1. *Цель работы*

Целью данной работы является имитационное моделирование и теоретический расчёт коэффициента готовности восстанавливаемой системы. Работа делается в несколько этапов:

1) Моделирование и теоретический расчёт коэффициентов готовности для простых восстанавливаемых систем.
2) Моделирование и расчёт коэффициента готовности для сложной системы. Расчёт верхней границы коэффициента готовности и нижней границы двумя способами.

2. *Исходные данные*

На рисунке 1 представлена сложная схема, которая имеет 4 элемента и 2 ремонтные бригады:

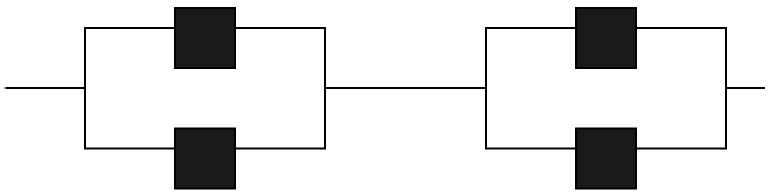

Рисунок 1 – Схема исходной сложной системы

Теоретическая система, изображенная на Рисунке 1 можно разбить на две простых двухэлементные системы.

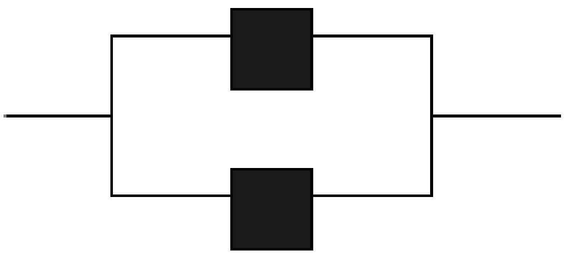

Рисунок 2 - Двухэлементная система

3. *Теоретические сведения*

Теоретические формулы для вычисления коэффициента готовности для простых систем:

1) Формула коэффициента готовности для системы с одним элементом и одной ремонтной бригадой.

$$K_{1,1} = \frac{\mu}{\lambda + \mu},$$

μ – интенсивность восстановления системы

λ – интенсивность отказов системы.

2) Формула коэффициента готовности для системы с двумя элементами и одной ремонтной бригадой.

$$K_{2,1} = \frac{2\mu\lambda + \mu^2}{2\lambda^2 + 2\lambda\mu + \mu^2}.$$

3) Формула коэффициента готовности для системы с двумя элементами и двумя ремонтными бригадами. Рассматривается как две схемы с одним элементом и одной ремонтной бригадой.

$$K_{2,2} = 1 - \left(1 - K_{1,1}\right)^2.$$

Формула для экспериментальной оценки коэффициента готовности системы

$$K_{model} = \frac{n_t}{N},$$

n_t – количество систем, работающих в момент времени t

N – число экспериментов (систем)

4. *Результаты выполнения работы*

Приведем результаты моделирования для простых систем:

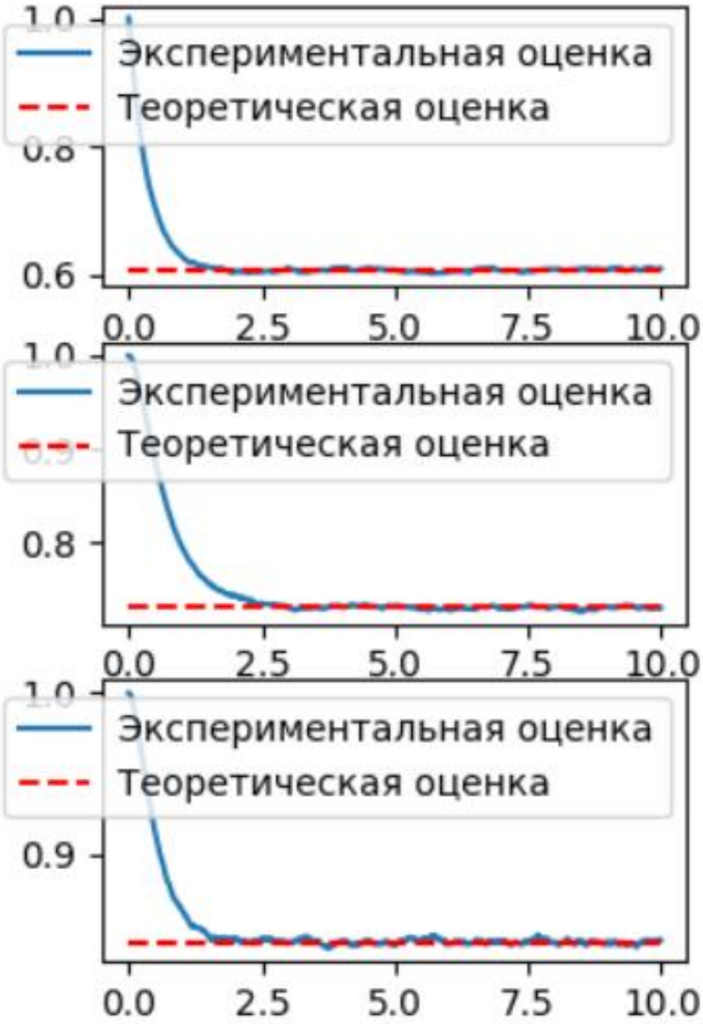

Рис.3 – Графики коэффициента готовности для (сверху вниз) системы с одним элементом и одной ремонтной бригадой; с двумя элементами и одной ремонтной бригадой; с двумя элементами и двумя ремонтными бригадами

Моделирование сложной системы с 4 элементами и 2 ремонтными бригадами

Моделирование проводилось при $N = 60000$, $\lambda = 1.1$, $\mu = 1.7$.

Результаты моделирования коэффициента готовности и сравнение с вычисленными верхней и, полученными двумя способами, нижними границами для сложной системы.

Для вычисления верхней границы делается предположение, что количество ремонтных бригад становится равным количеству элементов системы, то есть в нашем случае количество ремонтных бригад увеличится с 2 до 4. Верхняя граница будет вычисляться следующим образом:

$$K_{up} = K_{2,2} \cdot K_{2,2}.$$

Нижняя граница вычисляется двумя способами.

1) Первый способ заключается в предположении, что ремонтные бригады распределены по группам элементов и не выходят за их пределы:

$$K_{low_1} = K_{2,1} \cdot K_{2,1}$$

2) Второй способ заключается в предположении, что количество элементов системы становится равным количеству ремонтных бригад, при условии, что удаляются элементы только из параллельного соединения:

$$K_{low_2} = K_{1,1} \cdot K_{1,1}$$

На рисунке 4 представлен график коэффициента готовности для сложной восстанавливаемой системы.

