МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Ермолаева Екатерина Александровна

Отчет по лабораторным работам по курсу “Имитационное и статистическое моделирование”

студента 2 курса 14 группы

Работа сдана 2021г.

зачтена _____ 2021 г.

Преподаватель
Лобач Сергей Викторович
ассистент кафедры ММАД

_____
(подпись преподавателя)

Минск 2021

Лабораторная работа 1.

Условие:

Используя метод Макларена-Марсальи построить датчик БСВ (1 датчик должен быть мультипликативный конгруэнтный, второй – на выбор). Исследовать точность построенной БСВ.

1) Осуществить моделирование n = 1000 реализаций БСВ с помощью мультипликативного конгруэнтного метода (МКМ) с параметрами a0, β, M = 231 .

2) Осуществить моделирование n = 1000 реализаций БСВ с помощью метода Макларена-Марсальи (один датчик должен быть мультипликативный конгруэнтный (п. 1), второй – на выбор).

K – объем вспомогательной таблицы.

3) Проверить точность моделирования обоих датчиков (п. 1 и п. 2) с помощью критерия согласия Колмогорова и χ2-критерия Пирсона с уровнем значимости ε = 0.05.

Теория:

Мультипликативный конгруэнтный метод:

Псевдослучайная последовательность $a_1,\ a_2,\ldots,\ a_n$ строится по следующим рекуррентным формулам:

$$a_t = a_t^*/M,\ \ a_t^* = \{\beta a_{t-1}^*\} mod\ M\ (t = 1, 2,\ldots),$$

где $\beta,\ M,\ a_t^*$ - параметры датчика: β- множитель ($\beta < M$), M – модуль, $a_t^* \in \{1,\ldots, M - 1\}$- стартовое значение (нечетное число).

В данной работе брались значения: $M = 2147483648$, $a_t^* = \beta = 262147$.

Метод Макларена-Марсальи:

Пусть $\{\beta_t\}, \{c_t\}$- псевдослучайные последовательности, порожденные независимо работающими датчиками; $\{a_t\}$ - результирующая псевдослучайная последовательность реализация БСВ;

V={V(0), V(1), …,V(K-1)} – вспомогательная таблица *K* чисел.

Процесс вычисления $\{a_t\}$ включает следующие этапы:

- первоначальное заполнение таблицы

$$V: V(i) = \beta_i,\ i = \overline{0, K - 1};$$

- случайный выбор из таблицы:

$$a_t = V(s),\ s = [c_t \cdot K];$$

-обновление табличных значений:

$$V(s) = b_{t+K},\ t = 0, 1, 2,\ldots\ ..$$

В данной работе в качестве $\{\beta_t\}$бралась последовательность (из 1000 элементов), полученная мультипликативным конгруэнтным методом,

описанным выше. В качестве $\{c_t\}$ бралась последовательности (из 1000) элементов, полученная аналогичным способом с тем же M и $a_0^* = \beta = 50653,\ K = 256.$

χ^2 - критерий согласия Пирсона:

Область возможных значений случайной величины разбивается на интервалы $[x_{k-1},\ x_k),\ k = \overline{1,K}.$

Рассматривается следующая статистика:

$$\chi^2 = \sum_{k=1}^{K} \frac{(v_k - n \cdot p_k)}{n \cdot p_k}$$

где n – объем выборки,

v_k - количество элементов выборки, попавших в k-ый интервал,

p_k - вероятность попадания случайной величины в k-ый интервал.

Проверяется условие $\chi^2 < \Delta$, где $\Delta = G^{-1}(1 - \varepsilon)$, G - функция распределения распределения χ^2, ε - уровень значимости (обычно $\varepsilon = 0.05$).

В данной работе отрезок [0;1] разбивался на 10 интервалов.

Критерий согласия Колмогорова:

Рассматривается статистика:

$$D_n = \sup_{x \in \mathbb{R}} | F_\xi(x) - F_0(x) | \in [0;1],$$

где

$$F_\zeta(x) = \frac{1}{n} \sum_{i=1}^{n} I_{[-\infty;x]}(x_i),\ x \in R,$$

$$F_0(x) = x,\ x \in [0;1]$$

Проверяется условие $\sqrt{n} K_n < \Delta$, где $\Delta = K^{-1}(1 - \varepsilon)$, K - функция распределения распределения Колмогорова, ε - уровень значимости.

