Лабораторная работа №3
ИССЛЕДОВАНИЕ ИНТЕНСИВНОСТИ ОТКАЗОВ ДЛЯ
НЕВОССТАНАВЛИВАЕМЫХ СИСТЕМ

1. Цель работы

Провести имитационное моделирование процесса функционирования невосстанавливаемой системы для всех периодов жизни системы. Построить зависимости $\hat{\lambda}(t)$ оценки интенсивности отказов от времени.

2. Ход выполнения работы

Весь период эксплуатации объекта, начиная от ввода и заканчивая достижением предельного состояния, можно условно разделить на три периода:

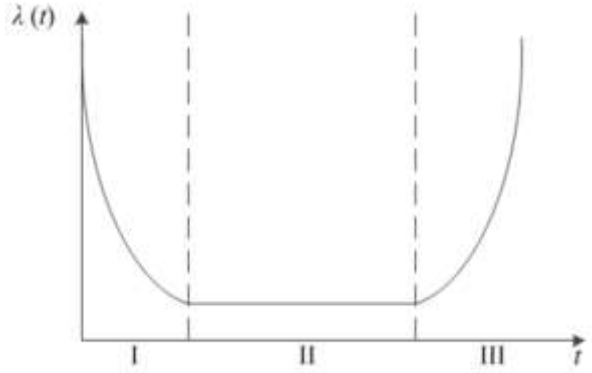

Рисунок 1 – Периоды жизни объекта

2.1. Первый период жизни (приработка)

Множество исследуемых экземпляров систем разбивается на k подмножеств. Попадание экземпляра во множество j характеризуется вероятностью p_j. При этом выполняется следующее равенство: $\sum_1^j p_j = 1$. Подмножество j характеризуется своей интенсивностью отказов λ_j. Предполагается, что каждый экземпляр системы состоит из единственного элемента и в течение всего времени функционирования системы интенсивность отказов λ_j остается величиной постоянной.

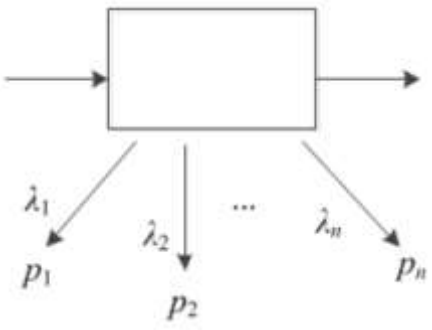

Рисунок 2 – Период приработки

<u>В качестве исходных данных для моделирования возьмем</u>:

Выборку из n = 30000 систем, количество подмножеств k = 2, вероятность попадания в первое подмножество $p_1 = 0.6$, вероятность попадания во второе подмножество $p_2 = 0.4$, интенсивность отказов первого подмножества $\lambda_1 = 0.9$, интенсивность отказов второго подмножества $\lambda_2 = 1.1$.

Для i-ой системы первоначально надо определить, к какому подмножеству она относится. Для этого используется распределение p_j. После того, как номер j помножества определен, необходимо сгенерировать значение τ_i, как случайной величины, распределенной по экспоненциальному закону с параметром λ_j, который определяется по номеру подмножества.

Функция надежности при процессе моделирования определяется как:

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

Теоретически она задается как:

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

Полученные графики:

