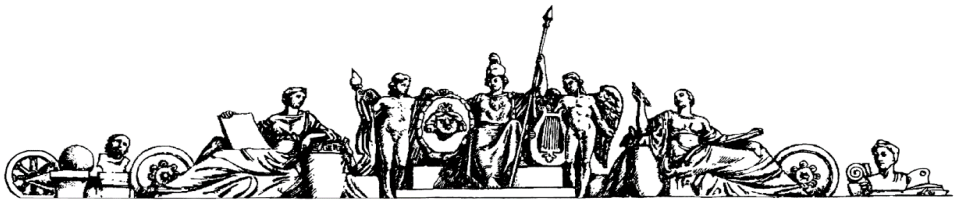


Министерство образования и науки Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени
Н. Э. Баумана (национальный исследовательский университет)»

Факультет «Робототехники и комплексной автоматизации»
Кафедра «Системы автоматизированного проектирования»

Домашнее задание

по курсу «Модели и методы анализа проектных решений»

Вариант №51

Студент: Бурча Артём гр. РК6-71

Преподаватель: к.т.н., доцент каф. РК6

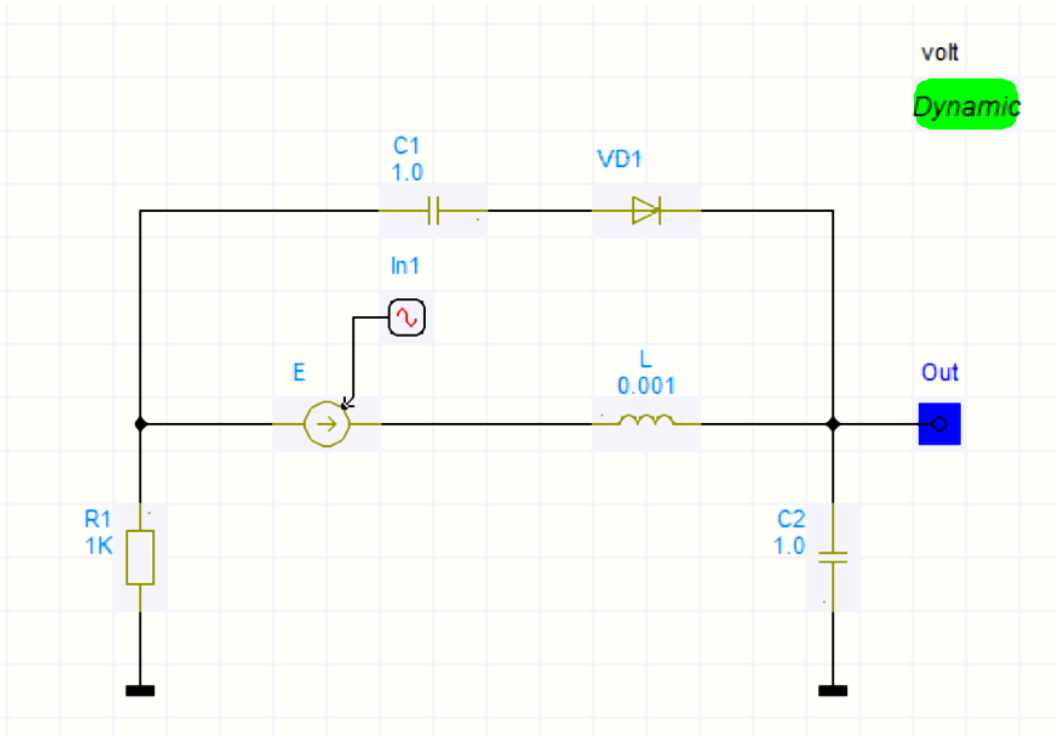
Трудоношин В.А.

Москва 2017г.

