МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра методов оптимального управления

Чубакова Валерия Вадимовна

Отчёт по лабораторной работе №3 по курсу «Имитационное и статистическое моделирование»

студентки 4 курса 8а группы

Преподаватель:

Лобач Виктор Иванович
доцент кафедры ММАД,
канд. физ.-мат. наук

Работа сдана _____ 2020 г.

Зачтена _____ 2020 г.

_____

Минск 2020

Условие:

Смоделировать непрерывную случайную величину. Исследовать точность моделирования.

1. Осуществить моделирование $n = 1000$ реализаций СВ из нормального закона распределения $N(m, s^2)$ с заданными параметрами. Вычислить несмещённые оценки математического ожидания и дисперсии, сравнить их с истинными.

2. Смоделировать $n = 1000$ СВ из заданных абсолютно непрерывных распределений. Вычислить несмещённые 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 значимости $\varepsilon = 0.05$. Проверить, что вероятность ошибки I рода стремится к 0.05.

4. Для каждой из случайных величин построить свой χ^2-критерий Пирсона с уровнем значимости $\varepsilon = 0.05$. Проверить, что вероятность ошибки I рода стремится к 0.05.

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

Теория:

Нормальное распределение (0,64):

НСВ $\xi \in \mathbb{R}$ с плотностью распределения

$$p_\xi(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

имеет одномерное нормальное распределение $N_1(\mu, \sigma^2)$ с параметрами: средним значением $\mu \in \mathbb{R}$ и дисперсией $\sigma^2 > 0$.

Математическое ожидание: μ.

Дисперсия: σ^2.

Распределение $N_1(0,1)$ называется стандартным нормальным распределением, а НСВ $\eta \sim N_1(0,1)$ —стандартной нормальной (гаусовской) величиной. Случайные величины ξ и η связаны соотношением: $\xi = \eta + \sigma \cdot \eta$, где ξ имеет распределение $N_1(\mu, \sigma^2)$.

χ^2–распределение(4):

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

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

имеет χ^2–распределение $\chi^2(m)$ с m степениями свободы ($m > 0, \quad m \in \mathbb{N}$).Здесь $\Gamma(z)$ — гамма-функция Эйлера.

Среднее значение: $\mu = m$.

Дисперсия: $\sigma^2 = 2m$.

Известно, что, если $\eta_1, \ldots, \eta_m \sim N_1(0,1)$ — независимые стандартные гаусовские СВ, то СВ $\xi = \sum_{i=1}^m \eta_i^2$ имеют χ^2–распределение.

Распределение Фишера (5,3):

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

$$p_\xi(x) = \frac{\Gamma\left(\frac{l+m}{2}\right) x^{\frac{l}{2}-1} \left(\frac{l}{m}\right)^{\frac{l}{2}}}{\Gamma\left(\frac{l}{2}\right)\Gamma\left(\frac{m}{2}\right)\left(1+\frac{l}{m}x\right)^{\frac{l+m}{2}}}, \quad x \geqslant 0$$

имеет распределение Фишера $F(l, m)$ с l и m числом степеней свободы. Здесь: l, m — натуральные числа, параметры распределения; $\Gamma(z)$ — гамма-функция Эйлера.

Среднее значение: $\mu = \dfrac{m}{m-2}, \quad m > 2.$

Дисперсия: $\sigma^2 = \dfrac{2m^2(l+m-2)}{l(m-2)^2(m-4)}, \quad m > 4.$

Пусть $\xi_l \sim \chi^2(l), \xi_m \sim \chi^2(m)$ — НСВ, а СВ ξ определяется соотношением

$$\xi = \frac{\xi_l/l}{\xi_m/m},$$

тогда СВ ξ имеет распределение $F(l, m)$.