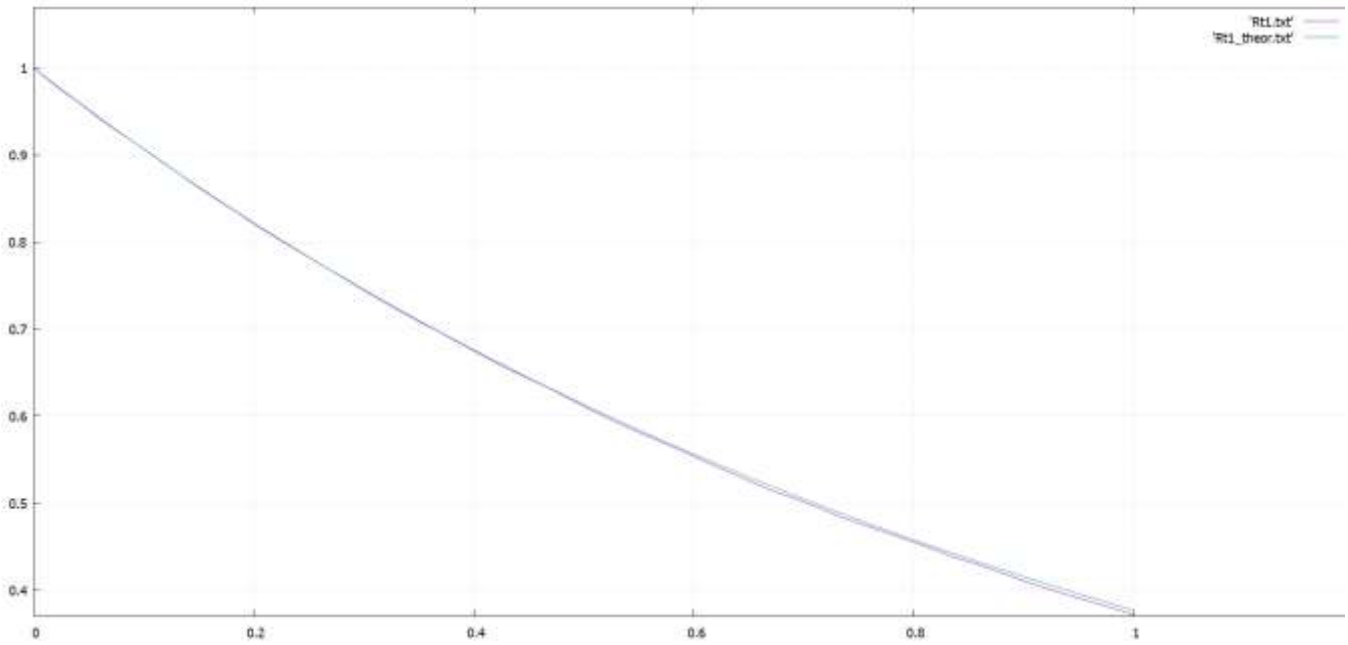

Рисунок 3 – Функция надёжности для периода приработки

По графику видно, что функция надежности – убывающая.

Оценка интенсивности отказов в точке t, определяется по следующей формуле:

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

где $\Delta t = 0.1 \cdot t_{\text{step}}$, n_t и $n_{t+\Delta t}$ – это число систем, остающихся работоспособными в момент времени t и $t + \Delta$ соответственно.

Теоретически интенсивности отказов определяется через функцию надежности $R(t)$:

$$\lambda_{\text{теор}}(t) = \frac{-R'(t)}{R(t)} = -\frac{-\lambda_1 \cdot e^{-\lambda_1 t} \cdot p_1 - \lambda_2 \cdot e^{-\lambda_2 t} \cdot p_2}{e^{-\lambda_1 t} \cdot p_1 + e^{-\lambda_2 t} \cdot p_2}$$

Полученные графики:

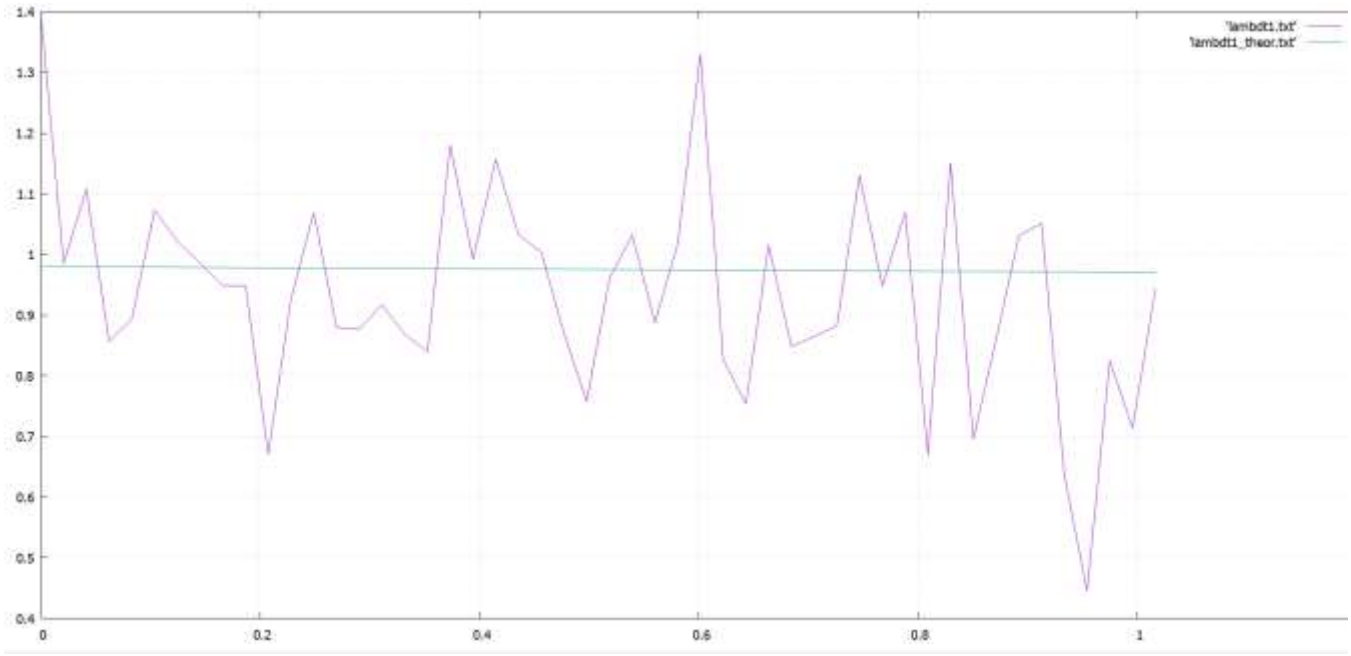

Рисунок 4 – Функция интенсивности отказов для периода приработки

По графику видно, что функция изменения интенсивности отказов в первый период жизни – убывающая.

2.2. Второй период жизни (нормальное функционирование)

В качестве модели системы используется схема с последовательным соединением n элементов. Элемент i характеризуется интенсивностью отказов λ_i, которая является постоянной величиной.

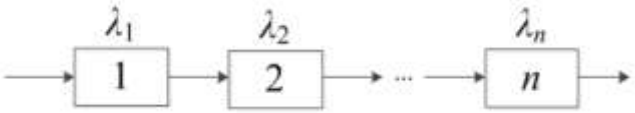

Рисунок 5 – Схема с последовательным соединением

В качестве исходных данных для моделирования возьмем:

Выборку из n = 30000 систем, количество элементов в системе n = 2, интенсивность отказов первого подмножества $\lambda_1 = 0.9$, интенсивность отказов второго подмножества $\lambda_2 = 1.1$.

Для j-ого элемента системы вычисляется время безотказной работы t_j, как случайной величины, распределенной по экспоненциальному закону с параметром λ_j. Значение времени безотказной работ i-ой системы вычисляется по следующей формуле: $\tau = \min_j(\tau_j)$.