Содержание

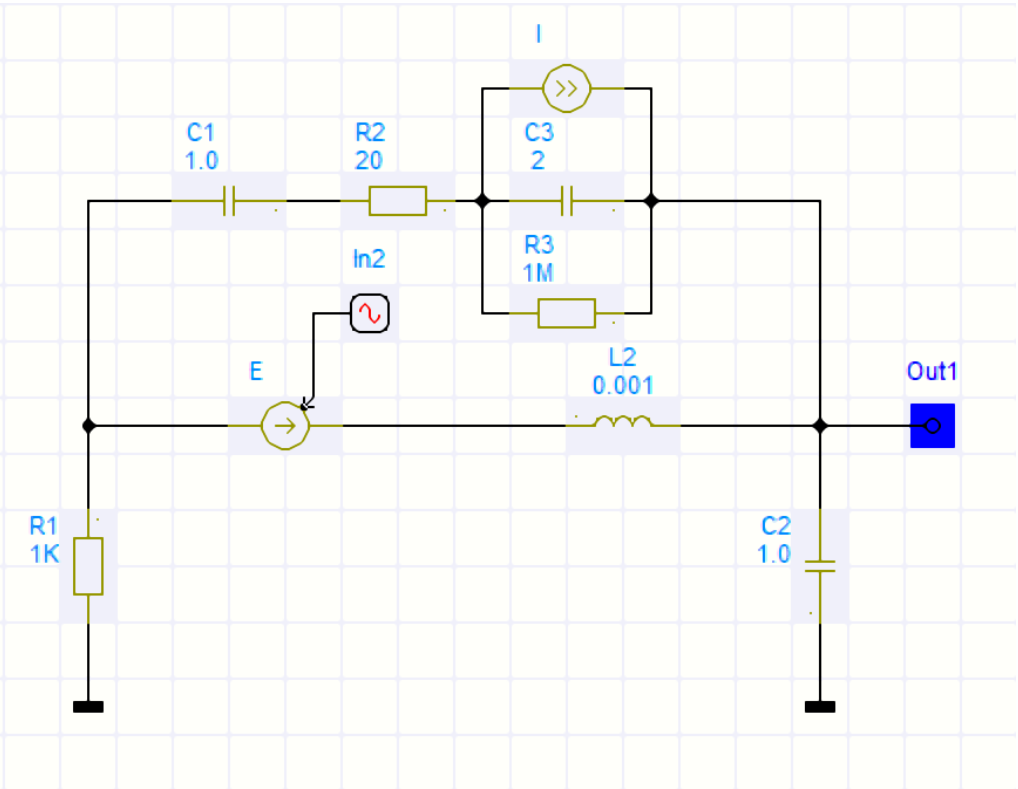
Задание.

Дана схема выполнить расчет в программе ПА9 и решить эту же схему табличным методом формирования математической модели.

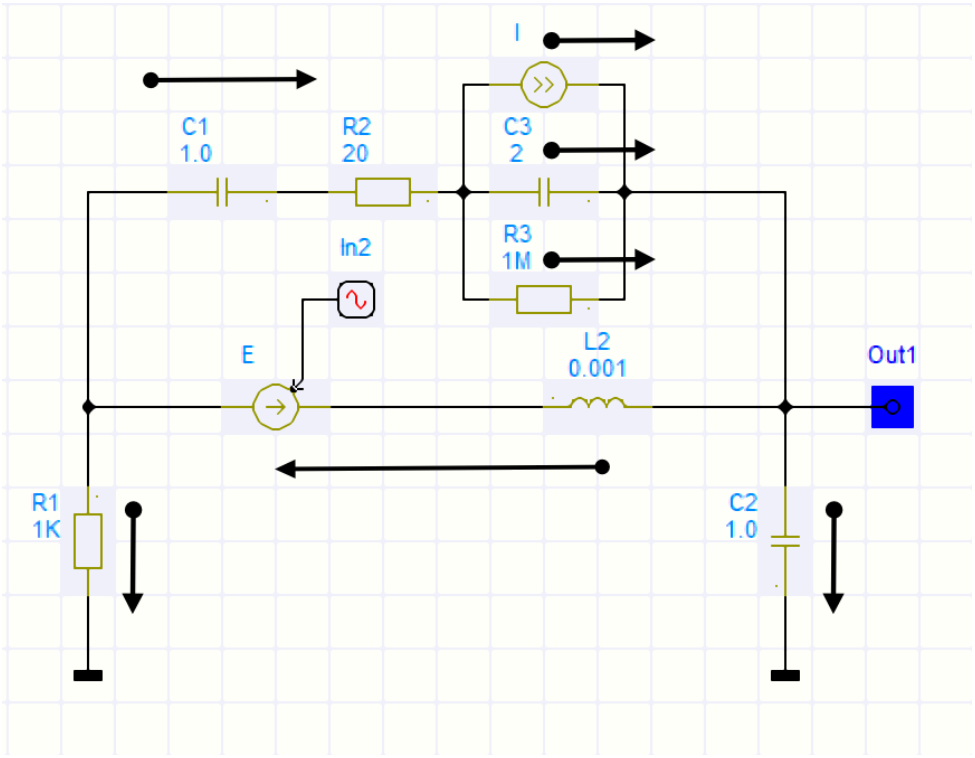


Табличный метод формирования ММ.