Результат выполнения:

```
Normal distribution: mu = 0, s_2 = 64
size = 1000
mu = 0
mean = 0.11
s_2 = 64
dispersion = 61.759568202351716
Kolmogorov test: 1.1172370734301782 < 1.36
Pearson test: 13.589919026696117 < 16.9


chi^2 distribution: m = 4
size = 1000
mu = 4
mean = 3.34
s_2 = 8.0
dispersion = 7.352233890566931
Kolmogorov test: 1.1117704616969226 < 1.36
Pearson test: 10.604705464737668 < 16.9


Fisher's distribution: l = 3 m = 5
size = 1000
mu = 1
mean = 1.294
s_2 = 11.11111111111111
dispersion = 9.720292919616401
Kolmogorov test: 0.9314553965477647 < 1.36
Pearson test: 7.5312940782174325 < 16.9
```

Код программы:

```
1  import java.io.FileNotFoundException;
2  import java.io.PrintStream;
3  import java.lang.reflect.Array;
4  import java.util.Arrays;
5  import static java.lang.Math.*;
6
7  public class SSM_Main3 {
8      public static double erf(double z) {
9          double t = 1.0 / (1.0 + 0.5 * Math.abs(z));
10
11         // use Horner's method
12         double ans = 1 - t * Math.exp( -z*z - 1.26551223 +
13                 t * ( 1.00002368 +
14                         t * ( 0.37409196 +
15                                 t * ( 0.09678418 +
16                                         t * (-0.18628806 +
17                                                 t * ( 0.27886807 +
18                                                         t * (-1.13520398 +
19                                                                 t * ( 1.48851587 +
20                                                                         t * (-0.82215223 +
21                                                                                 t * ( 0.17087277))))
                                                                                     ))))));
22         if (z >= 0) return ans;
23         else       return -ans;
24     }
25     public static double G(double x) {
26         double[] p = {0.99999999999980993, 676.5203681218851, -1259.1392167224028,
27                 771.32342877765313, -176.61502916214059, 12.507343278686905,
28                 -0.13857109526572012, 9.984369578019571 6e-6, 1.5056327351493116e-7};
29         int g = 7;
30         if (x < 0.5) return Math.PI / (Math.sin(Math.PI * x) * G(1 - x));
31         x -= 1;
32         double a = p[0];
33         double t = x + g + 0.5;
34         for (int i = 1; i < p.length; i++) {
35             a += p[i] / (x + i);
36         }
37         return Math.sqrt(2 * Math.PI) * Math.pow(t, x + 0.5) * Math.exp(-t) * a;
38     }
39     public static int fact(int x) {
40         if (x == 0) return 1;
41         if (x == 1) return 1;
42         return x * fact(x - 1);
43     }
44     public static double g(int m, double x) {
45         double res = (double)fact(m - 1);
46         res *= exp(-x);
47         double sum = 0;
48         for (int i = 0; i <= m - 1; i++) {
49             sum += pow(x, i) / fact(i);
50         }
51         res*=sum;
```

```
52          return 1-res;
53      }
54      public static double B(double a, double b){
55          return G(a)*G(b)/G(a+b);
56      }
57      public static double I(double x, double a, double b) {
58          double res = pow(x, a) * pow((1 - x), b) / (a * B(a, b));
59          double sum = 1;
60          for (int n = 0; n < 20; n++) {
61              sum += B(a + 1, n + 1) * pow(x, n + 1) / B(a + b, n + 1);
62          }
63          res *= sum;
64          return res;
65      }