Функция надежности при процессе моделирования определяется как:

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

Теоретически она задается как:

$$R(t) = R_1(t) + R_2(t) = e^{-(\lambda_1+\lambda_2)t}$$

Полученные графики:

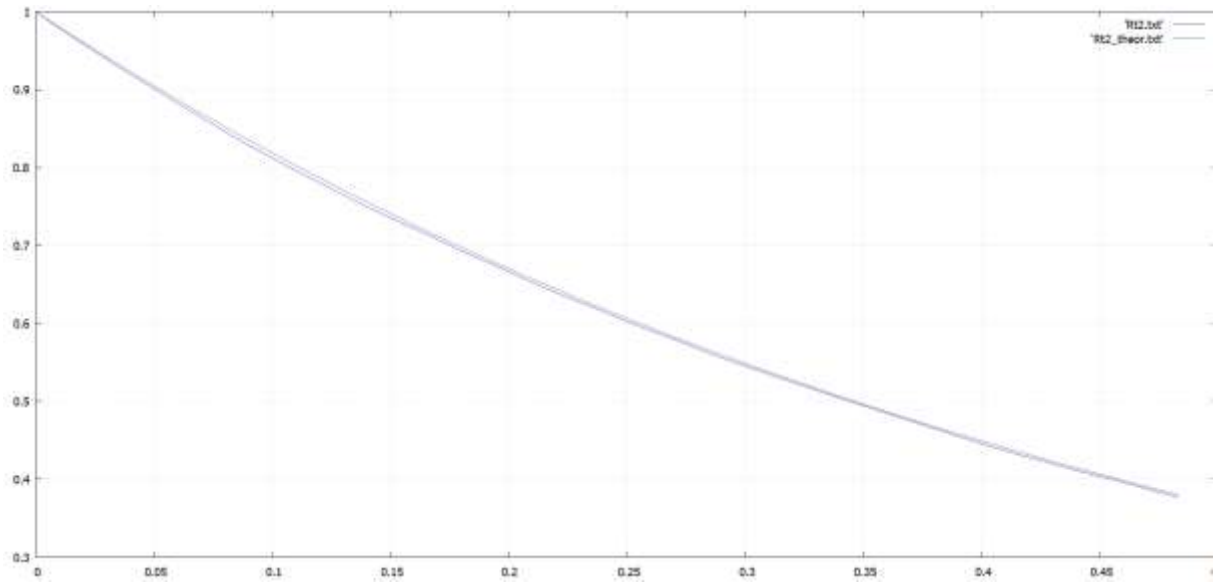

Рисунок 6 – Функция надёжности для периода нормального функционирования

По графику можно сделать вывод о том, что функция надежности – невозрастающая.

Оценка интенсивности отказов в точке t, определяется по следующей формуле:

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

где $\Delta t = 0.1 \cdot t_{\text{step}}$, n_t и $n_{t+\Delta t}$ – это число систем, остающихся работоспособными в момент времени t и $t + \Delta$ соответственно.

Теоретически интенсивности отказов определяется через функцию надежности $R(t)$:

$$\lambda_{\text{теор}}(t) = \frac{-R'(t)}{R(t)} = -\frac{-(\lambda_1 + \lambda_2) \cdot e^{-(\lambda_1+\lambda_2)t}}{e^{-(\lambda_1+\lambda_2)t}} = \lambda_1 + \lambda_2 = const$$

Полученные графики:

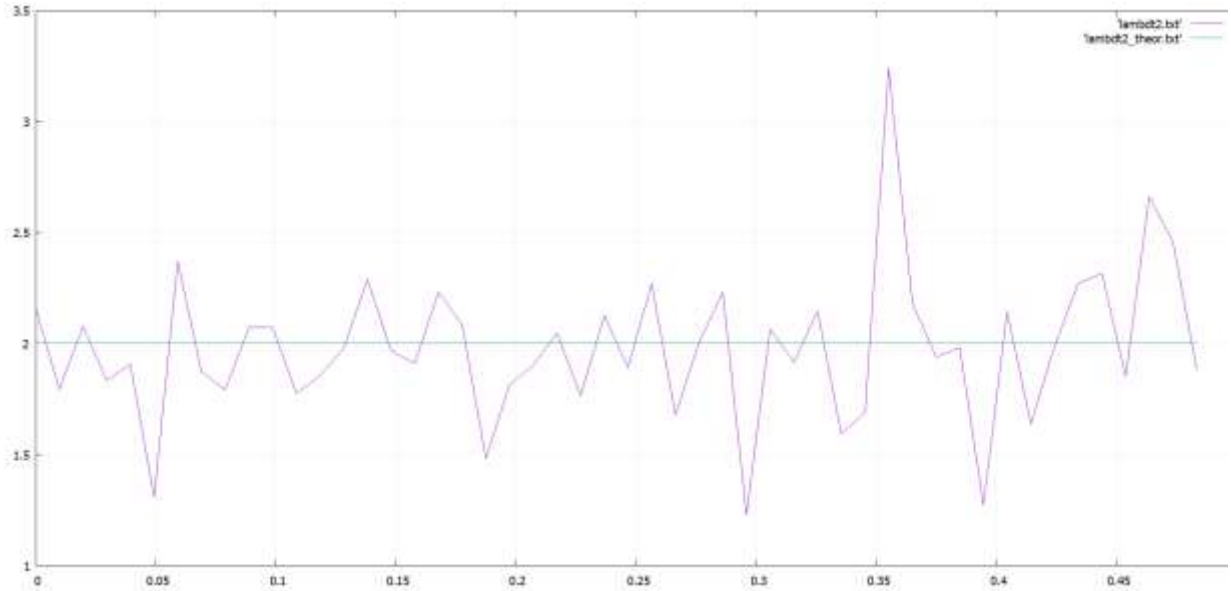

Рисунок 7 – Функция интенсивности отказов для периода нормального функционирования

По графику видно, что функция изменения интенсивности отказов во второй период жизни является константой и равна $\lambda(t) = \lambda_1 + \lambda_2 = 2$.

2.3. Третий период жизни (период старения)

Элемент i характеризуется интенсивностью отказов λ_i, которая является постоянной величиной. Значение n задается преподавателем индивидуально для каждого студента в бригаде.

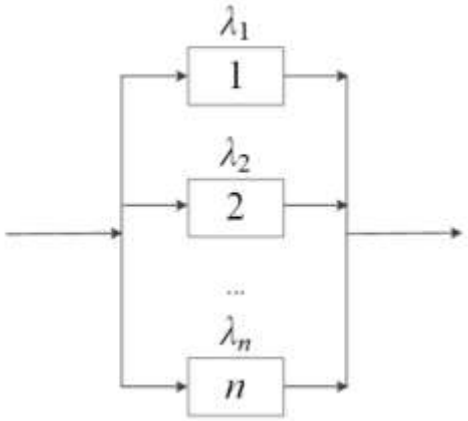

Рисунок 8 – Модель периода старения с параллельным соединением

В качестве исходных данных для моделирования возьмем:

Выборку из n = 30000 систем, количество элементов в системе n = 2, интенсивность отказов первого подмножества $\lambda_1 = 0.9$, интенсивность отказов второго подмножества $\lambda_2 = 1.1$.

Время безотказной работы для j-ого элемента системы t_j, вычисляется как случайная величина, распределенная по экспоненциальному закону с параметром λ_j. Для схемы с параллельным соединением n элементов значение времени безотказной работы i-ой системы вычисляется по следующей формуле:
$\tau = \max_j(\tau_j)$.

Функция надежности при процессе моделирования определяется как:

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

Теоретически она задается как:

$$R(t) = R_1(t) + R_2(t) - R_1(t)R_2(t) = e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-\lambda_1 t} \cdot e^{-\lambda_2 t}$$

Полученные графики:

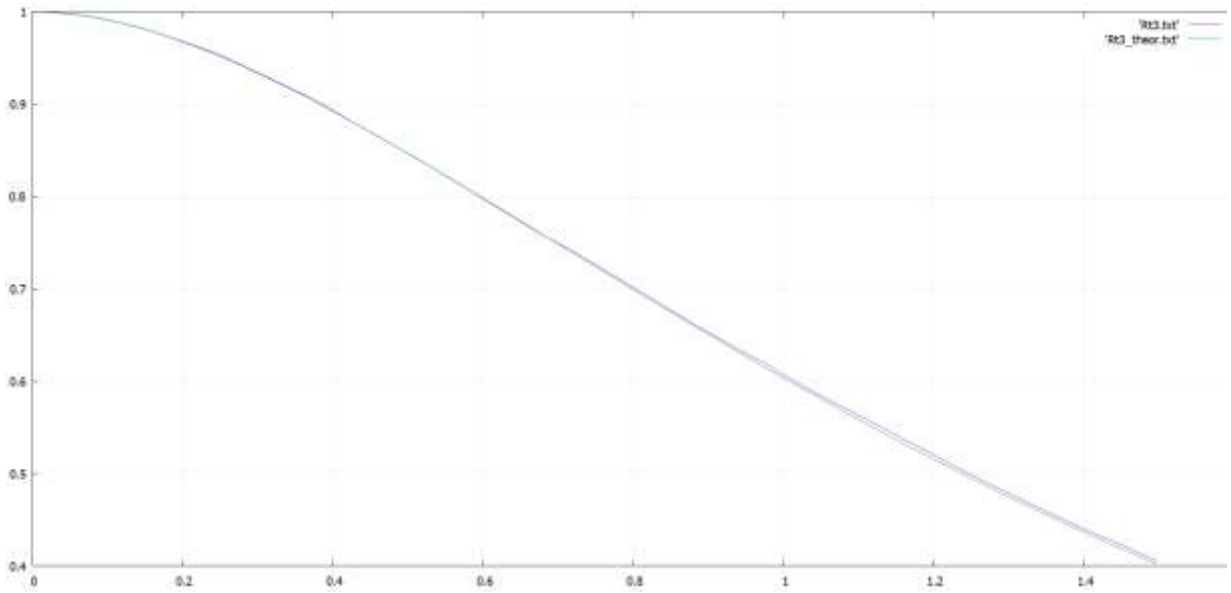

Рисунок 9 – Функция надежности для периода старения

По графику можно сделать вывод о том, что функция надежности – убывающая.

Оценка интенсивности отказов в точке t, определяется по следующей формуле:

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

где $\Delta t = 0.1 \cdot t_{\text{step}}$, n_t и $n_{t+\Delta t}$ – это число систем, остающихся работоспособными в момент времени t и $t + \Delta$ соответственно.

Теоретически интенсивности отказов определяется через функцию надежности $R(t)$:

$$\lambda_{\text{теор}}(t) = \frac{-R'(t)}{R(t)} = -\frac{-\lambda_1 \cdot e^{-\lambda_1 t} - \lambda_2 \cdot e^{-\lambda_2 t} + e^{-(\lambda_1+\lambda_2)t} \cdot (\lambda_1 + \lambda_2)}{e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-(\lambda_1+\lambda_2)t}}$$

Полученные графики:

