806
[0.85 ; 0.9) 50000
[0.9 ; 0.95) 49913
[0.95 ; 1) 49726

linear congruential: chi square = 27
```

```
Generating N random numbers
N = 1000000
k = 20
Engine: mersenne twister

[0 ; 0.05) 49627
[0.05 ; 0.1) 49904
[0.1 ; 0.15) 50115
[0.15 ; 0.2) 49944
[0.2 ; 0.25) 50132
[0.25 ; 0.3) 50092
[0.3 ; 0.35) 49613
[0.35 ; 0.4) 49956
[0.4 ; 0.45) 49914
[0.45 ; 0.5) 50353
[0.5 ; 0.55) 50097
[0.55 ; 0.6) 49910
[0.6 ; 0.65) 49913
[0.65 ; 0.7) 50296
[0.7 ; 0.75) 50028
[0.75 ; 0.8) 50042
[0.8 ; 0.85) 50099
[0.85 ; 0.9) 50134
[0.9 ; 0.95) 49797
[0.95 ; 1) 50034

mersenne twister: chi square = 13.1875
```

```
Generating N random numbers
N = 1000000
k = 20
Engine: subtract with carry

[0 ; 0.05) 50108
[0.05 ; 0.1) 50421
[0.1 ; 0.15) 49870
[0.15 ; 0.2) 49972
[0.2 ; 0.25) 50266
[0.25 ; 0.3) 50053
[0.3 ; 0.35) 50128
[0.35 ; 0.4) 49673
[0.4 ; 0.45) 49607
[0.45 ; 0.5) 50061
[0.5 ; 0.55) 50368
[0.55 ; 0.6) 50176
[0.6 ; 0.65) 49632
[0.65 ; 0.7) 49621
[0.7 ; 0.75) 50224
[0.75 ; 0.8) 49960
[0.8 ; 0.85) 50103
[0.85 ; 0.9) 50073
[0.9 ; 0.95) 49903
[0.95 ; 1) 49781

subtract with carry: chi square = 22.625
```

2) Автокорреляция

```
Generating N random numbers
N = 10^5
k = 20
Engine: linear congruential

offset = 5 N = 100000 autocorrelation = -0.0026124
offset = 15 N = 100000 autocorrelation = -0.0075719
offset = 25 N = 100000 autocorrelation = -0.0026861
offset = 35 N = 100000 autocorrelation = 0.0019608
offset = 45 N = 100000 autocorrelation = 0.0018654
offset = 55 N = 100000 autocorrelation = 0.0020958
offset = 65 N = 100000 autocorrelation = 0.0026353
offset = 75 N = 100000 autocorrelation = 0.0020301
offset = 85 N = 100000 autocorrelation = -0.0003449
offset = 95 N = 100000 autocorrelation = 0.0049483

linear congruential: autocorrelation = 0.000232035

Generating N random numbers
N = 10^5
k = 20
Engine: mersenne twister

offset = 5 N = 100000 autocorrelation = 0.0054556
offset = 15 N = 100000 autocorrelation = -0.0015640
offset = 25 N = 100000 autocorrelation = 0.0007044
offset = 35 N = 100000 autocorrelation = -0.0028602
offset = 45 N = 100000 autocorrelation = -0.0060097
offset = 55 N = 100000 autocorrelation = -0.0018165
offset = 65 N = 100000 autocorrelation = 0.0011541
offset = 75 N = 100000 autocorrelation = -0.0003107
offset = 85 N = 100000 autocorrelation = -0.0015665
offset = 95 N = 100000 autocorrelation = -0.0030891

mersenne twister: autocorrelation = -0.000990279

Generating N random numbers
N = 10^5
k = 20
Engine: subtract with carry

offset = 5 N = 100000 autocorrelation = 0.0016116
offset = 15 N = 100000 autocorrelation = -0.0049648
offset = 25 N = 100000 autocorrelation = 0.0037484
offset = 35 N = 100000 autocorrelation = 0.0019848
offset = 45 N = 100000 autocorrelation = -0.0013372
offset = 55 N = 100000 autocorrelation = -0.0032693
offset = 65 N = 100000 autocorrelation = 0.0017427
offset = 75 N = 100000 autocorrelation = 0.0052082
offset = 85 N = 100000 autocorrelation = 0.0017452
offset = 95 N = 100000 autocorrelation = -0.0017988