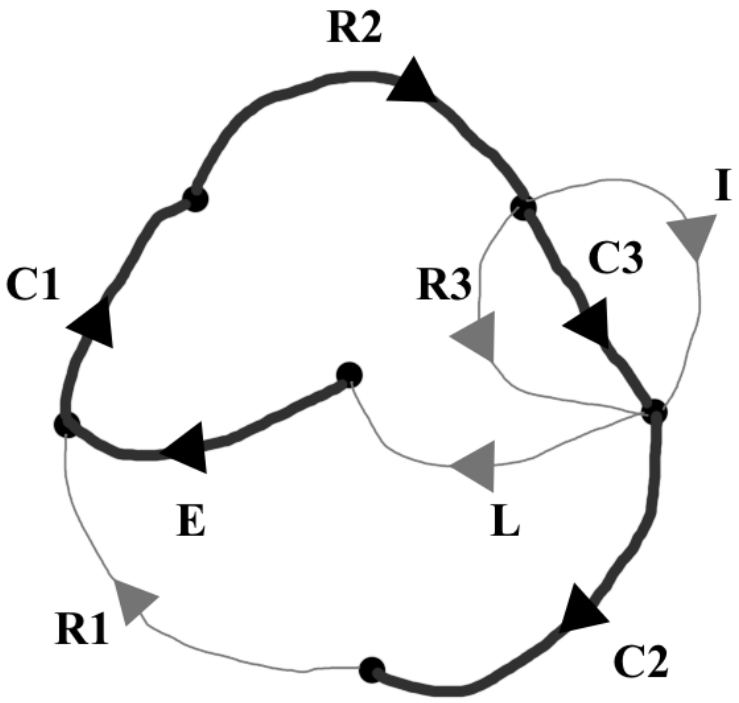
Для применения табличного метода диод был заменен по схеме замещения и, в результате, получилась следующая схема:



Для данной схемы зададим направления токов:



Теперь построим граф для этой схемы и выделим в нем дерево.



Черным обозначены ветви дерева, а серым хорды.

Теперь составим матрицу M по полученному дереву.

	E	**C1**	**C2**	**C3**	**R2**
R1		-1	-1	-1	-1
R3				-1	
L	1	1		1	1
I				-1	

Теперь составим СЛАУ для нахождения токов и напряжений всех элементов методом Ньютона. СЛАУ для табличного метода в общем случае имеет следующий вид:

$$\begin{pmatrix} 1 & & & M \\ & 1 & -M^T & \\ K_{31} & K_{32} & K_{33} & K_{34} \\ K_{41} & K_{42} & K_{43} & K_{44} \end{pmatrix} * \begin{pmatrix} dU_{хорд} \\ dI_{ветвей\ дерева} \\ dI_{хорд} \\ dU_{ветвей\ дерева} \end{pmatrix} = - \begin{pmatrix} M * U_{ветвей\ дерева} + U_{хорд} \\ I_{ветвей\ дерева} - M^T * I_{хорд} \\ f(U_{в.д.}, I_{в.д.}, I_{хорд}, U_{хорд}) \\ f(U_{в.д.}, I_{в.д.}, I_{хорд}, U_{хорд}) \end{pmatrix}$$

Для данной конкретной задачи СЛАУ примет следующий вид:

1														-1	-1	-1	-1	dUR1
	1															-1		dUR3
		1											1	1		1	1	dUL
			1													-1		dUI
				1							-1							dIE
					1				1		-1							dIC1
						1			1									dIC2
							1		1	1	-1	1						dIC3
								1	1		-1							dIR2
-1/R1									1									dIR1
	-1/R3									1								dIR3
		1									-L/dt							dIL
			-alpha									1						dII
													1					dUE
					1									-C1/dt				dUC1
						1									-C2/dt			dUC2
							1									-C3/dt		dUC3
								-R2									1	dUR2

