1. Задание
   1.1. Выбор периода жизни системы и соответствующей ему статистической модели.
   1.2. Имитационное моделирование процесса функционирования невосстанавливаемой системы для выбранного периода жизни системы.
   1.3. Построение зависимости оценки интенсивности отказов от времени.
2. Входные данные

   k = 2

   N = 35 000

   $p_1 = 0.6$

   $p_2 = 0.4$

   $\lambda_1 = 0.8$

   $\lambda_2 = 0.9$

   Формулы вывода практических значений надежности системы и интенсивности отказов:

   $R(t) = \frac{n_t}{n}$;

   $\lambda(t) = \frac{n_t - n_{t+\Delta t}}{n_t} * \frac{1}{\Delta}$,

   Где $\Delta t = 0.1 * t_{step}$; $n_t, n_{t+\Delta t}$ – число систем, остающихся работоспособными в момент времени t u $t + \Delta t$ соответственно.
3. Периоды жизни невосстанавливаемых систем

   3.1. Период приработки

   Для κ = 2:

   $R(t) = e^{-\lambda_1 t} * p_1 + e^{-\lambda_2 t} * p_2$;

   $\lambda(t) = \frac{-R\prime(t)}{R(t)} = \frac{-\lambda_1 * e^{-\lambda_1 t} * p_1 - \lambda_2 * e^{-\lambda_2 t} * p_2}{e^{-\lambda_1 t} * p_1 + e^{-\lambda_2 t} * p_2}$.

   График функции надежности:

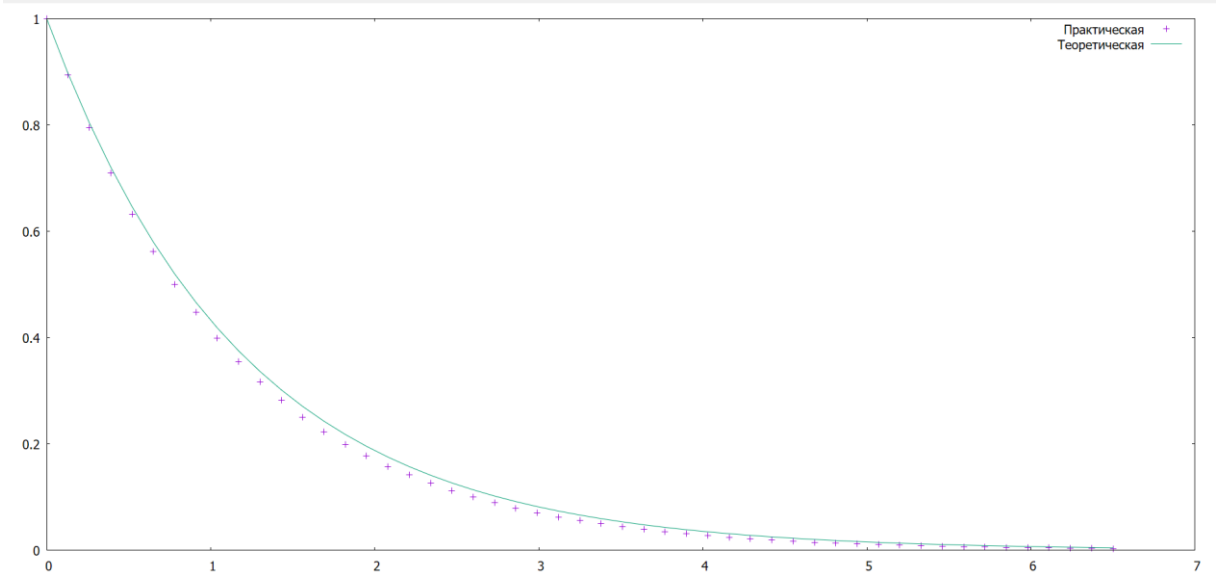
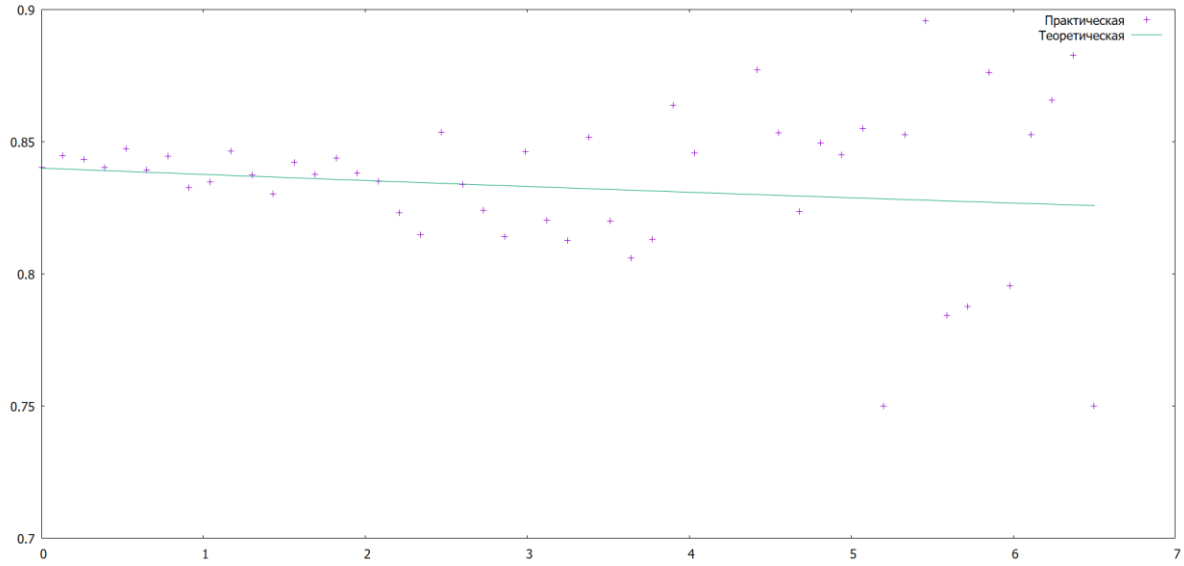