Код программы:

```
#include <iostream>
#include <algorithm>
using namespace std;
long long M = 2147483648;
long long a0 = 262147;
long long a1 = 50653;
long long K = 256;

void MKM(long double* sequence, int n, long long a, long long b) {
   long long aPrev = a;
   for (int i = 0; i < n; i++) {
         sequence[i] = (long double)aPrev / M;
         aPrev = (aPrev * b) % M;
   }
}

void MM(long double* sequence, int n) {
   long double* bSequence = new long double[1000];
```

```
    long double* cSequence = new long double[1000];
    MKM(bSequence, 1000, a0, a0);
    MKM(cSequence, 1000, a1, a1);

    long double* V = new long double[K];

    for (int i = 0; i < K; i++) {
          V[i] = bSequence[i];
    }
    for (int i = 0; i < n; i++) {
          int s = (int)(cSequence[i] * K);
          sequence[i] = V[s];
          V[s] = bSequence[i + K];
    }

    delete[] bSequence, cSequence, V;
}

void Pirson(long double* sequence, int n) {
    double v[10] = { 0 };
    double x = 0;
    for (int i = 0; i < n; i++) {
          for (int j = 9; j >= 0; j--) {
                if (sequence[i] >= 0.1 * j) {
                      v[j]++;
                      break;
                }
          }
    }

    for (int i = 0; i < 10; i++) {
          x += pow((v[i] - n * 0.1), 2) / (n * 0.1);
    }
    cout << "Критерий Пирсона (порог критерия - 16.92):\n";
    cout << "Результат - " << x << "\n";

}

void Kolmogorov(long double* sequence, int n) {
    long double* sortedSequence = new long double[n];
    for (int i = 0; i < n; i++) {
          sortedSequence[i] = sequence[i];
    }
    sort(sortedSequence, sortedSequence + n);
    long double D = 0;

    for (int i = 0; i < n; i++) {
          long double k = abs((long double)(i + 1) / n - sortedSequence[i]);
          if (D < k) {
                D = k;
          }
    }
    D *= sqrt(n);
    cout << "Критерий Колмогорова (порог критерия - 1.36):\n";
    cout << "Результат - " << D << "\n";
    delete[] sortedSequence;
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int n = 1000;
    long double* sequence1 = new long double[n];
    MKM(sequence1, n, a0, a0);
```

```
    long double* sequence2 = new long double[n];
    MM(sequence2, n);

    cout << "Мультипликативный конгруэнтный метод:\n";
    Pirson(sequence1, n);
    Kolmogorov(sequence1, n);
    cout << "Метод Макларена-Марсальи:\n";
    Pirson(sequence2, n);
    Kolmogorov(sequence2, n);
    delete[] sequence1, sequence2;
    return 0;
}
```

Результаты:

Мультипликативный конгруэнтный метод:

$\sqrt{n}D_n = 0.83 < 1.36, \chi^2 = 7.64 < 16.92$

Метод Макларена-Марсальи:

$\sqrt{n}D_n = 0.93 < 1.36, \chi^2 = 11.72 < 16.92$

Лабораторная работа 2.

Условие:

Смоделировать дискретную случайную величину (задания на стр. 18-22). Исследовать точность моделирования.

1) Осуществить моделирование n = 1000 реализаций СВ из заданных дискретных распределений.

2) Вывести на экран несмещенные оценки математического ожидания и дисперсии, сравнить их с истинными значениями.

3) Для каждой из случайных величин построить свой χ2-критерий Пирсона с уровнем значимости ε=0.05. Проверить, что вероятность ошибки I рода стремится к 0.05.

4) Осуществить проверку каждой из сгенерированных выборок каждым из построенных критериев.

Теория:

Отрицательное биномиальное распределение (с параметрами m и p):

Случайная величина ξ принимает только целые неотрицательные значения, причем $P(\xi = k) = C_{k+m-1}^{k} p^m (1-p)^k$, $k \in Z$, $k \geq 0$. Параметры распределения: m - натуральное число, $p \in (0,1)$.

В данной работе, сначала моделировалась последовательность БСВ, а потом по каждой БСВ строился соответствующий элемент выборки отрицательного биномиального распределения: отрезок [0;1] разбивался на интервалы длин $C_{k+m-1}^{k} p^m (1-p)^k$ и проверялось, в какой интервал попадает элемент последовательности БСВ.

Распределение Пуассона (с параметром λ):

Случайная величина ξ принимает только целые неотрицательные значения, причем $P(\xi = k) = \frac{\lambda^k}{k!}$, $k \in Z$, $k \geq 0$.

В данной работе, сначала моделировалась последовательность БСВ, а потом по каждой БСВ строился соответствующий элемент выборки распределения Пуассона: отрезок [0;1] разбивался на интервалы длин $\frac{\lambda^k}{k!}$ и проверялось, в какой интервал попадает элемент последовательности БСВ.

Геометрическое распределение (с параметром p):

Случайная величина ξ принимает только целые неотрицательные значения, причем $P(\xi = k) = p\ (1 - p)^k$, $k \in Z$, $k \geq 0$. Параметр распределения $p \in (0, 1)$.

В данной работе, сначала моделировалась последовательность БСВ, а потом по каждой БСВ строился соответствующий элемент выборки геометрического распределения: отрезок [0;1] разбивался на интервалы длин $p\ (1 - p)^k$ и проверялось, в какой интервал попадает элемент последовательности БСВ.