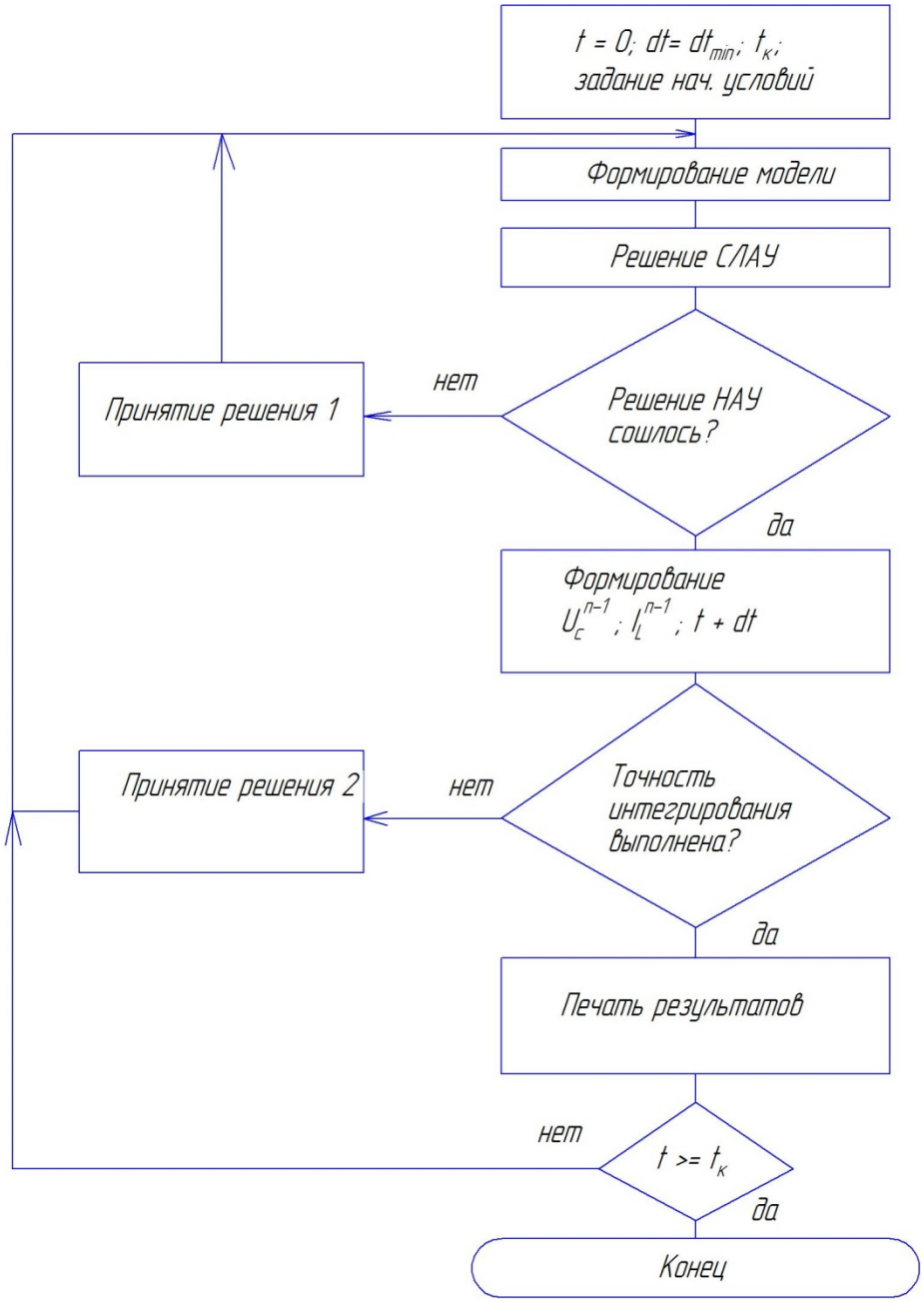
=

UR1 - UC1 - UC2 - UC3 - UR2
UR3 - UC3
UL + UE + UC1 + UC3 + UR2
UI - UC3
IE - IL
IC1 + IR1 - IL
IC2 + IR1
IC3 + IR1 + IR3 - IL + II
IR2 + IR1 - IL
IR1 - UR1/R1
IR3 - UR3/R3
- L/dt * (IL^{n} - IL^{n-1}) + UL
II - It * (exp(UI/MFT) - 1)
UE - 10*sin(2*pi *t / 0.0001)
- C1/dt * ($UC1^{n}$ - $UC1^{n-1}$) + IC1
- C2/dt * ($UC2^{n}$ - $UC2^{n-1}$) + IC2
- C3/dt * ($UC3^{n}$ - $UC3^{n-1}$) + IC3
UR2 - IR2*R2

