1. Цель работы.

Целью работы является имитационное моделирование и теоретический расчет коэффициента готовности восстанавливаемой системы.

2. Исходные данные.

На рис.1 представлена сложная схема, которая имеет 5 элементов и 3 ремонтные бригады:

Рис.1. Схема системы.

Данную сложную схему можно разбить на 3 простых элемента: 2 элемента двухэлементной системы и 1 один элемент одноэлементной системы.

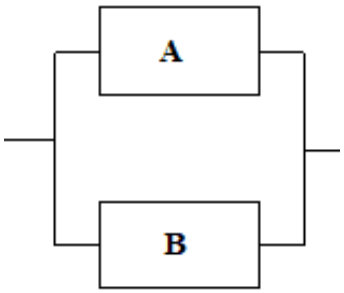

Рис.2. Двухэлементный фрагмент системы.

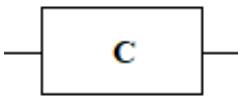

Рис.3. Одноэлементный фрагмент системы.

Зададим булеву функцию, определяющую состояние системы в целом по состояниям отдельных элементов.

Сложная система состоит из трех последовательно соединенных независимых фрагментов, следовательно система является работоспособной только при условии, что все три фрагмента также находятся в работоспособном состоянии.

Одноэлементный фрагмент системы, изображенный на рис. 3, находится в рабочем состоянии, если элемент C работает.

Двухэлементный фрагмент системы, изображенный на рис. 2, работает, если в рабочем состоянии находится хотя бы один из элементов A и B. Аналогично для двухэлементного фрагмента системы, состоящего из элементов D и E.

Таким образом булева функция, определяющая состояние системы, имеет вид:

$$F = C \cap (A \cup B) \cap (D \cup E)$$

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.

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

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. Результат моделирования.

Моделирование проводилось при количестве элементов $N = 30000$, интенсивности отказов $\lambda = 0{,}77$, интенсивности восстановления $\mu = 0{,}44$.

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 3 до 5. Верхняя граница будет вычисляться следующим образом:

$$K_{\text{верхняя}} = K_{2,2} \cdot K_{1,1} \cdot K_{2,2}$$

Нижняя граница вычисляется двумя способами.

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

$$K_{\text{нижняя1}} = K_{2,1} \cdot K_{1,1} \cdot K_{2,1}$$

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

$$K_{\text{нижняя2}} = K_{1,1} \cdot K_{1,1} \cdot K_{1,1}$$

5. График результата моделирования.

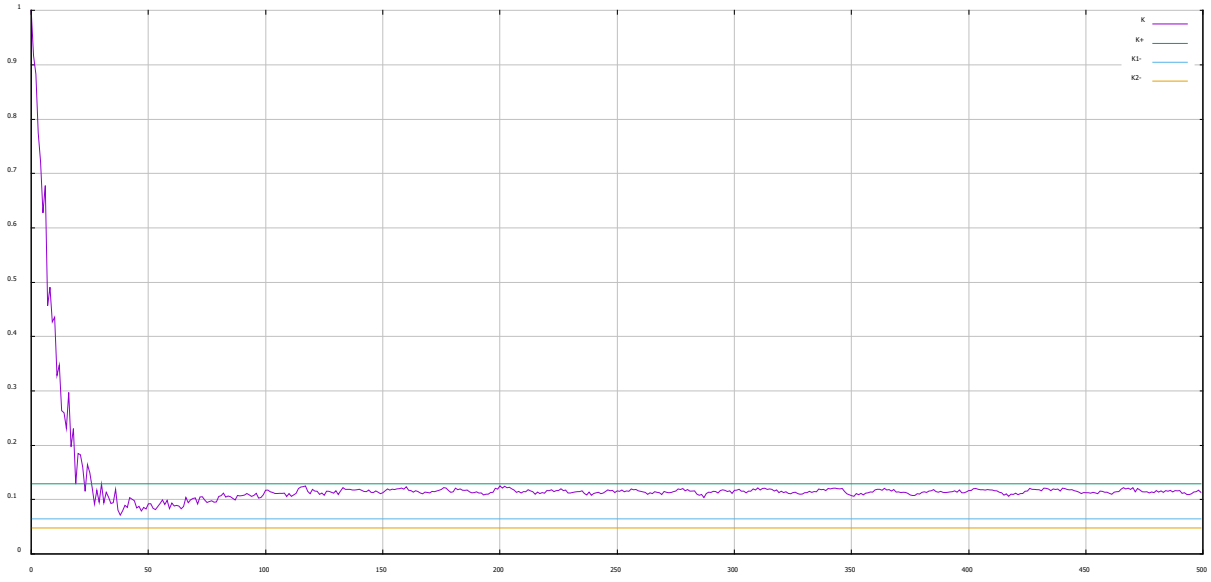

Рис.4. График зависимости коэффициента готовности схемы (5 систем и 3 ремонтные бригады) от времени.

6. 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.

Приложение 1.