subtract with carry: autocorrelation = 0.000467082
```

```
Generating N random numbers
N = 10^6
k = 20
Engine: linear congruential

offset = 5 N = 1000000 autocorrelation = -0.0003168
offset = 15 N = 1000000 autocorrelation = -0.0014499
offset = 25 N = 1000000 autocorrelation = 0.0012953
offset = 35 N = 1000000 autocorrelation = -0.0004175
offset = 45 N = 1000000 autocorrelation = -0.0007809
offset = 55 N = 1000000 autocorrelation = 0.0013563
offset = 65 N = 1000000 autocorrelation = -0.0000014
offset = 75 N = 1000000 autocorrelation = -0.0008938
offset = 85 N = 1000000 autocorrelation = -0.0002374
offset = 95 N = 1000000 autocorrelation = 0.0010006

linear congruential: autocorrelation = -4.45558e-05

Generating N random numbers
N = 10^6
k = 20
Engine: mersenne twister

offset = 5 N = 1000000 autocorrelation = 0.0011547
offset = 15 N = 1000000 autocorrelation = -0.0002945
offset = 25 N = 1000000 autocorrelation = 0.0003912
offset = 35 N = 1000000 autocorrelation = -0.0001278
offset = 45 N = 1000000 autocorrelation = 0.0015043
offset = 55 N = 1000000 autocorrelation = 0.0015213
offset = 65 N = 1000000 autocorrelation = 0.0008265
offset = 75 N = 1000000 autocorrelation = -0.0002299
offset = 85 N = 1000000 autocorrelation = -0.0003800
offset = 95 N = 1000000 autocorrelation = 0.0000286

mersenne twister: autocorrelation = 0.000439436

Generating N random numbers
N = 10^6
k = 20
Engine: subtract with carry

offset = 5 N = 1000000 autocorrelation = 0.0001960
offset = 15 N = 1000000 autocorrelation = -0.0027491
offset = 25 N = 1000000 autocorrelation = 0.0008000
offset = 35 N = 1000000 autocorrelation = 0.0007384
offset = 45 N = 1000000 autocorrelation = -0.0016334
offset = 55 N = 1000000 autocorrelation = 0.0009024
offset = 65 N = 1000000 autocorrelation = -0.0009008
offset = 75 N = 1000000 autocorrelation = 0.0003683
offset = 85 N = 1000000 autocorrelation = -0.0003189
offset = 95 N = 1000000 autocorrelation = -0.0010997

subtract with carry: autocorrelation = -0.000369678
```

```
Generating N random numbers
N = 10^5
k = 10
Engine: linear congruential

offset = 5 N = 100000 autocorrelation = -0.0006668
offset = 15 N = 100000 autocorrelation = 0.0011342
offset = 25 N = 100000 autocorrelation = 0.0018984
offset = 35 N = 100000 autocorrelation = -0.0013201
offset = 45 N = 100000 autocorrelation = 0.0011650
offset = 55 N = 100000 autocorrelation = 0.0027475
offset = 65 N = 100000 autocorrelation = 0.0029161
offset = 75 N = 100000 autocorrelation = -0.0019993
offset = 85 N = 100000 autocorrelation = -0.0075088
offset = 95 N = 100000 autocorrelation = 0.0045416

linear congruential: autocorrelation = 0.000290783

Generating N random numbers
N = 10^5
k = 10
Engine: mersenne twister

offset = 5 N = 100000 autocorrelation = -0.0037469
offset = 15 N = 100000 autocorrelation = -0.0042180
offset = 25 N = 100000 autocorrelation = 0.0028028
offset = 35 N = 100000 autocorrelation = -0.0093170
offset = 45 N = 100000 autocorrelation = -0.0008193
offset = 55 N = 100000 autocorrelation = 0.0026514
offset = 65 N = 100000 autocorrelation = 0.0010486
offset = 75 N = 100000 autocorrelation = -0.0050978
offset = 85 N = 100000 autocorrelation = 0.0002210
offset = 95 N = 100000 autocorrelation = 0.0043858

mersenne twister: autocorrelation = -0.00120894

Generating N random numbers
N = 10^5
k = 10
Engine: subtract with carry

offset = 5 N = 100000 autocorrelation = -0.0060109
offset = 15 N = 100000 autocorrelation = -0.0034510
offset = 25 N = 100000 autocorrelation = 0.0008922
offset = 35 N = 100000 autocorrelation = 0.0028921
offset = 45 N = 100000 autocorrelation = 0.0030404
offset = 55 N = 100000 autocorrelation = 0.0017016
offset = 65 N = 100000 autocorrelation = -0.0002727
offset = 75 N = 100000 autocorrelation = -0.0037249
offset = 85 N = 100000 autocorrelation = -0.0019344
offset = 95 N = 100000 autocorrelation = 0.0036914