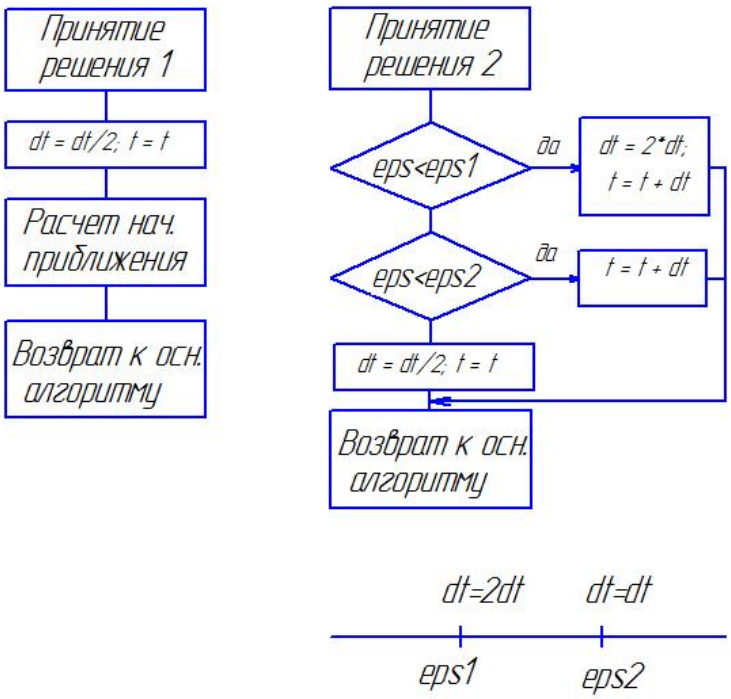
,

alpha = It/MFT * exp(UI/MFT)

Алгоритм решения задачи табличным методом.



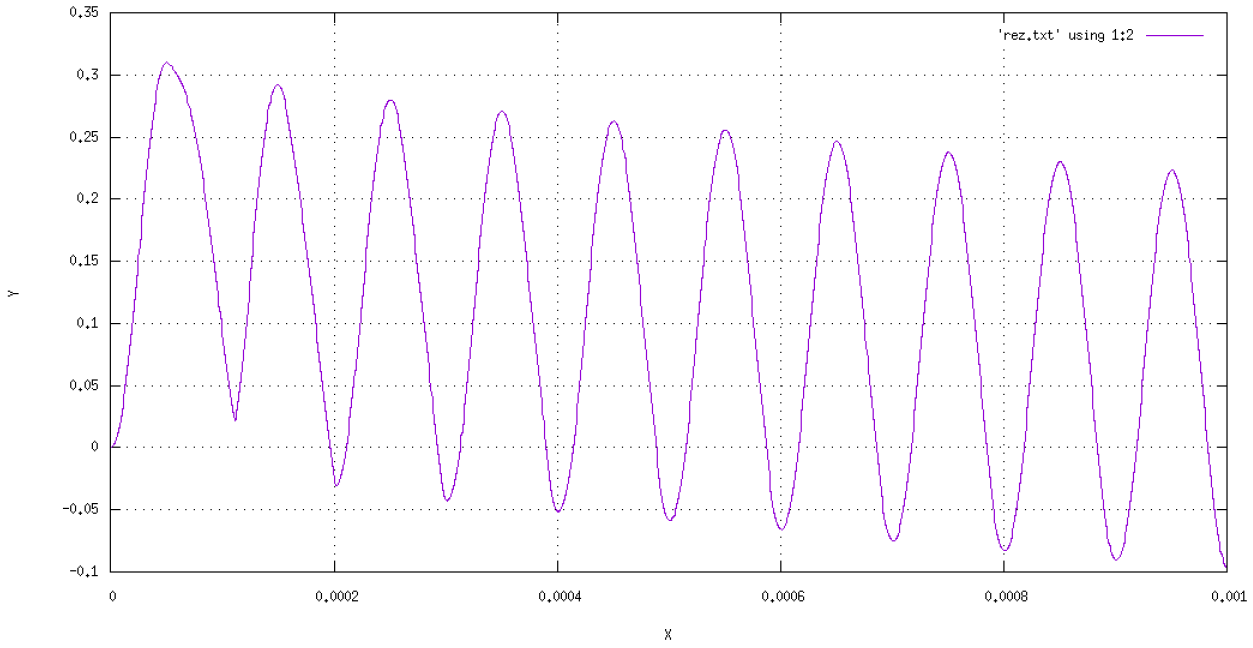
Блок схема основного алгоритма.