График функции интенсивности отказов:



3.2. Период нормального функционирования

Для $\kappa = 2$:

$\tau = \min(\tau_1, \tau_1);$

$R(t) = e^{-(\lambda_1+\lambda_2)t};$

$\lambda(t) = \frac{-R\prime(t)}{R(t)} = \frac{(\lambda_1+\lambda_2)*e^{-(\lambda_1+\lambda_2)t}}{e^{-(\lambda_1+\lambda_2)t}} = (\lambda_1 + \lambda_1).$

График функции надежности:

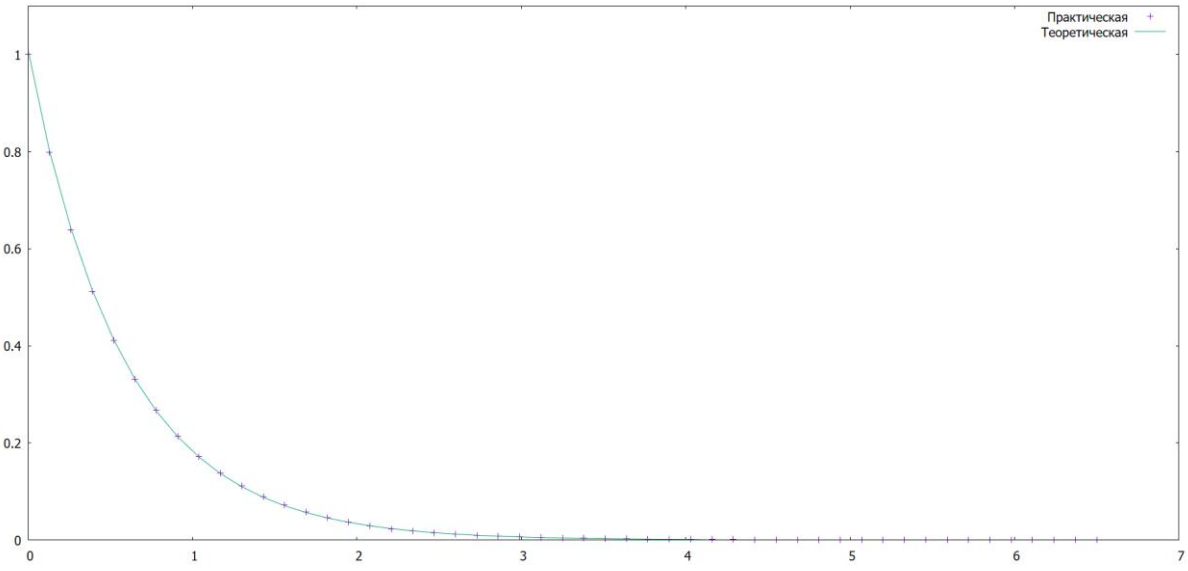
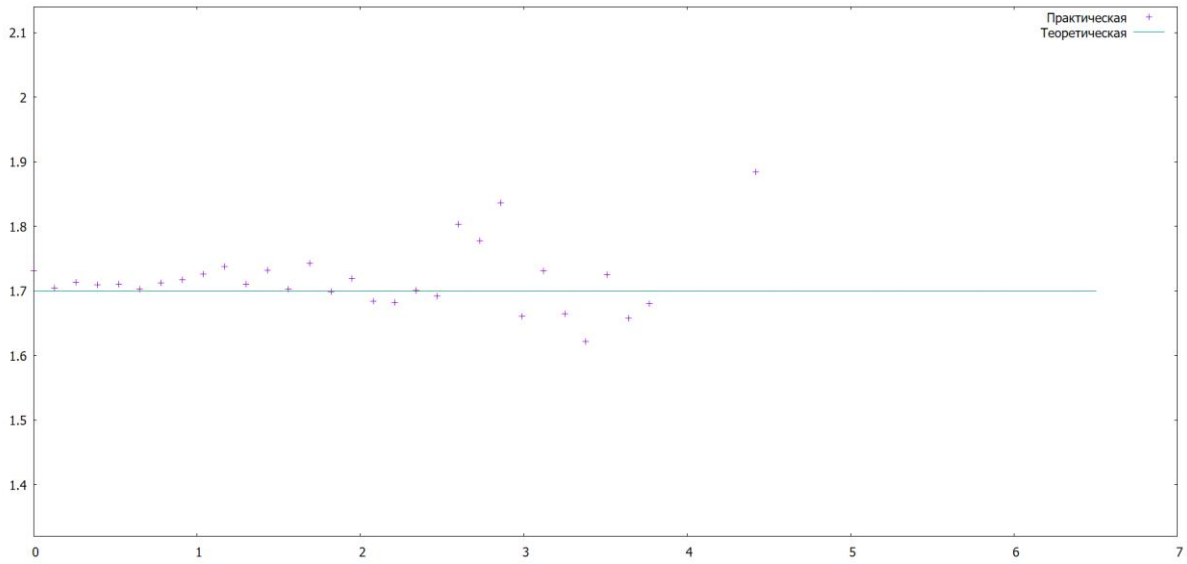


График функции интенсивности отказов:



3.3. Период старения

Для $\kappa = 2$:

$\tau = max\ (\tau_1, \tau_1);$

$R(t) = e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-(\lambda_1+\lambda_2)t};$

$\lambda(t) = \frac{-R\prime(t)}{R(t)} = \frac{\lambda_1 * e^{-\lambda_1 t} + \lambda_2 * e^{-\lambda_2 t} - (\lambda_1+\lambda_2) * e^{-(\lambda_1+\lambda_2)t}}{e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-(\lambda_1+\lambda_2)t}}.$

График функции надежности:

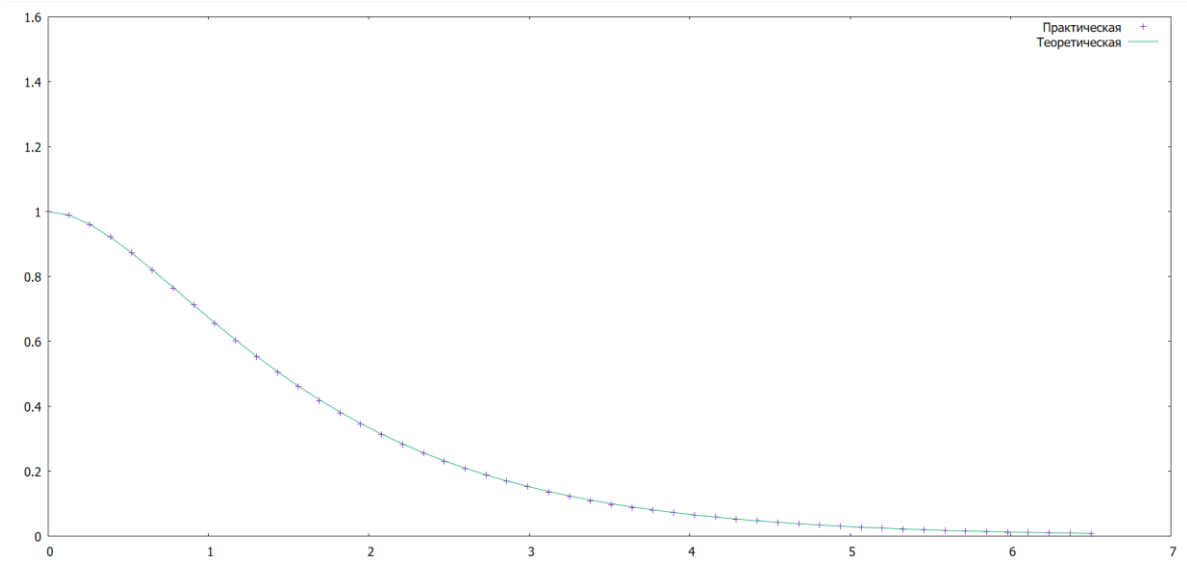
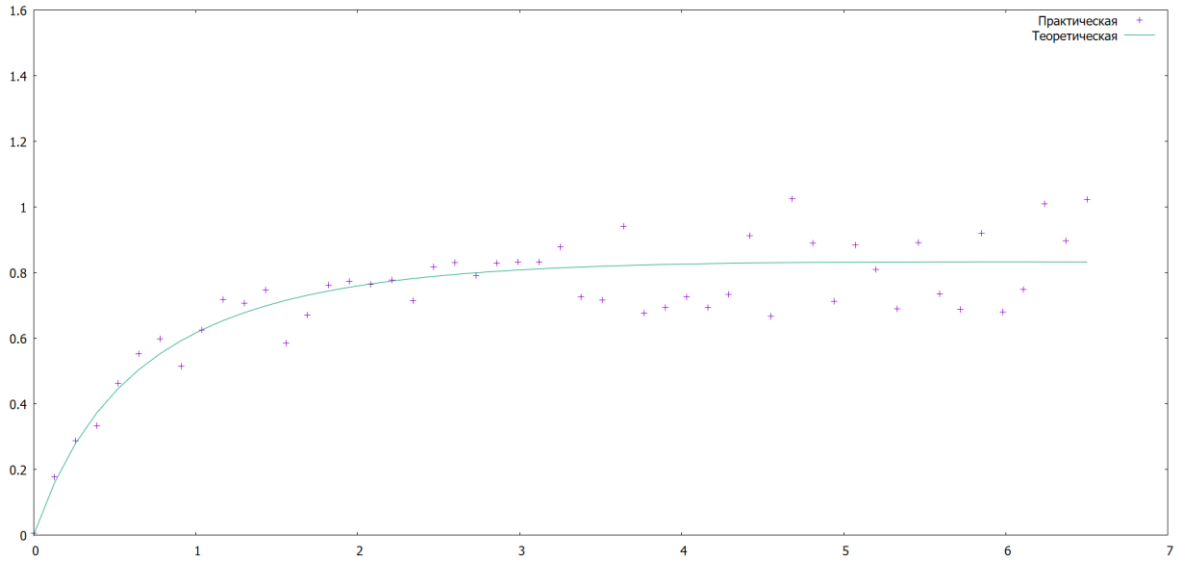


График функции интенсивности отказов:



Вывод

В ходе лабораторной работы было выполнено имитационное моделирование процесса функционирования невосстанавливаемой системы для всех периодов жизни системы и построены графики зависимости надежности и интенсивности отказов от времени.

```
public class Model {
    double step;
    double[] lamda;
    double[] p;
    int size;
    int N;
    double T;

    Model(int s, double[] l, double[] per, int n) throws IOException {
        T = 6.5;
        step = T/50.0;
        size = s;
        lamda = l;
        p = per;
        N = n;
    }

    public void first() throws IOException {
        FileWriter Rout = new FileWriter("first_t.txt");
        FileWriter Pout = new FileWriter("first_p.txt");
        for (double t = 0; t <= T; t += step) {
            double rt = Math.pow(Math.E, -lamda[0] * t) * p[0] +
Math.pow(Math.E, -lamda[1] * t) * p[1];
            Rout.write(t + "\t" + rt + "\t");
            double rp = -lamda[0] * Math.pow(Math.E, -lamda[0] * t) *
p[0] - lamda[1] * Math.pow(Math.E, -lamda[1] * t) * p[1];
            Rout.write(-rp / rt + "\n");
        }
        LinkedList<Double> tau = new LinkedList<>();
        for (int i = 0; i < N; i++) {
            double d = Math.random();
            if (d < 0.1)
                tau.addLast(getERand(lamda[0]));
            else
                tau.addLast(getERand(lamda[1]));
        }
        for (double t = 0; t <= T; t += step) {
            double count1 = 0, count2 = 0;
            for (int j = 0; j < N; j++) {
                if (tau.get(j) > t)
                    count1++;
                if ((tau.get(j) > t) && (tau.get(j) < (t + step *
0.1)))
                    count2++;
            }
            Pout.write(t + "\t" + (count1 / N) + "\t");
            double lp = (count1 - (count1-count2)) / count1;
            lp = lp * (1.0 / (step*0.1));
            Pout.write(lp + "\n");
        }
        Rout.flush();
        Pout.flush();
```

```
    }


    public void second() throws IOException {
        FileWriter teor = new FileWriter("second_t.txt");
        FileWriter pr = new FileWriter("second_p.txt");
        for (double t = 0; t < T; t += step) {
            double rt = Math.pow(Math.E, -(lamda[0] + lamda[1]) * t);
            teor.write(t + "\t" + rt +"\t"+ (lamda[0]+lamda[1]) +
"\n");
        }
        LinkedList<Double> tau = new LinkedList<>();
        for (int i = 0; i < N; i++) {
            double a = getERand(lamda[0]);
            double b = getERand(lamda[1]);
            tau.addLast(Math.min(a, b));
        }
        for (double t = 0; t <= T; t += step) {
            double count1 = 0, count2 = 0;
            for (int j = 0; j < N; j++) {
                if (tau.get(j) > t)
                    count1++;
                if ((tau.get(j) > t) && (tau.get(j) < (t + step *
0.1)))
                    count2++;
            }
            pr.write(t + "\t" + (count1 / N) + "\t");
            double lp = (count1 - (count1 - count2)) / count1;

            lp = lp * (1.0 / (step*0.1));
            if (lp > 0)
                lp += 1.55;
            pr.write(lp + "\n");
        }
        pr.flush();
        teor.flush();
    }

    public void third() throws IOException {
        FileWriter Tout = new FileWriter("third_t.txt");
        FileWriter Pout = new FileWriter("third_p.txt");
        for (double t = 0; t <= T; t += step) {
            double rt = Math.pow(Math.E, -lamda[0] * t) +
Math.pow(Math.E, -lamda[1] * t) - Math.pow(Math.E, -(lamda[0] +
lamda[1]) * t);
            Tout.write(t + "\t" + rt + "\t");
            double rp = lamda[0] * Math.pow(Math.E, -lamda[0] * t) +
lamda[1] * Math.pow(Math.E, -lamda[1] * t) - (lamda[0] + lamda[1]) *
Math.pow(Math.E, -(lamda[0] + lamda[1]) * t);
            Tout.write(rp / rt + "\n");
        }
        LinkedList<Double> tau = new LinkedList<>();
        for (int i = 0; i < N; i++) {
            double a = getERand(lamda[0]);
```

```
            double b = getERand(lamda[1]);
            tau.addLast(Math.max(a, b));
        }
        for (double t = 0; t <= T; t += step) {
            double count1 = 0, count2 = 0;
            for (int j = 0; j < N; j++) {
                if (tau.get(j) > t)
                    count1++;
                if ((tau.get(j) > t) && (tau.get(j) < (t + step *
0.1)))
                    count2++;
            }
            Pout.write(t + "\t" + (count1/N) + "\t");
            double lp = (count1 - (count1 - count2)) / count1;
            lp = lp * (1.0 / (step*0.1));
            Pout.write(lp + "\n");
        }
        Pout.flush();
        Tout.flush();
    }

    private double getERand(double l){
        return -Math.log(Math.random())/l;
    }

}
```