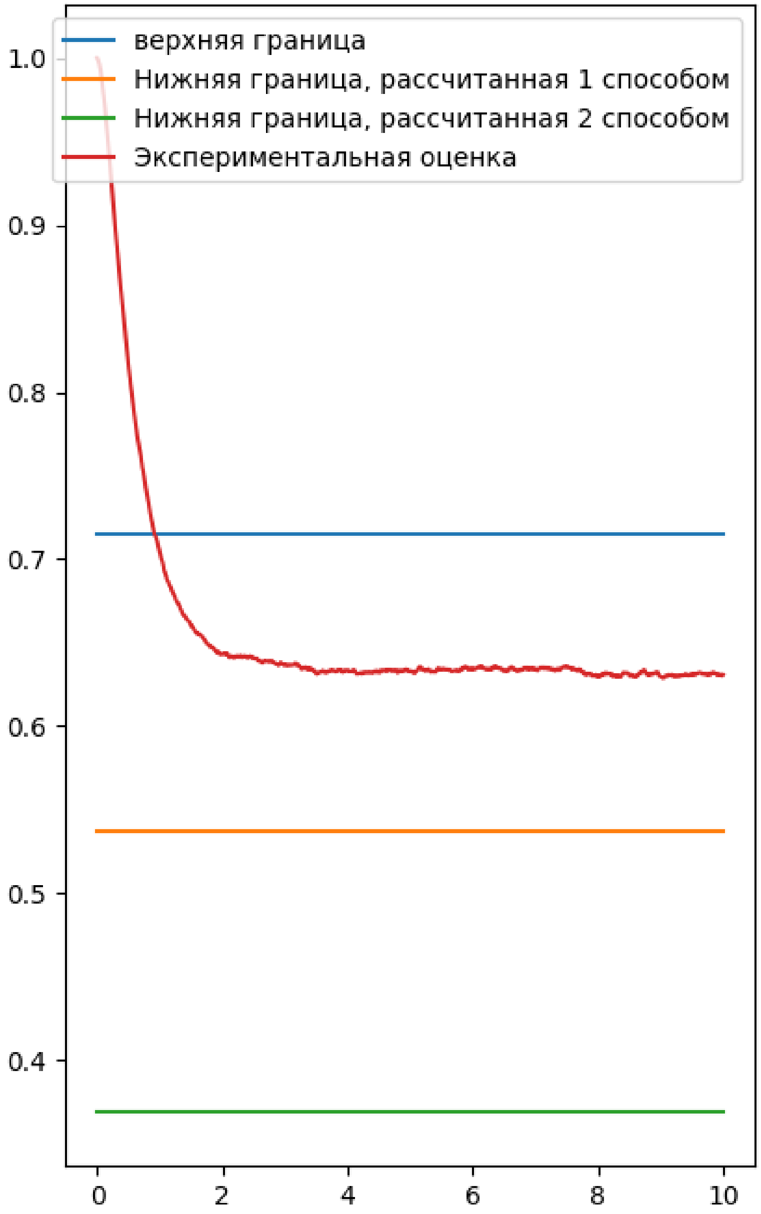

Рисунок 4 – Графики зависимости коэффициента готовности от времени для сложной системы

5. *Вывод*

В результате проделанной работы были смоделированы и теоретически посчитаны оценки коэффициентов готовности для простых восстанавливаемых систем. Также была смоделирована работа сложной восстанавливаемой системы, и результаты моделирования были сравнены с рассчитанными теоретическими значениями верхней границы и, полученными двумя способами, нижними границами коэффициента готовности системы.