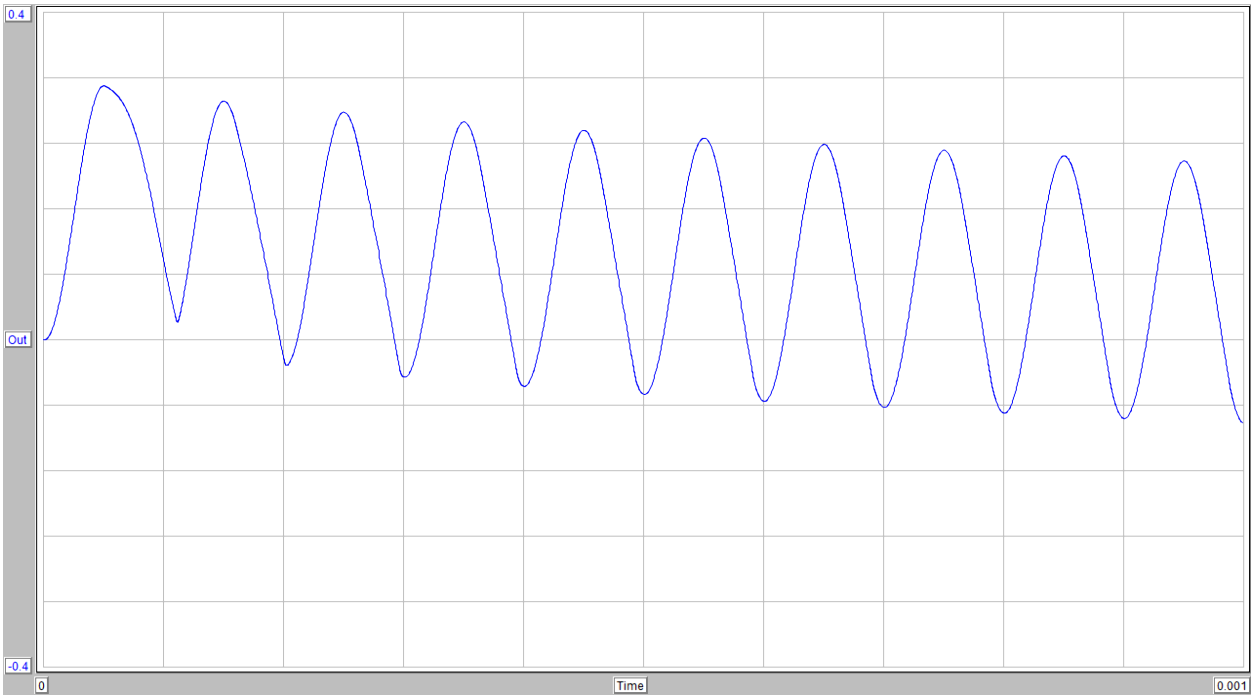
Блок-схема.

Результат работы программы.

Результатом работы программы является график зависимости напряжения на конденсаторе C2 от времени, т.е. показания элемента out1 в ПА9 ($t \in [0; T_{конечн.}]$, $T_{конечн.} = 0.001$ с). График построен с помощью программы gnuplot 5.2 patchlevel 2.



Сравнение результатов с программой ПА9.



Результаты расчетов в ПА9.

Листинг программы.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


int Gauss(double *matrix, double *vector, int N) {
    long int i, j, k;
    double diagonalElement;

    for (k = 0; k < N; k++) {
        diagonalElement = matrix[N * k + k];
        if (diagonalElement == 0)
            return 1;
        for (j = k; j < N; j++)
            matrix[N * k + j]/= diagonalElement;
        vector[k]/= diagonalElement;

        for (i = k + 1; i < N; i++) {
            diagonalElement = matrix[N * i + k];
            for (j = k; j < N; j++)
                matrix[N * i + j]-= matrix[N * k + j] * diagonalElement;
            vector[i]-= vector[k] * diagonalElement;
        }
    }

    for (i = N - 2; i >= 0; i--)
        for (j = i + 1; j < N; j++)
            vector[i]-= matrix[N * i + j] * vector[j];
    return 0;
}
```

```
double VoltageSource(double t) {
    double amplitude = 10.0;
    double frequency = 2 * M_PI/1e-4;
    double phase = 0.0;

    return amplitude * sin(frequency * t + phase);
}

double MaxElement(double *vector, int N) {
    int i;
    double result = 0;

    for (i = 0; i < N; i++) {
        if (result < fabs(vector[i])) {
            result = fabs(vector[i]);
        }
    }
    return result;
}


int main(int argc, const char * argv[]) {
    FILE* f=fopen("rez.txt","w");
    const int dimension = 18;

    const double t_fin = 1e-3;
    double step_t = 1e-6;
    double step_t_last = step_t;

    const double C1 = 1e-6;
    const double C2 = 1e-6;
    const double L = 0.001;
    const double R1 = 1000.0;

    const double IT = 1e-12;
    const double C3 = 2e-12;
    const double R2 = 20.0;
    const double MFT = 0.026;
    const double R3 = 1000000.0;