66      public static void print(double[] A) {
67          System.out.println("");
68          for (int i = 0; i < A.length; i++)
69              System.out.print(A[i] + " ");
70      }
71      public static void print(int[] A) {
72          System.out.println("");
73          for (int i = 0; i < A.length; i++)
74              System.out.print(A[i] + " ");
75      }
76      public static int[] freq(double[] A, int K) {
77          Arrays.sort(A);
78          double interval = (A[A.length - 1] - A[0]) / K;
79          int[] frequencies = new int[K];
80          for (int i = 0; i < A.length; i++)
81              for (int j = 1; j <= K; j++)
82                  if (A[i] < A[0] + j * interval)
83                      if (A[i] >= A[0] + (j - 1) * interval)
84                          frequencies[j - 1]++;
85          return frequencies;
86      }
87      public static double[] mult_cong(long alpha_0, long beta, int n, long M) {
88          long buffer[] = new long[n];
89          buffer[0] = (long) alpha_0;
90          double A[] = new double[n];
91          A[0] = (double) buffer[0] / M;
92          for (int i = 1; i < n; i++) {
93              long tmp = beta * buffer[i - 1];
94              buffer[i] = tmp - M * (tmp / M);
95              A[i] = (double) buffer[i] / M;
96          }
97          return A;
98      }
99      public static double mean(double[] A) {
100         long sum = 0;
101         for (int i = 0; i < A.length; i++)
102             sum += A[i];
103         double res = sum / (double) A.length;
104         return res;
105     }
106     public static double dispersion(double[] A) {
```

```
        double mean_A = mean(A);
        double res = 0;
        for (int i = 0; i < A.length; i++)
            res += pow((A[i] - mean_A), 2);
        res = res / (double) (A.length - 1);
        return res;
    }
    public static double empiricalFunction(double[] A, double x) {
        double result = 0;
        for (int i = 0; i < A.length; i++) {
            if (A[i] <= x) result++;
        }
        return result / A.length;
    }


    public static double[] normal(int n, int m, int s_2) {
        double s = sqrt((double) s_2);
        int N = 12;
        double result[] = new double[n];
        double sum, eta;
        long alpha = 65643;
        double[] A = mult_cong(alpha, alpha, n * N, 2147483648L);
        for (int i = 0; i < n; i++) {
            sum = 0;
            for (int j = 0; j < N; j++) {
                sum += A[N * i + j];
            }
            eta = sqrt(12. / N) * (sum - N / 2.);
            result[i] = m + s * eta;
        }
        return result;
    }

    public static double Pearson_normal(double[] A) {
        int K = 10;
        double theor_freq, xi_2 = 0;
        int[] frequencies = freq(A, K);
        double interval = (A[A.length - 1] - A[0]) / K;
        double mean_ = mean(A);
        double s_2 = dispersion(A);

        for (int i = 0; i < K; i++) {
            double x = A[0] + ((i + 0.5) * interval);
            double u = pow(x - mean_, 2);
            double p_k = interval / sqrt(2 * PI * s_2) * exp(-u / (2 * s_2));
            theor_freq = A.length * p_k;
            xi_2 += pow((frequencies[i] - theor_freq), 2) / theor_freq;
        }
        return xi_2;
    }

    public static double Kolmogorov_normal(double[] A, int mu, int s_2) {
        double D_n = 0;
        double empiricalFRes, theoreticalFRes;
```