Код программы:

```
#include <iostream>
#include <time.h>
#include <algorithm>
using namespace std;

void NegBi(int* sequence, int n, int r, double p) {
    srand(time(0));
    double q = 1 - p;
    double p0 = pow(p, r);
    for (int i = 0; i < n; i++) {
          p = p0;
          int z = 0;
          double a = (double)rand() / RAND_MAX;

          a -= p;
          while (a >= 0) {
                z++;
                p = p * q * (z - 1 + r) / z;
                a -= p;
          }
          sequence[i] = z;
    }
}

void P(int* sequence, int n, int lambda) {
    srand(time(0));
    double p0 = exp(-lambda), p;
    for (int i = 0; i < n; i++) {
          p = p0;
          int z = 0;
          double a = (double)rand() / RAND_MAX;

          a -= p;
          while (a >= 0) {
                z++;
                p = p * lambda / z;
                a -= p;
          }
```

```
            sequence[i] = z;
        }
    }

    void G(int* sequence, int n, double p) {
        srand(time(0));
        double p0 = p;
        double q = 1 - p;
        for (int i = 0; i < n; i++) {
            p = p0;
            int z = 0;
            double a = (double)rand() / RAND_MAX;

            a -= p;
            while (a >= 0) {
                z++;
                p *= q;
                a -= p;
            }
            sequence[i] = z;
        }
    }

    double E(int* sequence, int n) {
        double res = 0;
        for (int i = 0; i < n; i++) {
            res += (double)sequence[i];
        }
        return res / n;
    }

    double D(int* sequence, int n, double e) {
        double res = 0;
        for (int i = 0; i < n; i++) {
            double k = ((double)sequence[i] - e);
            res += k * k;
        }
        return res / (n - 1);
    }

    void Pirson(int* sequence, int n, double(*f)(int)) {
        int* sortedSequence = new int[n];
        for (int i = 0; i < n; i++) {
            sortedSequence[i] = sequence[i];
        }
        sort(sortedSequence, sortedSequence + n);
        int max = sortedSequence[n - 1];
        double x = 0;
        int count = 0;
        int j = 0;
        for (int i = 0; i <= max; i++) {
            count = 0;
            while (j < n && sortedSequence[j] == i) {
                count++;
                j++;
            }
            double p = f(i) * n;
            x += pow((count - p), 2) / p;
        }
        cout << "Критерий Пирсона (количество степеней свободы - " << max - 1 <<
")\n";
        cout << "Результат - " << x << "\n";
        delete[] sortedSequence;
    }
```

```
double fBi(int x) {
    double p = pow(0.25, 6);
    double q = pow(0.75, x);
    double C = 1;
    for (int i = 1; i <= x; i++) {
          C *= (double)(5 + i);
          C /= i;
    }
    return C * p * q;
}

double fP(int x) {
    int lambda = 3;
    double res = pow(lambda, x) * exp(-lambda);
    for (int i = 2; i <= x; i++) {
          res /= i;
    }
    return res;
}

double fG(int x) {
    double p = 0.25;
    return p * pow(1 - p, x);
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int n = 1000;
    int* sequenceBi = new int[n];
    int* sequenceP = new int[n];
    int* sequenceG = new int[n];

    NegBi(sequenceBi, n, 6, 0.25);
    P(sequenceP, n, 3);
    G(sequenceG, n, 0.25);

    double e = E(sequenceBi, n);
    double d = D(sequenceBi, n, e);
    cout << "Отрицательное биномиальное распределение (с параметрами 6 и
0.25):\n";
    cout << "Несмещенная оценка математического ожидания (истинное значение -
18): " << e <<"\n";
    cout << "Несмещенная оценка дисперсии (истинное значение - 72): " << d <<
"\n";
    Pirson(sequenceBi, n, fBi);
    cout << "\n";

    e = E(sequenceP, n);
    d = D(sequenceP, n, e);
    cout << "Распределение Пуассона (с параметром 3):\n";
    cout << "Несмещенная оценка математического ожидания (истинное значение -
3): " << e << "\n";
    cout << "Несмещенная оценка дисперсии (истинное значение - 3): " << d <<
"\n";
    Pirson(sequenceP, n, fP);
    cout << "\n";

    e = E(sequenceG, n);
    d = D(sequenceG, n, e);
    cout << Геометрическое распределение (с параметром 0.25):\n";
    cout << "Несмещенная оценка математического ожидания (истинное значение -
3): " << e << "\n";
    cout << "Несмещенная оценка дисперсии (истинное значение - 12): " << d <<
"\n";
```

```
    Pirson(sequenceG, n, fG);

    delete[] sequenceBi, sequenceP, sequenceG;
    return 0;
}
```

Результаты:

Отрицательное биномиальное распределение:

Несмещенная оценка математического ожидания (истинное значение - 18): 18.138

Несмещенная оценка дисперсии (истинное значение - 72): 73.3203

Критерий Пирсона (количество степеней свободы - 53)

Результат - 49.5619

Распределение Пуассона:

Несмещенная оценка математического ожидания (истинное значение - 3): 3.039

Несмещенная оценка дисперсии (истинное значение - 3): 2.96444

Критерий Пирсона (количество степеней свободы - 9)

Результат - 10.0552

Геометрическое распределение:

Несмещенная оценка математического ожидания (истинное значение - 3): 3.011

Несмещенная оценка дисперсии (истинное значение - 12): 12.4173

Критерий Пирсона (количество степеней свободы - 23)

Результат - 27.5227

Лабораторная работа 3.

Условие:

Смоделировать непрерывную случайную величину (задания на стр. 25-47). Исследовать точность моделирования.

1) Осуществить моделирование n = 1000 реализаций СВ из нормального закона распределения N(m, s2) с заданными параметрами. Вычислить несмещенные оценки математического ожидания и дисперсии, сравнить их с истинными.

2) Смоделировать n = 1000 СВ из заданных абсолютно непрерывных распределений. Вычислить несмещенные оценки математического ожидания и дисперсии, сравнить их с истинными значениями (если это возможно).