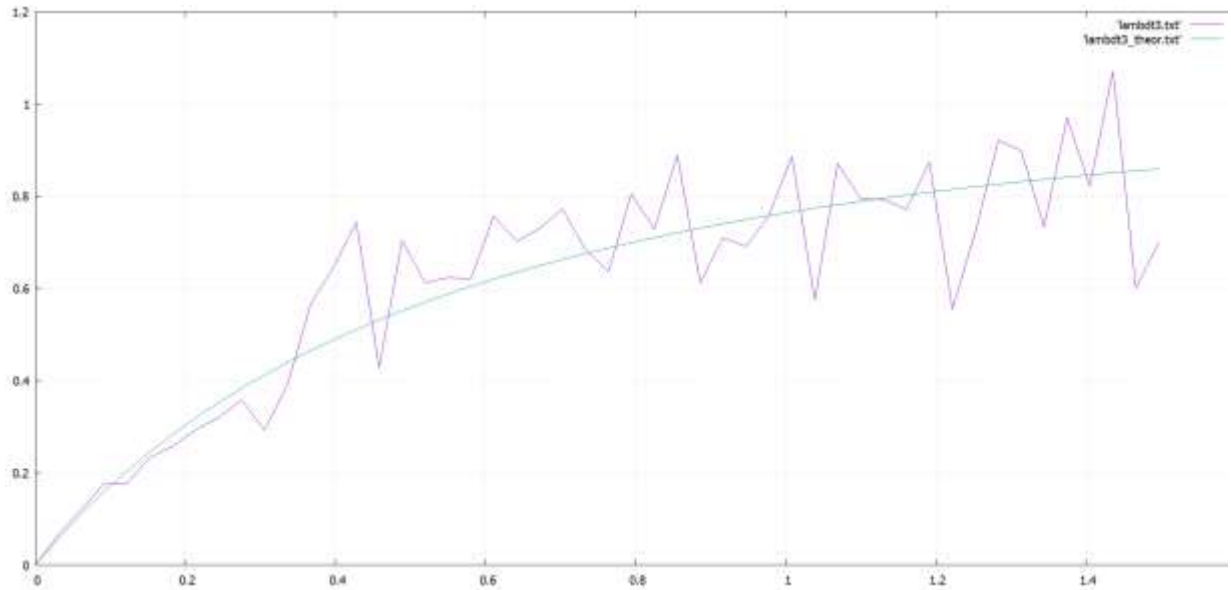

Рисунок 10 – Функция интенсивности отказов для периода старения

По графику видно, что функция изменения интенсивности отказов в третий период жизни является возрастающей, так как со временем число систем, вышедших из строя, увеличивается.

3. Выводы

В ходе выполнения лабораторной работы, мы произвели имитационное моделирование процесса функционирования невосстанавливаемой системы для трех периодов жизни. Построили зависимости функции надежности и функции интенсивности от времени.

По графикам можно сделать вывод о том, что на первом этапе приработки интенсивность отказа постепенно уменьшается, по причине того, что в начале многие системы выходят из строя по различным причинам. На втором этапе интенсивность выхода систем из строя приходит к постоянному значению и начинается период нормального функционирования. На третьем этапе интенсивность отказов увеличивается, так как наступает предельное состояние работы основной части систем.