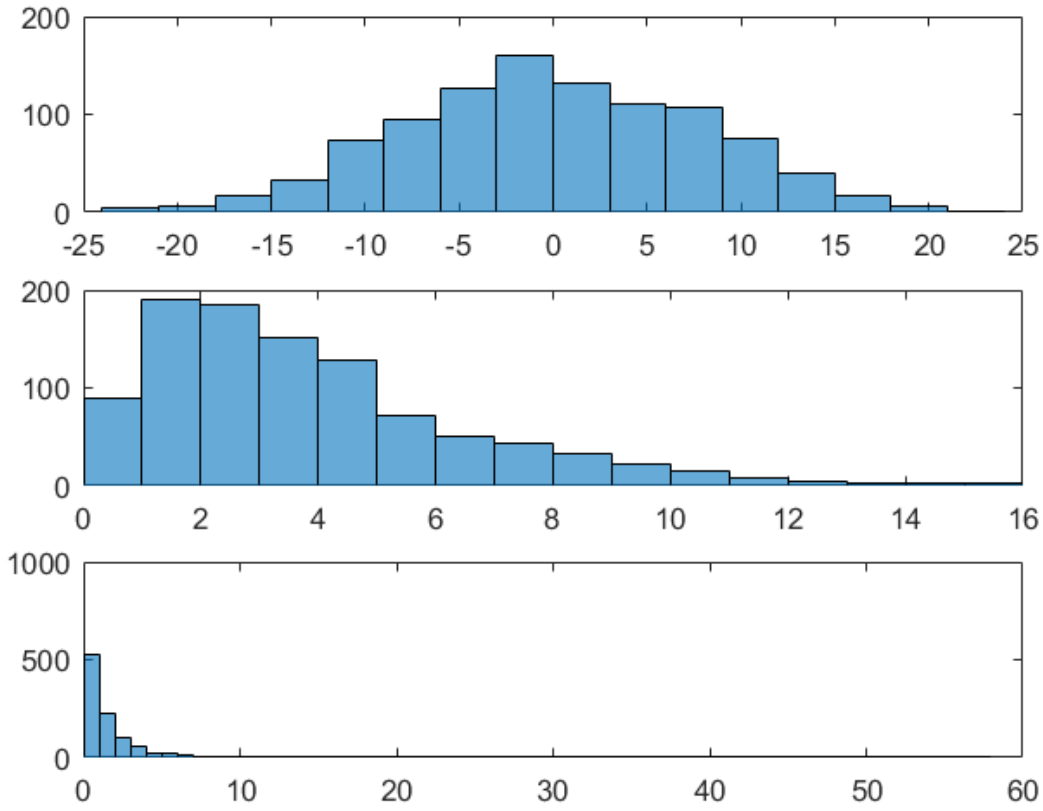
```
        for (int i = 0; i < A.length; i++) {
            empiricalFRes = empiricalFunction(A, A[i]);
            theoreticalFRes = theorNormFunction(A, A[i]);
            D_n = Math.max(D_n, Math.abs(empiricalFRes - theoreticalFRes));
        }
        return D_n*sqrt(A.length);
    }

    public static double theorNormFunction(double[] A, double x) {
        return 0.5 * (1. + erf((x - mean(A)) / sqrt(2 * dispersion(A))));
    }


    public static double[] chi_2(int n, int m, long alpha) {
        double[] result = new double[n];
        int N = 12;
        double[] A = mult_cong(alpha, alpha, n * m * N, 2147483648L);
        double sum, big_sum;
        double eta;
        for (int i = 0; i < n; i++) {
            big_sum = 0;
            for (int j = 0; j < m; j++) {
                sum = 0;
                for (int k = 0; k < N; k++) {
                    sum += A[12 * m * i + 12 * j + k];
                }
                eta = sqrt(12. / N) * (sum - N / 2.);
                big_sum += pow(eta, 2);
            }
            result[i] = big_sum;
        }
        return result;
    }

    public static double Pearson_chi_2(double[] A, int m) {
        int K = 10;
        int s = 10;
        double theor_freq, xi_2 = 0;
        int[] frequencies = freq(A, K);
        double interval = (A[A.length - 1] - A[0]) / K;
        for (int i = 0; i < K; i++) {
            double p_k = 0;
            for (int j = 0; j < s; j++) {
                double x = A[0] + ((i + (double) j / s) * interval);
                double chisl = pow(x, (m - 2.) / 2) * exp(-x / 2);
                double znam = pow(2, m / 2.) * G(m / 2.);
                p_k += interval / s * chisl / znam;
            }
            theor_freq = A.length * p_k;
            xi_2 += pow((frequencies[i] - theor_freq), 2) / theor_freq;
        }
        return xi_2;
    }

    public static double Kolmogorov_chi_2(double[] A, int m) {
```

```
217         Arrays.sort(A);
218         double D_n = 0;
219         double empiricalFRes, theoreticalFRes;
220         for (int i = 0; i < A.length; i++) {
221             empiricalFRes = empiricalFunction(A, A[i]);
222             theoreticalFRes = theorChi2Function(A, A[i], m);
223             D_n = Math.max(D_n, Math.abs(empiricalFRes - theoreticalFRes));
224            // System.out.println(empiricalFRes+" "+theoreticalFRes);
225         }
226         return D_n*sqrt(A.length);
227     }
228
229     public static double theorChi2Function(double[] A, double x, int m) {
230         return g(m/2,x/2)/G(m / 2.);
231     }
232
233
234     public static double[] Fisher(int n, int l, int m) {
235         double[] result = new double[n];
236         long alpha1 = 262147;
237         long alpha2 = 78125;
238         double[] chi_l = chi_2(n, l, alpha1);
239         double[] chi_m = chi_2(n, m, alpha2);
240         for (int i = 0; i < n; i++) {
241             result[i] = (chi_l[i] / (double) l) / (chi_m[i] / (double) m);
242         }
243         return result;
244     }
245
246     public static double Pearson_Fisher(double[] A, int l, int m) {
247         int K = 10;
248         int s = 50;
249         int[] frequencies = freq(A, K);
250         double interval = (A[A.length - 1] - A[0]) / K;
251         double theor_freq, xi_2 = 0;
252
253         double lm2 = (l + m) / 2.;
254         double l22 = (l - 2.) / 2.;
255         double lm = (double) l / m;
256         double l2 = l / 2.;
257         double m2 = m / 2.;
258
259         for (int i = 0; i < K; i++) {
260             double p_k = 0;
261             for (int j = 0; j < s; j++) {
262                 double x = A[0] + ((i + (double) j / s) * interval);
263                 double m1x = 1. + l * x / m;
264                 p_k += interval / s * G(lm2) * pow(x, l22) * pow(lm, l2) /