3) Для каждой из случайных величин построить свой критерий Колмогорова с уровнем значимость ε=0.05. Проверить, что вероятность ошибки I рода стремится к 0.05.

4) Для каждой из случайных величин построить свой $\chi 2$-критерий Пирсона с уровнем значимость ε=0.05. Проверить, что вероятность ошибки I рода стремится к 0.05.

5) Осуществить проверку каждой из сгенерированных выборок каждым из построенных критериев.

Теория:

Распределение Вейбулла-Гнеденко (с параметрами k, λ):

НСВ $\xi \in [0; +\infty)$ с плотностью распределения

$$p_{\xi}(x) = k \cdot \lambda x^{k-1} exp\{-\lambda x^{k}\},\ x \geq 0$$

имеет распределение Вейбулла-Гнеденко W(λ, k). Функция распределения имеет вид:

$$F_{\xi}(x) = 1 - exp\{-\lambda x^{k}\},\ x \geq 0$$

Среднее значение и дисперсия равны:

$$\mu = \lambda^{-1/k} \cdot \Gamma(1 + \frac{1}{k}),$$

$$\sigma^{2} = \lambda^{-2/k}(\Gamma(1 + \frac{2}{k}) - \Gamma^{2}(1 + \frac{1}{k})),$$

здесь Г(x)-гамма-функция Эйлера, то есть

$$\Gamma(x) = \int_{0}^{+\infty} x^{x-1} e^{-x} dx;\ \Gamma(k) = (k-1)!,\ k = 1, 2,...$$

Распределение может быть смоделировано методом обратной функции:

$$x = (-\frac{1}{\lambda} lnU)^{1/k},\ U - \text{БСВ}$$

Логистическое распределение (с параметрами k, μ):

НСВ $\xi \in R$ с плотностью распределения

$$p_{\xi}(x) = \frac{exp\{(x-\mu)/k\}}{k(1+exp\{(x-\mu)/k\})^{2}},\ x \in R$$

имеет логистическое распределение LG(k, μ), где μ-среднее значение, $k = \sqrt{3}\sigma/\pi > 0$, σ - стандартное отклонение СВ. Функция распределения имеет вид:

$$F_{\xi}(x) = (1 + exp\{-(x - \mu)/k\})^{-1}$$

Распределение может быть смоделировано методом обратной функции:

$$x = \mu + k \cdot ln(\frac{U}{1-U}),\ U - \text{БСВ}$$

Распределение Фишера (с параметрами l, m):

НСВ $\xi \in [0; +\infty)$ с плотностью распределения

$$p_{\xi}(x) = \frac{\Gamma(\frac{l+m}{2}) x^{\frac{l}{2}-1} (l/m)^{l/2}}{\Gamma(1/2)\Gamma(m/2)(1+x/m)^{(l+m)/2}},\ x \geq 0$$

имеет распределение Фишера F(l, m).

Среднее значение и дисперсия равны:

$$\mu = \frac{m}{m-2}, m > 2,$$

$$\sigma^2 = 2m^2(l + m - 2)/l(m - 2)^2(m - 4), m > 4$$

Распределение может быть смоделировано по формуле:

$$x = \frac{\xi_l/l}{\xi_m/m}, \text{ где } \xi_l \sim \chi^2(l),\ \xi_m \sim \chi^2(m)$$

χ^2-распределение (с параметром m):

НСВ $\xi \in [0;\ +\infty)$ с плотностью распределения

$$p_\xi(x) = \frac{x^{(m-2)/2}e^{-x/2}}{2^{m/2}\Gamma(m/2)},\ x \geq 0$$

имеет хи-квадрат распределение $\chi^2(m)$, m > 0 - натуральное число.

Среднее значение и дисперсия равны:

$$\mu = m,$$
$$\sigma^2 = 2m$$

Распределение может быть смоделировано по формуле:

$$x = \sum_{i=1}^{m} z_i^2,\ z_i \sim N(0,1)$$

Код программы:

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <time.h>
#include <gsl/gsl_sf.h>
#include <gsl/gsl_cdf.h>

using namespace std;

double m = -2, s = 1;
double a1 = 0.5, a2 = -1, b1 = 1, b2 = 2;
double m1 = 4, m2 = 3, l1 = 5;

void N(double* sequence, int n, double m, double s) {
   int k = 12;
   for (int i = 0; i < n; i++) {
         sequence[i] = 0;
         for (int j = 0; j < k; j++) {
               sequence[i] += (double)rand() / RAND_MAX;
         }
         sequence[i] -= 6;
         sequence[i] = m + sequence[i] * s;
   }
}

void W(double* sequence, int n, double a, double b) {
   for (int i = 0; i < n; i++) {
         double r = (double)rand() / RAND_MAX;
         sequence[i] = a * pow(-log(r), 1 / b);
   }
}

void LG(double* sequence, int n, double a, double b) {
   for (int i = 0; i < n; i++) {
```