4. Листинг программы

```
ыpublic class Main {

    public static void main(String[] args) {
        int n = 30000;
        double lambd1 = 0.9;
        double lambd2 = 1.5;
        double p1 = 0.6;
        double p2 = 1 - p1;
        ArrayList <Double> Ti1 = new ArrayList<>();
        ArrayList <Double> Ti2 = new ArrayList<>();
        ArrayList <Double> Ti3 = new ArrayList<>();

        for (int i = 0; i < (int) (p1 * n); i++) {
            Ti1.add(-log(random()) / lambd1);
        }
        for (int i = 0; i < n - (int) (p1 * n); i++) {
            Ti1.add(-log(random()) / lambd2);
        }
        firstPeriod(Ti1, n, lambd1, lambd2, p1);

        for (int i = 0; i < n; i++) {
            double T1 = -log(random()) / lambd1;
            double  T2 = -log(random()) / lambd2;
            Ti2.add(min(T1,T2));
        }
        secondPeriod(Ti2, n, lambd1, lambd2);

        for (int i = 0; i < n; i++) {
            double T1 = -log(random()) / lambd1;
            double  T2 = -log(random()) / lambd2;
            Ti3.add(max(T1,T2));
        }
        thirdPeriod(Ti3, n, lambd1, lambd2);
    }

    public static void firstPeriod (ArrayList <Double> Ti1, int n, double
lambd1, double lambd2, double p1) {
        ArrayList <Double> Rt = new ArrayList<>(), RtT = new ArrayList<>();
        ArrayList <Double> lambda = new ArrayList<>(), lambdaT = new Ar-
rayList<>();
```

```
        double p2 = 1 - p1;

        double Tm = countTm(Ti1, n);
        double tStep = Tm/50;

        for (double i = 0; i < Tm; i += tStep) {
            int nt = countNt(Ti1,i);
            int ntdelta = countNt(Ti1,i + tStep * 0.1);
            lambda.add((nt-ntdelta)/(nt * tStep * 0.1));
            Rt.add ( (double) nt/n);
            RtT.add (exp(-lambd1 * i) * p1 + exp(-lambd2 * i) * p2);
            lambdaT.add(-(-lambd1 * exp(-lambd1 * i) * p1 - lambd2 * exp(-
lambd2 * i) * p2)/(exp(-lambd1 * i) * p1 + exp(-lambd2 * i) * p2));
        }
        save(lambda,"lambdt1.txt", tStep);
        save(Rt, "Rt1.txt",tStep);
        save(RtT, "Rt1_theor.txt",tStep);
        save(lambdaT, "lambdt1_theor.txt",tStep);
    }

    public static void secondPeriod (ArrayList <Double> Ti2, int n, double
lambd1, double lambd2) {
        ArrayList <Double> Rt = new ArrayList<>(), RtT = new ArrayList<>();
        ArrayList <Double> lambda = new ArrayList<>(), lambdaT = new Ar-
rayList<>();

        double Tm = countTm(Ti2, n);
        double tStep = Tm/50;

        for (double i = 0; i < Tm; i += tStep) {
            int nt = countNt(Ti2,i);
            int ntdelta = countNt(Ti2,i + tStep * 0.1);
            lambda.add((nt-ntdelta)/(nt * tStep * 0.1));
            Rt.add ( (double) nt/n);
            RtT.add (exp(i * (-lambd1 - lambd2)));
            lambdaT.add(lambd1 + lambd2);
        }
        save(lambda,"lambdt2.txt", tStep);
        save(Rt, "Rt2.txt",tStep);
        save(RtT, "Rt2_theor.txt",tStep);
        save(lambdaT, "lambdt2_theor.txt",tStep);
    }

    public static void thirdPeriod (ArrayList <Double> Ti3, int n, double
lambd1, double lambd2) {
        ArrayList <Double> Rt = new ArrayList<>(), RtT = new ArrayList<>();
        ArrayList <Double> lambda = new ArrayList<>(), lambdaT = new Ar-
rayList<>();

        double Tm = countTm(Ti3, n);
        double tStep = Tm/50;

        for (double i = 0; i < Tm; i += tStep) {
            int nt = countNt(Ti3, i);
            int ntdelta = countNt(Ti3, i + tStep * 0.1);
            lambda.add((nt - ntdelta) / (nt * tStep * 0.1));
            Rt.add((double) nt / n);
```

```
            RtT.add(exp(-lambd1 * i) + exp(-lambd2 * i) - exp(-lambd1 * i) *
exp(-lambd2 * i));
            lambdaT.add(-(-lambd1 * exp(-lambd1 * i) - lambd2 * exp(-lambd2
* i) + exp(-lambd1 * i - lambd2 * i) * (lambd1 + lambd2)) / (exp(-lambd1 * i)
+ exp(-lambd2 * i) - exp(-lambd1 * i) * exp(-lambd2 * i)));
        }
        save(lambda, "lambdt3.txt", tStep);
        save(Rt, "Rt3.txt", tStep);
        save(RtT, "Rt3_theor.txt", tStep);
        save(lambdaT, "lambdt3_theor.txt", tStep);
    }

    public static double countTm(ArrayList <Double> Ti, int n){
        double Tm = 0;
        for (int i = 0; i < n; i++) {
            Tm += Ti.get(i);
        }
        Tm = Tm / n;
        return Tm;
    }

    public static int countNt (ArrayList<Double> Ti, double t) {
        int count = 0;
        for (double i = 0; i < Ti.size(); i++) {
            if (t < Ti.get( (int) i)) {
                count++;
            }
        }
        return count;
    }
```