Листинг программы.

```
#include <iostream>
#include <vector>
#include <fstream>

using namespace std;
#define intMax 32767

double lambda = 0.77;
double mue = 0.44;

int n = 30000; //количество систем
int k = 500;
double delt = 0.1;

struct Block
{
	double Tr;		//время работы
	double Tv;		//время восстановления
	bool state;		//true - работает	false - восстанавливается
	double end_time;	//до какого времени

	void init(double x)
	{
		Tr = -(log(x) / lambda);
		Tv = -(log(x) / mue);
		state = true;
		end_time = Tr;
	}
	void change(bool st, double time)
	{
		state = st;
		end_time = time;
	}
};

struct Brig
{
	bool state; // занята - true	не занята - false
	double end_time;	// до какого времени занята
	int index;

	void init()
	{
		state = false;
		end_time = 0;
		index = 0;
	}
};

struct System
{
	vector<Block> block;	// 5
	vector<Brig> brig;	// 3

	void init()
	{
		for (int i = 0; i < 5; i++)
		{
			double x;
			while (1)
			{
```

```
                        x = (double)(rand() % intMax) / intMax;
                        if (x > 0)
                        {
                                break;
                        }
                }
                Block tmp;
                tmp.init(x);
                block.push_back(tmp);
        }
        for (int i = 0; i < 3; i++)
        {
                Brig tmp;
                tmp.init();
                brig.push_back(tmp);
        }
}

double in_repair(int b, double time) // ремонт
{
        int minvr = 0;
        for (int i = 0; i < brig.size(); i++)
        {
                if (!brig[i].state) // не занята
                {
                        brig[i].state = true;
                        brig[i].end_time = time + block[b].Tv;
                        return brig[i].end_time;
                }
                else // ищем которая быстрее освободится
                {
                        if (brig[i].end_time < brig[minvr].end_time)
                        {
                                minvr = i;
                        }
                }
        }
        brig[minvr].end_time += block[b].Tv; // увеличиваем время работы
бригады
        return brig[minvr].end_time;
}

bool get_state(double time)
{
        bool res = false;
        if (block[0].state || block[1].state)
        {
                if (block[2].state)
                {
                        if (block[3].state || block[4].state)
                        {
                                res = true;
                        }
                }
        }
        for (int i = 0; i < brig.size(); i++)
        {
                if (time + delt > brig[i].end_time)
                {
                        brig[i].state = false; // освобождаем бригаду
                }
        }
        for (int i = 0; i < block.size(); i++)
        {
                if (time + delt > block[i].end_time)
```

```
                {
                        if (block[i].state) // работал -> не работает (сломался)
                        {
                                double t = in_repair(i, time+delt);
                                block[i].change(false, t);
                        }
                        else // не работал -> работает (починили)
                        {
                                block[i].change(true, time + delt + block[i].Tr);
                        }
                }
        }
        return res;
    }
};

vector<System> systems;

void init()
{
    for (int i = 0; i < n; i++)
    {
        System tmp;
        tmp.init();
        systems.push_back(tmp);
    }
}

void getK()
{
    vector<double> Ks;
    for (int j = 0; j < k; j++)
    {
        double K = 0;
        for (int i = 0; i < n; i++)
        {
            K += systems[i].get_state(j * delt);
        }
        Ks.push_back(K / n);
    }

    string namef = "K.txt";
    ofstream f1;
    f1.open(namef);
    if (!f1.is_open())
    {
        cout << "!!!!!!!!!" << endl;
    }
    for (int i = 0; i < Ks.size(); i++)
    {
        f1 << Ks[i] << endl;
    }
    f1.close();
}

void getKver()
{
    double Kver;

    double K11 = mue / (mue + lambda);
    double K22 = 1 - (1 - K11) * (1 - K11);

    Kver = K22 * K11 * K22;

    string namef = "Kver.txt";
```

```
	ofstream f1;
	f1.open(namef);
	if (!f1.is_open())
	{
		cout << "!!!!!!!!!" << endl;
	}
	for (int i = 0; i < k; i++)
	{
		f1 << Kver << endl;
	}
	f1.close();

}

void getKnish1()
{
	double Knish;
	double K11 = mue / (mue + lambda);
	double K21 = ((2 * mue * lambda) + pow(mue, 2)) / ((2 * pow(lambda, 2)) + (2
* mue * lambda) + pow(mue, 2));

	Knish = K21 * K11 * K21;

	string namef = "Knish1.txt";
	ofstream f1;
	f1.open(namef);
	if (!f1.is_open())
	{
		cout << "!!!!!!!!!" << endl;
	}
	for (int i = 0; i < k; i++)
	{
		f1 << Knish << endl;
	}
	f1.close();
}

void getKnish2()
{
	double Knish;

	double K11 = mue / (mue + lambda);

	Knish = K11 * K11 * K11;

	string namef = "Knish2.txt";
	ofstream f1;
	f1.open(namef);
	if (!f1.is_open())
	{
		cout << "!!!!!!!!!" << endl;
	}
	for (int i = 0; i < k; i++)
	{
		f1 << Knish << endl;
	}
	f1.close();
}

void drow()
{
	ofstream f2;
	f2.open("K.plt");
	if (!f2.is_open())
	{
```

```
            cout << "!!!!!!!!!" << std::endl;
        }
        f2 << "set terminal win\n";
        f2 << "set grid\n";
        f2 << "set multiplot\n";
        f2 << "plot \"K.txt\" using 1 with lines linecolor 1 title \"K\" ,";
        f2 << "\"Kver.txt\" using 1 with lines linecolor 2 title \"K+\" ,";
        f2 << "\"Knish1.txt\" using 1 with lines linecolor 3 title \"K1-\" ,";
        f2 << "\"Knish2.txt\" using 1 with lines linecolor 4 title \"K2-\" \n";
        f2 << "unset multiplot\n";
        f2.close();
}

int main()
{
        init();
        getK();
        getKver();
        getKnish1();
        getKnish2();
        drow();
}
```