```
            double r = (double)rand() / RAND_MAX;
            sequence[i] = a + b * log(r / (1 - r));
        }
    }

    void X(double* sequence, int n, double m) {
        double* r = new double[m];
        for (int i = 0; i < n; i++) {
            N(r, m, 0, 1);
            sequence[i] = 0;
            for (int j = 0; j < m; j++) {
                sequence[i] += r[j] * r[j];
            }
        }
        delete[] r;
    }

    void F(double* sequence, int n, double m, double l) {
        double* lr = new double[1];
        double* mr = new double[1];
        for (int i = 0; i < n; i++) {
            X(lr, 1, l);
            X(mr, 1, m);
            sequence[i] = (lr[0] / l) / (mr[0] / m);
        }
        delete[] lr, mr;
    }

    double E(double* sequence, int n) {
        double res = 0;
        for (int i = 0; i < n; i++) {
            res += (double)sequence[i];
        }
        return res / n;
    }

    double D(double* sequence, int n, double e) {
        double res = 0;
        for (int i = 0; i < n; i++) {
            double k = ((double)sequence[i] - e);
            res += k * k;
        }
        return res / (n - 1);

    }

    double Pirson(double* sequence, int n, double(*f)(double)) {
        double v[10] = { 0 };
        double x = 0;
        double* sortedSequence = new double[n];
        for (int i = 0; i < n; i++) {
            sortedSequence[i] = sequence[i];
        }
        sort(sortedSequence, sortedSequence + n);
        double step = (sortedSequence[n - 1] - sortedSequence[0]) / 10;
        int i = 0;
        for (int j = 0; i < n && j < 10; j++) {
            while (sortedSequence[i] < sortedSequence[0] + j * step + step &&
i < n) {
                v[j]++;
                i++;
            }
        }
        double curr = sortedSequence[0];
        for (int i = 0; i < 10; i++) {
```

```
            double p = f(curr + step) - f(curr);
            x += pow(v[i] - n * p, 2) / (n * p);
            curr += step;
        }
        delete[] sortedSequence;
        return x;
    }

    double Kolmogorov(double* sequence, int n, double(*f)(double)) {
        double* sortedSequence = new double[n];
        for (int i = 0; i < n; i++) {
            sortedSequence[i] = sequence[i];
        }
        sort(sortedSequence, sortedSequence + n);
        double D = 0;
        for (int i = 0; i < n; i++) {
            double k = abs((double)(i + 1) / n - f(sortedSequence[i]));
            if (D < k) {
                D = k;
            }
        }
        D *= sqrt(n);
        delete[] sortedSequence;
        return D;
    }

    double fN(double x) {
        return gsl_cdf_ugaussian_P((x - m) / s);
    }

    double fW(double x) {
        return 1 - exp(-pow((x / a1), b1));
    }

    double fLG(double x) {
        return 1 / (1 + exp(-(x - a2) / b2));
    }

    double fX(double x) {
        return gsl_sf_gamma_inc_P(m1 / 2, x / 2);
    }

    double fF(double x) {
        return gsl_sf_beta_inc(l1 / 2, m2 / 2, l1 * x / (l1 * x + m2));
    }

    int main()
    {
        setlocale(LC_ALL, "Russian");
        srand(time(0));

        int n = 1000;
        double* sequenceN = new double[n];
        double* sequenceW = new double[n];
        double* sequenceLG = new double[n];
        double* sequenceX = new double[n];
        double* sequenceF = new double[n];

        N(sequenceN, n, m, s);
        W(sequenceW, n, a1, b1);
        LG(sequenceLG, n, a2, b2);
        X(sequenceX, n, m1);
        F(sequenceF, n, m2, l1);

        double e = E(sequenceN, n);
```

```
	double d = D(sequenceN, n, e);
	double realE = m;
	double realD = s;
	double pirson = Pirson(sequenceN, n, fN);
	double kolmogorov = Kolmogorov(sequenceN, n, fN);
	cout << "\nНормальное распределение:\n";
	cout << "Несмещенная оценка математического ожидания (истинное значение
- " << realE <<"): " << e << "\n";
	cout << "Несмещенная оценка дисперсии (истинное значение - " << realD
<< "): " << d << "\n";
	cout << "Критерий Пирсона (порог критерия - 16.92):\n";
	cout << "Результат - " << pirson << "\n";
	int mistakeCount = 0;
	for (int i = 0; i < 1000; i++) {
		N(sequenceN, n, m, s);
		if (Pirson(sequenceN, n, fN) > 16.92) {
			mistakeCount++;
		}
	}
	cout << "Процент ошибок первого рода - " << (double)mistakeCount / 10
<< "\n";
	cout << "Критерий Колмогорова (порог критерия - 1.36):\n";
	cout << "Результат - " << kolmogorov << "\n";
	mistakeCount = 0;
	for (int i = 0; i < 1000; i++) {
		N(sequenceN, n, m, s);
		if (Kolmogorov(sequenceN, n, fN) > 1.36) {
			mistakeCount++;
		}
	}
	cout << "Процент ошибок первого рода - " << (double)mistakeCount / 10
<< "\n";

	e = E(sequenceW, n);
	d = D(sequenceW, n, e);
	realE = a1 * tgamma(1 / b1 + 1);
	realD = a1 * a1 * (tgamma(2 / b1 + 1) - pow(tgamma(1 / b1 + 1), 2));
	pirson = Pirson(sequenceW, n, fW);
	kolmogorov = Kolmogorov(sequenceW, n, fW);
	cout << "\nРаспределение Вейбулла:\n";
	cout << "Несмещенная оценка математического ожидания (истинное значение
- " << realE << "): " << e << "\n";
	cout << "Несмещенная оценка дисперсии (истинное значение - " << realD
<< "): " << d << "\n";
	cout << "Критерий Пирсона (порог критерия - 16.92):\n";
	cout << "Результат - " << pirson << "\n";
	cout << "Критерий Колмогорова (порог критерия - 1.36):\n";
	cout << "Результат - " << kolmogorov << "\n";