subtract with carry: autocorrelation = -0.00031762
```

4 Выводы

На основе полученных данных χ^2 можно сказать, что, при степени своды 17 и уровне значимости $\alpha = 0.05$, наши экспериментальные значения близки к равномерному. Так как:

$k = m - r - 1$ — степень свободы

m — количество интервалов (20 интервалов)

r — число параметров предполагаемого распределения (2 параметра у равномерного распределения)

$$k = 20 - 2 - 1 = 17$$

$$\alpha = 0.05$$

$$\chi^2_{\text{теор.}} = 27.6$$

$\chi^2_{\text{эксп1}} = 27$ – линейно конгруэнтный метод

$\chi^2_{\text{эксп2}} = 13.1875$ – вихрь мерсенна

$\chi^2_{\text{эксп3}} = 22.625$ – вычесть с переносом

Ближе всего к равномерному это результаты полученный с помощью генератора вихря мерсенна, так как отклонение экспериментальных значений от теоритических больше чем при других запусках. Нулевая гипотеза подтверждена для всех ГСЧ.

Отрицательная корреляция – увеличении одной переменной связано уменьшение другой. Положительная корреляция – увеличение одной переменной связано увеличением другой. Прямой зависимости автокорреляции от смещения нету. Смещение помогает выявить зависимость между значениями. Так как все значения автокорреляции колеблются около нуля, значит, что все генераторы выдают независимые случайные величины.

5 Приложение

Список литературы

1) Отрицательная автокорреляция // Большая энциклопедия нефти и газа. [Электронный ресурс]. URL: https://www.ngpedia.ru/id181103p1.html (Дата обращения 17.03.2020).
2) Критерий хи-квадрат // Распознование. [Электронный ресурс]. URL: http://www.machinelearning.ru/wiki/index.php?title=%D0%9A%D1%80%D0%B8%D1%82%D0%B5%D1%80%D0%B8%D0%B9_%D1%85%D0%B8-%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82 (Дата обращения 17.03.2020).
3) Автокорреляция, Коэффициент автокорреляции. [Электронный ресурс]. URL: http://univer-nn.ru/ekonometrika/avtokorrelyaciya-koefficient-avtokorrelyacii/ (Дата обращения 17.03.2020).

Исходный код

main.cpp

```
#include <iostream>
#include <random>
#include <vector>
#include <fstream>
#include <string.h>
#include <chrono>

using namespace std;

float calc_chi_square(int N, int k, vector<int> n);
float calcAutocorrelation2(vector<float> rand_namber);

vector<float> gen_list_rand_numbers(int N, double
(*randomNumberGenerator)(double, double));
vector<int> check_hits_interval(int k, vector<float>
rand_numbers);

double getRandCongruential(double a, double b);
double getRandMersenne(double a, double b);
double getRandSubtract(double a, double b);

void printVector(vector<int> vec, int k, int N);

void
printVector(vector<int> vec, int k, int N)
{
  float step = 1.0 / k;
```

```
  for (int i = 0; i < k; i++)
    cout << "[" << step * i  << " ; "
         << step * i + step << ")"
         << " " << vec[i] << endl;
}

float
taskChiSquare(char *prefix_name,
           double (*randomNumberGenerator)(double, double))
{
  int N = 1000000, k = 20;
  char filename[64];
  vector<float> rand_numbers;
  vector<int> n;
  float chi_square;

  rand_numbers = gen_list_rand_numbers(N,
                                       randomNumberGenerator);
  n = check_hits_interval(k, rand_numbers);
  chi_square = calc_chi_square(N, k, n);

  cout << "Generating N random numbers" << endl;
  cout << "N = " << N << endl;
  cout << "k = " << k << endl;
  cout << "Engine: " << prefix_name << endl << endl;

  printVector(n, k, N);

  cout << endl <<  prefix_name << ": chi square = "
       << chi_square << endl << endl;

  return chi_square;
}

float
taskAutocorr(char *prefix_name, int N, int k,
          double (*randomNumberGenerator)(double, double))
{
  char filename[64];
  vector<float> rand_numbers;
  vector<int> n;
  float autocor;

  rand_numbers = gen_list_rand_numbers(N,
                                       randomNumberGenerator);
  n = check_hits_interval(k, rand_numbers);