6. *Листинг*

```
import numpy as np
from matplotlib import pyplot as plt
from random import random


N = 60000
n_repairs = 150
L = 1.1
M = 1.7
delta_t = 0.01
t = np.arange(0, 10, delta_t)


def model_11():
    T_work = [-np.log(random()) / L for i in range(n_repairs)]
    T_repair = [-np.log(random()) / M for j in range(n_repairs)]

    n_t = []
    T = T_work[0]
    is_on = 1
    step = 0
    for ti in t:
        if ti > T:
            if is_on == 1:
                is_on = 0
                T += T_repair[step]
                step += 1
            else:
                is_on = 1
                T += T_work[step]
        n_t.append(is_on)

    return n_t


def model_21():
    free_rem = 1
    T = [-np.log(random()) / L for i in range(2)]
    working_on = [0, 0]
    is_on = [1, 1]
    queue = []
    n_t = [0] * len(t)
    for step in range(len(t)):
        remove_from_queue = []
        for i in range(len(T)):
            if T[i] < t[step]:
                is_on[i] = 0
                if i not in queue:
                    if working_on[i] == 0:
                        queue.append(i)
                    else:
                        free_rem += 1
                        T[i] += -np.log(random()) / L
                        working_on[i] = 0
                        is_on[i] = 1
        for i in range(len(T)):
            if T[i] < t[step]:
                if i in queue:
                    if queue.index(i) < free_rem:
                        T[i] = t[step] -np.log(random()) / M
                        working_on[i] = 1
                        remove_from_queue.append(i)
```

```
        free_rem -= len(remove_from_queue)
        for rem in remove_from_queue:
            queue.remove(rem)
        n_t[step] = 1 if (is_on[0] == 1 or is_on[1] == 1) else 0

    return n_t


def model_22():
    T_work = []
    T_repair = []
    for i in range(2):
        T_work.append([])
        T_repair.append([])
        for j in range(n_repairs):
            T_work[i].append(-np.log(random()) / L)
            T_repair[i].append(-np.log(random()) / M)
    n_t = []
    T1 = T_work[0][0]
    T2 = T_work[1][0]
    T1_is_on = 1
    T2_is_on = 1
    i1 = 0
    i2 = 0
    for ti in t:
        if ti > T1:
            if T1_is_on == 1:
                T1_is_on = 0
                T1 += T_repair[0][i1]
                i1 += 1
            else:
                T1_is_on = 1
                T1 += T_work[0][i1]
        if ti > T2:
            if T2_is_on == 1:
                T2_is_on = 0
                T2 += T_repair[1][i2]
                i2 += 1
            else:
                T2_is_on = 1
                T2 += T_work[1][i2]
        n_t.append(T1_is_on + T2_is_on)

    return [1 if i > 0 else 0 for i in n_t]


def model_42():
    free_rem = 2
    T = [-np.log(random()) / L for i in range(4)]
    working_on = [0, 0, 0, 0]
    is_on = [1, 1, 1, 1]
    queue = []
    n_t = [0]*len(t)
    for step in range(len(t)):
        remove_from_queue = []
        for i in range(len(T)):
            if T[i] < t[step]:
                is_on[i] = 0
                if i not in queue:
                    if working_on[i] == 0:
                        queue.append(i)
                    else:
                        free_rem += 1
                        T[i] += -np.log(random()) / L
```

```
                        working_on[i] = 0
                        is_on[i] = 1
                if i in queue:
                    if queue.index(i) < free_rem:
                        T[i] = t[step] -np.log(random()) / M
                        working_on[i] = 1
                        remove_from_queue.append(i)
        free_rem -= len(remove_from_queue)
        for rem in remove_from_queue:
            queue.remove(rem)
        n_t[step] = 1 if (is_on[0] == 1 or is_on[1] == 1) and  (is_on[2]
== 1 or is_on[3] == 1) else 0
    return n_t


if __name__ == '__main__':
    K_11 = [0] * len(t)
    for i in range(N):
        i_exp = model_11()
        K_11 = [K_11[i] + i_exp[i] for i in range(len(K_11))]
    K_11 = [K_11_i / N for K_11_i in K_11]
    K_11_th = M / (L + M)
    plt.subplot(3, 2, 1)
    plt.plot(t, K_11, label='Экспериментальная оценка')
    plt.plot(t, [K_11_th] * len(t), 'r--', label='Теоретическая оценка')
    #plt.title("один элемент и одна ремонтная бригада")
    plt.legend()

    K_21 = [0] * len(t)
    for i in range(N):
        i_exp = model_21()
        K_21 = [K_21[i] + i_exp[i] for i in range(len(K_21))]
    K_21 = [K_21_i / N for K_21_i in K_21]
    K_21_th = (2 * M * L + M ** 2) / (2 * (L ** 2) + 2 * M * L + M ** 2)
    plt.subplot(3, 2, 3)
    plt.plot(t, K_21, label='Экспериментальная оценка')
    plt.plot(t, [K_21_th] * len(t), 'r--', label='Теоретическая оценка')
    #plt.title("2 элемента и одна ремонтная бригада")
    plt.legend()

    K_22 = [0] * len(t)
    for i in range(N):
        i_exp = model_22()
        K_22 = [K_22[i] + i_exp[i] for i in range(len(K_22))]
    K_22 = [K_22_i / N for K_22_i in K_22]
    K_22_th = 1 - (1 - M / (M + L)) ** 2
    plt.subplot(3, 2, 5)
    plt.plot(t, K_22, label='Экспериментальная оценка')
    plt.plot(t, [K_22_th] * len(t), 'r--', label='Теоретическая оценка')
    #plt.title("2 элемента и 2 ремонтные бригады")
    plt.legend()

    K_42_1 = K_21_th * K_21_th
    K_42_2 = K_11_th * K_11_th
    K_42_up = K_22_th * K_22_th
    K_42_exp = [0] * len(t)
    for i in range(N):
        i_exp = model_42()
        K_42_exp = [K_42_exp[i] + i_exp[i] for i in range(len(K_42_exp))]
    K_42_exp = [K_42_exp_i / N for K_42_exp_i in K_42_exp]
    plt.subplot(1, 2, 2)
    plt.plot(t, [K_42_up] * len(t), label='верхняя граница')
    plt.plot(t, [K_42_1] * len(t), label='Нижняя граница, рассчитанная 1
способом')
```

```
    plt.plot(t, [K_42_2] * len(t), label='Нижняя граница, рассчитанная 2 способом')
    plt.plot(t, K_42_exp, label='Экспериментальная оценка')
    plt.legend()

    plt.show()
```