265                        (G(l2) * G(m2) * pow(m1x, lm2));
266             }
267             theor_freq = A.length * p_k;
268             xi_2 += pow((frequencies[i] - theor_freq), 2) / theor_freq;
269         }
270         return xi_2;
271     }
```

```
272
273     public static double Kolmogorov_Fisher(double[] A, int l, int m) {
274         double D_n = 0;
275         double empiricalFRes, theoreticalFRes;
276         for (int i = 0; i < A.length; i++) {
277             empiricalFRes = empiricalFunction(A, A[i]);
278             theoreticalFRes = theorFisherFunction(A, A[i],l,m);
279             D_n = Math.max(D_n, Math.abs(empiricalFRes - theoreticalFRes));
280         }
281         return D_n;
282     }
283
284     public static double theorFisherFunction(double[] A, double x, int l, int m) {
285         double k = l*x/(l*x+m);
286         return I(k,l/2.,m/2.);
287     }
288
289
290     public static void main(String[] args) {
291         try (PrintStream res = new PrintStream("result.txt")) {
292             PrintStream out = new PrintStream("out.txt");
293             int n = 1000;
294             int mu = 0;
295             int s_2 = 64;
296             double[] A = normal(n, mu, s_2);
297             double mean_ = mean(A);
298             double sigma_2 = dispersion(A);
299             for (int i = 0; i < n; i++) out.println(A[i]);
300             res.println("Normal distribution: mu = " + mu + ", s_2 = " + s_2);
301             res.println("size = " + n);
302             res.println("mu = " + mu);
303             res.println("mean = " + mean_);
304             res.println("s_2 = " + s_2);
305             res.println("dispersion = " + sigma_2);
306             res.println("Kolmogorov test: " + Kolmogorov_normal(A, mu, s_2) + " < 1.36");
307             res.println("Pearson test: " + Pearson_normal(A) + " < 16.9\n\n");
308
309             PrintStream out1 = new PrintStream("out1.txt");
310             int n1 = 1000;
311             int m = 4;
312             long alpha = 65643;
313             double[] B = chi_2(n1, m, alpha);
314             for (int i = 0; i < n1; i++) out1.println(B[i]);
315             res.println("chi^2 distribution: m = " + m);
316             res.println("size = " + n1);
317             res.println("mu = " + m);
318             res.println("mean = " + mean(B));
319             res.println("s_2 = " + 2. * m);
320             res.println("dispersion = " + dispersion(B));
321             res.println("Kolmogorov test: " + Kolmogorov_chi_2(B,m) + " < 1.36");
322             res.println("Pearson test: " + Pearson_chi_2(B, m) + " < 16.9\n\n");
323
324             PrintStream out2 = new PrintStream("out2.txt");
325             int n2 = 1000;
326             int l = 3;
```

```
            int m_ = 5;
            double[] C = Fisher(n2, l, m_);
            for (int i = 0; i < n2; i++) out2.println(C[i]);
            res.println("Fisher's distribution: l = " + l + " m = " + m_);
            res.println("size = " + n2);
            res.println("mu = " + m_ / (m_ - 2));
            res.println("mean = " + mean(C));
            double s2 = (2. * m_ * m_ * (l + m_ - 2.)) /
                   (l * (m_ - 2.) * (m_ - 2.) * (m_ - 4.));
            res.println("s_2 = " + s2);
            res.println("dispersion = " + dispersion(C));
            res.println("Kolmogorov test: " + Kolmogorov_Fisher(C,l,m_) + " < 1.36");
            res.println("Pearson test: " + Pearson_Fisher(C, l, m_) + " < 16.9");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Построение гистограмм Matlab:



Код программы Matlab:

```
clc;
clear all;
X = fscanf(fopen('out.txt','r'), '%f');
subplot(3,1,1);
histogram(X)

Y = fscanf(fopen('out1.txt','r'), '%f');
subplot(3,1,2);
histogram(Y)

Z = fscanf(fopen('out2.txt','r'), '%f');
subplot(3,1,3);
histogram(Z)
```