	e = E(sequenceLG, n);
	d = D(sequenceLG, n, e);
	realE = a2;
	realD = b2 * b2 * acos(-1) * acos(-1) / 3;
	pirson = Pirson(sequenceLG, n, fLG);
	kolmogorov = Kolmogorov(sequenceLG, n, fLG);
	cout << "\nЛогистическое распределение:\n";
	cout << "Несмещенная оценка математического ожидания (истинное значение
- " << realE << "): " << e << "\n";
	cout << "Несмещенная оценка дисперсии (истинное значение - " << realD
<< "): " << d << "\n";
	cout << "Критерий Пирсона (порог критерия - 16.92):\n";
	cout << "Результат - " << pirson << "\n";
	cout << "Критерий Колмогорова (порог критерия - 1.36):\n";
	cout << "Результат - " << kolmogorov << "\n";
```

```
        e = E(sequenceX, n);
        d = D(sequenceX, n, e);
        realE = m1;
        realD = 2 * m1;
        pirson = Pirson(sequenceX, n, fX);
        kolmogorov = Kolmogorov(sequenceX, n, fX);
        cout << "\nРаспределение Хи-квадрат:\n";
        cout << "Несмещенная оценка математического ожидания (истинное значение
- " << realE << "): " << e << "\n";
        cout << "Несмещенная оценка дисперсии (истинное значение - " << realD
<< "): " << d << "\n";
        cout << "Критерий Пирсона (порог критерия - 16.92):\n";
        cout << "Результат - " << pirson << "\n";
        cout << "Критерий Колмогорова (порог критерия - 1.36):\n";
        cout << "Результат - " << kolmogorov << "\n";

        e = E(sequenceF, n);
        d = D(sequenceF, n, e);
        realE = m2 / (m2 - 2);
        realD = 2 * l1 * l1 * (m2 + l1 - 2) / m2 / (l1 - 4) / (m2 - 2) / (m2 -
2);
        pirson = Pirson(sequenceF, n, fF);
        kolmogorov = Kolmogorov(sequenceF, n, fF);
        cout << "\nРаспределение Фишера:\n";
        cout << "Несмещенная оценка математического ожидания (истинное значение
- " << realE << "): " << e << "\n";
        cout << "Несмещенная оценка дисперсии: " << d << "\n";
        cout << "Критерий Пирсона (порог критерия - 16.92):\n";
        cout << "Результат - " << pirson << "\n";
        cout << "Критерий Колмогорова (порог критерия - 1.36):\n";
        cout << "Результат - " << kolmogorov << "\n";

        delete[] sequenceN, sequenceW, sequenceLG, sequenceX, sequenceF;
        return 0;
    }
```

Результат:

Нормальное распределение:

Несмещенная оценка математического ожидания (истинное значение - -2): -2.02608

Несмещенная оценка дисперсии (истинное значение - 1): 1.03288

Критерий Пирсона (порог критерия - 16.92):

Результат - 12.1765

Критерий Колмогорова (порог критерия - 1.36):

Результат - 0.929363

Распределение Вейбулла:

Несмещенная оценка математического ожидания (истинное значение - 0.5): 0.516864

Несмещенная оценка дисперсии (истинное значение - 0.25): 0.259032

Критерий Пирсона (порог критерия - 16.92):

Результат - 5.13084

Критерий Колмогорова (порог критерия - 1.36):

Результат - 1.15681

Логистическое распределение:

Несмещенная оценка математического ожидания (истинное значение - -1): -0.947544

Несмещенная оценка дисперсии (истинное значение - 13.1595): 13.4934

Критерий Пирсона (порог критерия - 16.92):

Результат - 10.5762

Критерий Колмогорова (порог критерия - 1.36):

Результат - 0.519319

Распределение Хи-квадрат:

Несмещенная оценка математического ожидания (истинное значение - 4): 3.95465

Несмещенная оценка дисперсии (истинное значение - 8): 7.19646

Критерий Пирсона (порог критерия - 16.92):

Результат - 10.6012

Критерий Колмогорова (порог критерия - 1.36):

Результат - 0.99958

Распределение Фишера:

Несмещенная оценка математического ожидания (истинное значение - 3): 2.8252

Несмещенная оценка дисперсии: 87.3072

Критерий Пирсона (порог критерия - 16.92):

Результат - 7.32921

Критерий Колмогорова (порог критерия - 1.36):

Результат - 0.638314

Лабораторная работа 4.

Условие:

Вычислить значение интеграла, используя метод Монте-Карло. Оценить точность.

1. По методу Монте-Карло вычислить приближенное значения интегралов.
2. Сравнить полученное значение либо с точным значением (если его получится вычислить), либо с приближенным, полученным в каком-либо математическом пакете (например, в mathematica). Для этого построить график зависимости точности вычисленного методом Монте-Карло интеграла от числа итераций *n*.

Интегралы:

$$\int_0^{\infty} e^{-x}\sqrt{1+x}dx,$$

$$\int_0^1 \int_0^2 (x^2+y^2)dxdy$$

Теория:

Метод Монте-Карло приближенного вычисления интеграла:

Необходимо вычислить $a = \int_A g(x)dx,\ x \in R^n, A \subseteq R^n$.

Пусть η- произвольная случайная величина с плотностью распределения $P_\eta(x),\ x \in A$имеющая конечный момент второго порядка.

Рассмотрим СВ $\xi = \frac{g(\eta)}{P_\eta(\eta)}$. Для нее справедливо$E\{\xi\} = a,\ D\{\xi\} < \infty$.

Значит, в качестве приближенного значения a можно взять

$$\frac{1}{n}\sum_{i=1}^{n}\xi_i = \frac{1}{n}\sum_{i=1}^{n}\frac{g(\eta_i)}{P_\eta(\eta_i)}$$

В данной работе в качестве ηдля первого интеграла бралась экспоненциально распределенная случайная величина (с параметром 1), а для второго интеграла бралась случайная величина, равномерно распределенная на $[0; 1] \times [0; 2]$.

Код программы:

```
#include <iostream>
#include <time.h>

using namespace std;

double a = 1;

void E(double* sequence, int n, double a) {
	double r;
	for (int i = 0; i < n; i++) {
		do {
			r = (double)rand() / RAND_MAX;
		} while (r == 0);
		sequence[i] = -log(r) / a;
	}
}

double pE(double x) {
	return a * exp(-a * x);
}

double calculateIntegral(double(*g)(double), double(*p)(double), double*
sequence, int n) {
	double result = 0;
	for (int i = 0; i < n; i++) {
		result += (g(sequence[i]) / p(sequence[i]) / n);
	}
	return result;
}

double calculateDoubleIntegral(double(*g)(double, double),
double(*p)(double, double), double* sequence1, double* sequence2, int n) {
	double result = 0;
	for (int i = 0; i < n; i++) {
		result += (g(sequence1[i], sequence2[i]) / p(sequence1[i],
sequence2[i]) / n);
	}
	return result;
}
```

```
double g1(double x) {
    return exp(-x) * sqrt(1 + x);
}

double g2(double x, double y) {
    return x * x + y * y;
}

double p2(double x, double y) {
    return 1.0 / 2.0;
}

int main()
{
    srand(time(0));

    int n[] = { 10, 20, 50, 100, 500, 1000 };
    double realIntegral1 = 1.37894;
    double realIntegral2 = 10.0 / 3.0;
    double integral1;
    double integral2;
    double* sequence1;
    double* sequence21;
    double* sequence22;


    printf("ListLinePlot[{");
    for (int i = 0; i < _countof(n); i++) {
        sequence1 = new double[n[i]];

        E(sequence1, n[i], a);

        integral1 = calculateIntegral(g1, pE, sequence1, n[i]);

        printf("{%d, %lf}", n[i], 1 - abs(integral1 - realIntegral1) /
realIntegral1);
        if (i != _countof(n) - 1) {
            printf(", ");
        }
    }
    printf("}]\n");

    printf("ListLinePlot[{");
    for (int i = 0; i < _countof(n); i++) {
        sequence21 = new double[n[i]];
        sequence22 = new double[n[i]];

        for (int j = 0; j < n[i]; j++) {
            sequence21[j] = (double)rand() / RAND_MAX * 2;
            sequence22[j] = (double)rand() / RAND_MAX;
        }

        integral2 = calculateDoubleIntegral(g2, p2, sequence21, sequence22,
n[i]);

        printf("{%d, %lf}", n[i], 1 - abs(integral2 - realIntegral2) /
realIntegral2);
        if (i != _countof(n) - 1) {
            printf(", ");
        }
    }
    printf("}]\n");

    delete[] sequence1;
```

```
    return 0;
}
```

Результат:

При n=1000 для первого интеграла:

Реальное значение - 1.378940.

Подсчитанное приближение - 1.380619.

Для второго интеграла:

Реальное значение - 3.333333.

Подсчитанное приближение - 3.389709.

Лабораторная работа 5.

Условие:

Решить систему линейных уравнений, используя метод Монте-Карло.

1. Решить систему линейных алгебраических уравнений методом Монте-Карло.
2. Сравнить с решением данного уравнения, полученным в произвольном математическом пакете.
3. Построить график зависимости точности решения от длины цепи маркова и числа смоделированных цепей маркова.

$$Ax=f$$

$$A = \begin{pmatrix} 1.2 & -0.3 & 0.4 \\ 0.4 & 0.7 & -0.2 \\ 0.2 & -0.3 & 0.9 \end{pmatrix}, f = \begin{pmatrix} -4 \\ 2 \\ 0 \end{pmatrix}$$

Теория:

Метод Монте-Карло приближенного решения системы линейных алгебраических уравнений:

Необходимо решить систему, представленную в виде $x=Ax+f$, где $x = (x_1,\dots, x_n)^T$, $f = (f_1,\dots, f_n)^T$, $A = (a_{ij})$, $i,j = \overline{1,n}$, собственные значения A по модулю меньше 1.

Наша цель – вычислить скалярное произведение вектора решения $x = (x_1,\dots, x_n)^T$ с некоторым вектором $h = (h_1,\dots, h_n)^T$. Для нахождения x_i берем h такой, что $h_j = 0\ \forall j \neq i$ и $h_i = 1$.

Рассмотрим цепь Маркова с параметрами $\pi = (\pi_1,\dots, \pi_n)^T$, $P = (p_{ij})$ такими, что

$$\pi_i \geq 0,\ \sum_{i=1}^{n} \pi_i = 1,\ p_{ij} \geq 0,\ \sum_{j=1}^{n} p_{ij} = 1,\ i = \overline{1,n},$$

$$\pi_i > 0, \text{ если } h_i \neq 0,\ p_{ij} > 0, \text{ если } a_{ij} \neq 0$$

Положим

$$g_i^{(0)} = h_i/\pi_i,\ \pi_i > 0\ (\text{при}\ \pi_i = 0\ g_i^{(0)} = 0),$$
$$g_i^{(k)} = a_{ij}/p_{ij},\ p_{ij} > 0\ (\text{при}\ p_{ij} = 0\ g_i^{(k)} = 0),$$

Выберем некоторое натуральное N и рассмотрим случайную величину

$$\xi_N = \sum_{m=0}^{N} Q_m f_{i_m}$$

Где $i_0 \to i_1 \to \ldots \to i_m$ - траектория цепи Маркова, а Q_m определяется как:

$$Q_0 = g_{i_0}^{(0)},\ Q_m = Q_{m-1} g_{i_{m-1} i_m}^{(m)}$$

Тогда скалярное произведение векторов h и x приблизительно равно $E\{\xi_N\}$.

В данной работе брали $\pi_i = \frac{1}{3},\ p_{ij} = \frac{1}{2}$.

Код программы:s