  cout << "Generating N random numbers" << endl;
  cout << "N = " << N << endl;
  cout << "k = " << k << endl;
  cout << "Engine: " << prefix_name << endl << endl;

  autocor = calcAutocorrelation2(rand_numbers);
```

```
  cout << endl <<  prefix_name << ": autocorrelation = "
       << autocor << endl << endl;

  return autocor;
}

void
taskChiSquare2()
{
  taskChiSquare("linear congruential",getRandCongruential);
  taskChiSquare("mersenne twister", getRandMersenne);
  taskChiSquare("subtract with carry", getRandSubtract);
}

void
taskAutocorr2()
{
  int N1 = 100000, N2 = 1000000, k1 = 20, k2 = 10;

  taskAutocorr("linear congruential", N1, k1,
               getRandCongruential);
  taskAutocorr("mersenne twister", N1, k1,
               getRandMersenne);
  taskAutocorr("subtract with carry", N1, k1,
               getRandSubtract);

  taskAutocorr("linear congruential", N2, k1,
               getRandCongruential);
  taskAutocorr("mersenne twister", N2, k1,
               getRandMersenne);
  taskAutocorr("subtract with carry", N2, k1,
               getRandSubtract);

  taskAutocorr("linear congruential", N1, k2,
               getRandCongruential);
  taskAutocorr("mersenne twister", N1, k2,
               getRandMersenne);
  taskAutocorr("subtract with carry", N1, k2,
               getRandSubtract);
}

int
main(int argc, char **argv)
{
  taskChiSquare2();
  taskAutocorr2();
  return 0;
}

vector<float>
gen_list_rand_numbers(int N,
          double (*randomNumberGenerator)(double, double))
```

```
{
  vector<float> rand_nambers(N);
  for (int i = 0; i < N; i++)
    rand_nambers[i] = randomNumberGenerator(0.0, 1.0);
  return rand_nambers;
}

vector<int>
check_hits_interval(int k, vector<float> rand_numbers)
{
  vector<int> n(k);
  float step = 1.0 / k, p_i = step;
  for (int i = 0; i < rand_numbers.size(); i++) {
    for (int j = 1; j <= k; j++) {
      if (step * j > rand_numbers[i]) {
        n[j - 1]++;
        break;
      }
    }
  }
  return n;
}

float
calc_expected_value(vector<float> rand_number, int power)
{
  float exp_value = 0.0;
  for (auto number : rand_number)
    exp_value += pow(number, power);
  return exp_value / rand_number.size();
}

float
calc_dispersion(float exp_value_square, float exp_value)
{
  return exp_value_square - pow(exp_value, 2.0);
}

float
calcAutocorrelation2(vector<float> randNumber)
{
  float exp_value = 0.0, exp_value_square = 0.0;
  float dispersion;
  int N = randNumber.size();
  exp_value = calc_expected_value(randNumber, 1);
  exp_value_square = calc_expected_value(randNumber, 2);
  dispersion = calc_dispersion(exp_value_square,
                               exp_value);
  vector<double> R_av;
  for (int offset = 5; offset < 100; offset += 10) {
    double R = 0.0;
    for (uint64_t i = 0; i < N; i++) {
      R += (randNumber[i] - exp_value)
```

```
            * (randNumber[(i + offset) % N] - exp_value);
    }
    R /= (dispersion * (N - offset));
    R_av.push_back(R);
    cout << "offset = " << offset
         << " N = " << N;
    printf(" autocorrelation = %0.7f\n", R);
  }
  double av = 0.0;
  for (auto i: R_av)
    av += i;
  return av / R_av.size();
}

float
calc_chi_square(int N, int k, vector<int> n)
{
  float rand_num, sum = 0.0, p_i = 1.0 / k;
  for (int i = 0; i < k; i++)
    sum += pow(n[i], 2.0) / p_i;
  return (sum / N) - N;
}

double
getRandCongruential(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
getRandMersenne(double a, double b)
{
  long seed =
chrono::system_clock::now().time_since_epoch().count();
  mt19937_64 rand_generator(seed);
  uniform_real_distribution<double> distribution(a, b);
  return distribution(rand_generator);
}

double
getRandSubtract(double a, double b)
{
  long seed =
chrono::system_clock::now().time_since_epoch().count();
  subtract_with_carry_engine<unsigned, 24, 10, 24>
rand_generator(seed);
  uniform_real_distribution<double> distribution(a, b);
  return distribution(rand_generator);
}
```