    double time = 0.0;

    double UR1 = 0.0;  double prevUR1 = 0.0;
    double UR3 = 0.0;  double prevUR3 = 0.0;
    double UL = 0.0;   double prevUL = 0.0;
    double UI = 0.0;   double prevUI = 0.0;
    double IE = 0.0;   double prevIE = 0.0;
    double IC1 = 0.0;  double prevIC1 = 0.0;
    double IC2 = 0.0;  double prevIC2 = 0.0;
    double IC3 = 0.0;  double prevIC3= 0.0;
    double IR2 = 0.0;  double prevIR2 = 0.0;
    double IR1 = 0.0;  double prevIR1 = 0.0;
    double IR3 = 0.0;  double prevIR3 = 0.0;
    double IL = 0.0;   double prevIL = 0.0;
    double II = 0.0;   double prevII = 0.0;
    double UE = 0.0;   double prevUE = 0.0;
    double UC1 = 0.0;  double prevUC1 =0.0;
    double UC2 = 0.0;  double prevUC2 =0.0;  double extraUC2 =0.0;
    double UC3 = 0.0;  double prevUC3 =0.0;
    double UR2 = 0.0;  double prevUR2 = 0.0;
    double epsilon_max = 0.001;
```

```
    double epsilon_min = 0.00001;
    double *matrix = (double*)malloc(sizeof(double) * dimension * dimension);
    double *vector = (double*)malloc(sizeof(double) * dimension); //delta

    double K = 1000.0;

    while (time <= t_fin) {
        double delta_t = step_t;
        double deviation = 0.0001;
        int iteration = 0;
        int continue_flag = 1;

        while (continue_flag) {
            for (int i = 0; i < dimension; i++) {
                for (int j = 0; j < dimension; j++) {
                    matrix[dimension * i + j] = 0;
                }
                vector[i] = 0;
            }

            matrix[dimension * 0 + 0] = 1.0;
            matrix[dimension * 0 + 14] = -1.0;
            matrix[dimension * 0 + 15] = -1.0;
            matrix[dimension * 0 + 16] = -1.0;
            matrix[dimension * 0 + 17] = -1.0;
            vector[0] = -(UR1-UC1-UC2-UC3-UR2);

            matrix[dimension * 1 + 1] = 1.0;
            matrix[dimension * 1 + 16] = -1.0;
            vector[1] = -(UR3-UC3);

            matrix[dimension * 2 + 2] = 1.0;
            matrix[dimension * 2 + 13] = 1.0;
            matrix[dimension * 2 + 14] = 1.0;
            matrix[dimension * 2 + 16] = 1.0;
            matrix[dimension * 2 + 17] = 1.0;
            vector[2] = -(UL+UE+UC1+UC3+UR2);

            matrix[dimension * 3 + 3] = 1.0;
            matrix[dimension * 3 + 16] = -1.0;
            vector[3] = -(UI-UC3);

            matrix[dimension * 4 + 4] = 1.0;
            matrix[dimension * 4 + 11] = -1.0;
            vector[4] = -(IE-IL);

            matrix[dimension * 5 + 5] = 1.0;
            matrix[dimension * 5 + 9] = 1.0;
            matrix[dimension * 5 + 11] = -1.0;
            vector[5] = -(IC1+IR1-IL);

            matrix[dimension * 6 + 6] = 1.0;
            matrix[dimension * 6 + 9] = 1.0;
            vector[6] = -(IC2+IR1);

            matrix[dimension * 7 + 7] = 1.0;
            matrix[dimension * 7 + 9] = 1.0;
            matrix[dimension * 7 + 10] = 1.0;
            matrix[dimension * 7 + 11] = -1.0;
            matrix[dimension * 7 + 12] = 1.0;
            vector[7] = -(IC3 + IR1 + IR3 - IL + II);
```

```
            matrix[dimension * 8 + 8] = 1.0;
            matrix[dimension * 8 + 9] = 1.0;
            matrix[dimension * 8 + 11] = -1.0;
            vector[8] = -(IR2+IR1-IL);

            matrix[dimension * 9 + 0] = -1.0/R1;
            matrix[dimension * 9 + 9] = 1.0;
            vector[9] = -(IR1-UR1/R1);

            matrix[dimension * 10 + 1] = -1.0/R3;
            matrix[dimension * 10 + 10] = 1.0;
            vector[10] = -(IR3-UR3/R3);

            matrix[dimension * 11 + 2] = 1.0;
            matrix[dimension * 11 + 11] = -L/delta_t;
            vector[11] = -(-L/delta_t*(IL-prevIL) + UL);

            matrix[dimension * 12 + 3] = -(IT/MFT * exp(UI/MFT)) ;
            matrix[dimension * 12 + 12] = 1.0;
            vector[12] = -(II - IT*(exp(UI/MFT)-1));