```
#include <iostream>
#include <time.h>

using namespace std;

double* calculate(double** A, double* f, int n, int L, int N) {
	double* result = new double[n] {0};
	int i;
	int iPrev;
	double Q;
	double ksi;
	double x;

	for (int ind = 0; ind < n; ind++) {
		x = 0;
		for (int j = 0; j < N; j++)
		{
			do {
				i = floor((double)rand() / RAND_MAX * n);
			} while (i == n);

			if (ind == i) {
				Q = n;
				ksi = Q * f[i];
				for (int k = 1; k <= L; k++)
				{
					iPrev = i;
					do {
						i = floor((double)rand() / RAND_MAX * n);
					} while (i == n);
					Q *= A[iPrev][i] * n;
					ksi += Q * f[i];
				}
				x += ksi;
			}
		}
		result[ind] = x / N;
	}

	return result;
}
```

```
int main()
{
	srand(time(0));
	int n = 3;

	double* f = new double[3]{ -4, 2, 0 };
	double** A = new double* [3]{
		new double[3]{ 1.2, -0.3, 0.4 },
		new double[3]{ 0.4, 0.7, -0.2 },
		new double[3]{ 0.2, -0.3, 0.9 }
	};
	double realX[3]{ -2.82857, 5.14286, 2.34286 };

	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			A[i][j] *= -1;
		}
		A[i][i]++;
	}

	int lengths[]{ 100, 200, 500, 1000, 2000 };
	int amounts[]{ 100, 200, 500, 1000, 2000 };
	double* x;
	double norm;
	double* bestX = new double[n] {0};
	int bestLength;
	int bestAmount;
	double bestNorm = 1000;
	printf("ListPlot3D[{");
	for (int i = 0; i < _countof(lengths); i++) {
		for (int j = 0; j < _countof(amounts); j++)
		{
			norm = 0;
			x = calculate(A, f, 3, lengths[i], amounts[j]);
			for (int k = 0; k < n; k++) {
				norm += (x[k] - realX[k]) * (x[k] - realX[k]);
			}
			norm = sqrt(norm);
			if (norm < bestNorm) {
				bestNorm = norm;
				bestLength = lengths[i];
				bestAmount = amounts[j];
				for (int j = 0; j < n; j++) {
					bestX[j] = x[j];
				}
			}
			printf("{%d, %d, %lf}", lengths[i], amounts[j], norm);
			if (i != _countof(lengths) - 1 || j != _countof(amounts) - 1)
{
				printf(", ");
			}
		}
	}
	printf("}]\n\n");

	for (int i = 0; i < n; i++) {
		cout << bestX[i] << " ";
	}
	cout << "\n";
	cout << "best length - " << bestLength;
	cout << "\n";
	cout << "best amount - " << bestAmount;
	cout << "\n";
```

```
    for (int i = 0; i < n; i++) {
          cout << realX[i] << " ";
    }
    cout << "\n";

    for (int i = 0; i < n; i++) {
          delete[] A[i];
    }
    delete[] x, f, A, bestX;

    return 0;
}
```

Результат:

Реальное решение - $(-\ 2.82857,\ 5.14286,\ 2.34286)^T$.

Полученное приближение (при моделировании 1000 цепей Маркова длины 100000) - $(-\ 2.89962,\ 5.16828,\ 2.2997)^T$.

Литература

1. Харин Ю.С., Малюгин В.И., Кирлица В.П., Лобач В.И., Хацкевич Г.А. Основы имитационного и статистического моделирования. Учебное пособие. Минск: ДизайнПРО, 1997 – 228 с.

2. Лобач В.И., Кирлица В.П., Малюгин В.И., Сталевская С.Н. Имитационное и статистическое моделирование. Практикум для студентов математических и экономических специальностей. Минск, БГУ, 2004 –189 с.