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

ФАКУЛЬТЕТ *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ДОМАШНЕГО ЗАДАНИЯ

по дисциплине: «Модели и методы анализа проектных решений»

Студент	Мудриченко Михаил
Группа	РК6-72Б
Тип задания	Домашнее задание

Студент _____ **Мудриченко М.Н.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Трудоношин В.А.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2022 г.

Оглавление

Задание

Требуется сформировать математическую модель схемы, изображенной на рисунке используя узловой модифицированный метод (1 вариант) и сравить результат с полученным в ПА9.

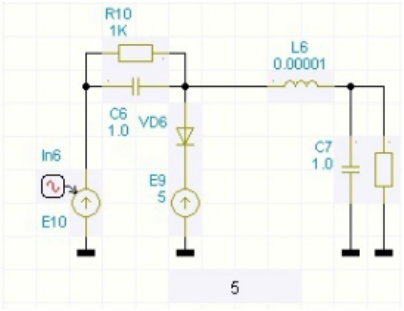

Рисунок 1 — Исходная схема

Математическая модель

Базис узлового модифицированного метода составляют узловые потенциалы и токи источников напряжения.

Для использования диода используется его эквивалентная схема, при этом сила тока диода насчитывается по формуле:

$$I_d = I_t(\exp(\frac{U_d}{MFt}) - 1)$$

На рисунке 2 представлена эквивалентная схема с расставленными номерами узлов и направлениями токов.

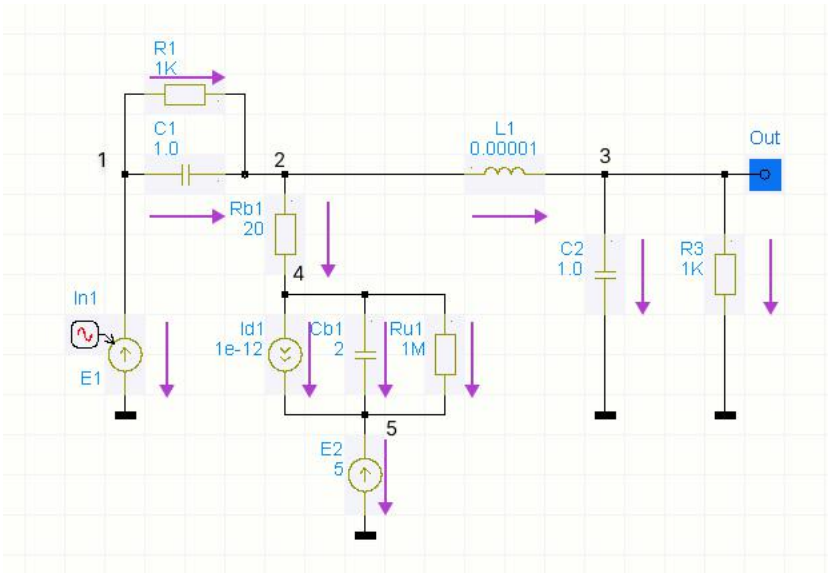

Рисунок 2 — Эквивалентная схема

Ниже представлены математические модели компонентов. Итоговая математическая модель получается путем ансамблирования моделей компонентов в соответствии со схемой.

Резистор

$$\begin{bmatrix} \frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & \frac{1}{R} \end{bmatrix} \begin{bmatrix} \Delta\varphi_i \\ \Delta\varphi_j \end{bmatrix} = - \begin{bmatrix} \frac{\varphi_i - \varphi_j}{R} \\ -\frac{\varphi_i - \varphi_j}{R} \end{bmatrix}$$

Емкость

$$\begin{bmatrix} \frac{C}{\Delta t} & -\frac{C}{\Delta t} \\ -\frac{C}{\Delta t} & \frac{C}{\Delta t} \end{bmatrix} \begin{bmatrix} \Delta\varphi_i \\ \Delta\varphi_j \end{bmatrix} = - \begin{bmatrix} \frac{C}{\Delta t}(\varphi_i - \varphi_j - U_c^{n-1}) \\ \frac{C}{\Delta t}(\varphi_i - \varphi_j - U_c^{n-1}) \end{bmatrix}$$

Индуктивность

$$\begin{bmatrix} \frac{\Delta t}{L} & -\frac{\Delta t}{L} \\ -\frac{\Delta t}{L} & \frac{\Delta t}{L} \end{bmatrix} \begin{bmatrix} \Delta\varphi_i \\ \Delta\varphi_j \end{bmatrix} = - \begin{bmatrix} I_L^{n-1} + \frac{\Delta t}{L}(\varphi_i - \varphi_j) \\ -I_L^{n-1} - \frac{\Delta t}{L}(\varphi_i - \varphi_j) \end{bmatrix}$$