//            matrix[dimension * 12 + 3] = -K ;
//            matrix[dimension * 12 + 12] = 1.0;
//            vector[12] = -(II - K*(UI-0.7));

            matrix[dimension * 13 + 13] = 1.0;
            vector[13] = -(UE - VoltageSource(time));

            matrix[dimension * 14 + 5] = 1.0;
            matrix[dimension * 14 + 14] = -C1/delta_t;
            vector[14] = -(-C1/delta_t*(UC1-prevUC1) + IC1);

            matrix[dimension * 15 + 6] = 1.0;
            matrix[dimension * 15 + 15] = -C2/delta_t;
            vector[15] = -(-C2/delta_t*(UC2-prevUC2) + IC2);

            matrix[dimension * 16 + 7] = 1.0;
            matrix[dimension * 16 + 16] = -C3/delta_t;
            vector[16] = -(-C3/delta_t*(UC3-prevUC3) + IC3);

            matrix[dimension * 17 + 8] = 1.0;
            matrix[dimension * 17 + 17] = 1.0;
            vector[17] = -(UR2-IR2*R2);

            if (Gauss(matrix, vector, dimension))
                return 1;

            UR1 += vector[0];
            UR3 += vector[1];
            UL += vector[2];
            UI += vector[3];
            IE += vector[4];
            IC1 += vector[5];
            IC2 += vector[6];
            IC3 += vector[7];
            IR2 += vector[8];
            IR1 += vector[9];
            IR3 += vector[10];
            IL += vector[11];
            II += vector[12];
            UE += vector[13];
            UC1 += vector[14];
```

```
            UC2 += vector[15];
            UC3 += vector[16];
            UR2 += vector[17];

            iteration++;
            //counter++;
            if (MaxElement(vector, dimension) > deviation) {
                continue_flag = 1;
            }
            else {
                continue_flag = 0;
            }

            if (iteration > 17 && continue_flag == 1) {
                iteration = 0;
                delta_t /= 2;
                UR1 = prevUR1;
                UR3 = prevUR3;
                UL = prevUL;
                UI = prevUI;
                IE = prevIE;
                IC1 = prevIC1;
                IC2 = prevIC2;
                IC3 = prevIC3;
                IR2 = prevIR2;
                IR1 = prevIR1;
                IR3 = prevIR3;
                IL = prevIL;
                II =  prevII;
                UE = prevUE;
                UC1 = prevUC1;
                UC2 = prevUC2;
                UC3 = prevUC3;
                UR2 = prevUR2;
            }
        } //Newton cycle

        double epsilon = 0.5*fabs(step_t * ((prevUC2 - extraUC2)/step_t_last
- (UC2 - prevUC2)/step_t));
        //last предыдущее значение.
        //extra - предпредыдущее значение.

        if (epsilon > epsilon_max) {
            step_t /=2;
            UR1 = prevUR1;
            UR3 = prevUR3;
            UL = prevUL;
            UI = prevUI;
            IE = prevIE;
            IC1 = prevIC1;
            IC2 = prevIC2;
            IC3 = prevIC3;
            IR2 = prevIR2;
            IR1 = prevIR1;
            IR3 = prevIR3;
            IL = prevIL;
            II =  prevII;
            UE = prevUE;
            UC1 = prevUC1;
            UC2 = prevUC2;
            UC3 = prevUC3;
            UR2 = prevUR2;
```

```
        }
        else {

            extraUC2 = prevUC2;
            prevUR1 = UR1;
            prevUR3 = UR3;
            prevUL = UL;
            prevUI = UI;
            prevIE = IE;
            prevIC1 = IC1;
            prevIC2 = IC2;
            prevIC3 = IC3;
            prevIR2 = IR2;
            prevIR1 = IR1;
            prevIR3 = IR3;
            prevIL = IL;
            prevII = II;
            prevUE = UE;
            prevUC1 = UC1;
            prevUC2 = UC2;
            prevUC3 = UC3;
            prevUR2 = UR2;
            step_t_last = delta_t;

            fprintf(f, "%9.6f %7.6f\n", time, UC2);

            time +=delta_t;
            if (epsilon < epsilon_min && step_t < 1e-6) {
                step_t *= 2;
            }
        }
    }
    fclose(f);
    return 0;
}
```