Источник напряжения

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta\varphi_i \\ \Delta\varphi_j \\ \Delta I_E \end{bmatrix} = - \begin{bmatrix} -I_E \\ I_E \\ \varphi_j - \varphi_i - E \end{bmatrix}$$

Источник тока в модели диода

$$\begin{bmatrix} \frac{I_t}{MFt}\exp(\frac{\varphi_i - \varphi_j}{MFt}) & -\frac{I_t}{MFt}\exp(\frac{\varphi_i - \varphi_j}{MFt}) \\ -\frac{I_t}{MFt}\exp(\frac{\varphi_i - \varphi_j}{MFt}) & \frac{I_t}{MFt}\exp(\frac{\varphi_i - \varphi_j}{MFt}) \end{bmatrix} \begin{bmatrix} \Delta\varphi_i \\ \Delta\varphi_j \end{bmatrix} = - \begin{bmatrix} I_t(\exp(\frac{\varphi_i - \varphi_j}{MFt}) - 1) \\ -I_t(\exp(\frac{\varphi_i - \varphi_j}{MFt}) - 1) \end{bmatrix}$$

Итоговая математическая модель:

$$\begin{bmatrix}
\frac{C_1}{\Delta t}+\frac{1}{R_1} & -\frac{C_1}{\Delta t}-\frac{1}{R_1} & 0 & 0 & 0 & 0 & 1 \\
-\frac{C_1}{\Delta t}-\frac{1}{R_1} & \frac{C_1}{\Delta t}+\frac{1}{R_1}+\frac{\Delta t}{L}+\frac{1}{R_b} & -\frac{\Delta t}{L} & -\frac{1}{R_b} & 0 & 0 & 0 \\
0 & -\frac{\Delta t}{L} & \frac{\Delta t}{L}+\frac{C_2}{\Delta t}+\frac{1}{R_2} & 0 & 0 & 0 & 0 \\
0 & -\frac{1}{R_b} & 0 & \frac{1}{R_b}+\frac{C_b}{\Delta t}+\frac{1}{R_u}+a & -\frac{C_b}{\Delta t}-\frac{1}{R_u}-a & 0 & 0 \\
0 & 0 & 0 & -\frac{C_b}{\Delta t}-\frac{1}{R_u}-a & \frac{C_b}{\Delta t}+\frac{1}{R_u}+a & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\Delta\varphi_1 \\ \Delta\varphi_2 \\ \Delta\varphi_3 \\ \Delta\varphi_4 \\ \Delta\varphi_5 \\ \Delta I_{E_2} \\ \Delta I_{E_1}
\end{bmatrix} =$$

$$= -\begin{bmatrix}
\frac{C_1}{\Delta t}(\varphi_1-\varphi_2-U_{c_1}^{n-1})+\frac{\varphi_1-\varphi_2}{R_1}+I_{E_1} \\
-\frac{C_1}{\Delta t}(\varphi_1-\varphi_2-U_{c_1}^{n-1})-\frac{\varphi_1-\varphi_2}{R_1}+\frac{\Delta t}{L}(\varphi_2-\varphi_3)+I_L^{n-1}+\frac{\varphi_2-\varphi_4}{R_b} \\
-\frac{\Delta t}{L}(\varphi_2-\varphi_3)-I_L^{n-1}+\frac{C_2}{\Delta t}(\varphi_3-U_{c_2}^{n-1})+\frac{\varphi_3}{R_2} \\
-\frac{\varphi_2-\varphi_4}{R_b}+\frac{C_b}{\Delta t}(\varphi_4-\varphi_5-U_{c_b}^{n-1})+\frac{\varphi_4-\varphi_5}{R_u}+I_d \\
-\frac{C_b}{\Delta t}(\varphi_4-\varphi_5-U_{c_b}^{n-1})-\frac{\varphi_4-\varphi_5}{R_u}-I_d+I_{E_2} \\
\varphi_5-E_2 \\
\varphi_1-E_1
\end{bmatrix}$$

$$a=\frac{I_t}{MFt}\exp\left(\frac{\varphi_4-\varphi_5}{MFt}\right)$$

$$I_d=I_t\left(\exp\left(\frac{\varphi_4-\varphi_5}{MFt}\right)-1\right)$$

$$E_1=A\sin\left(\frac{2\pi}{T}\right)$$

A — амплитуда источника E_1

T — период источника E_1

Результат работы программы

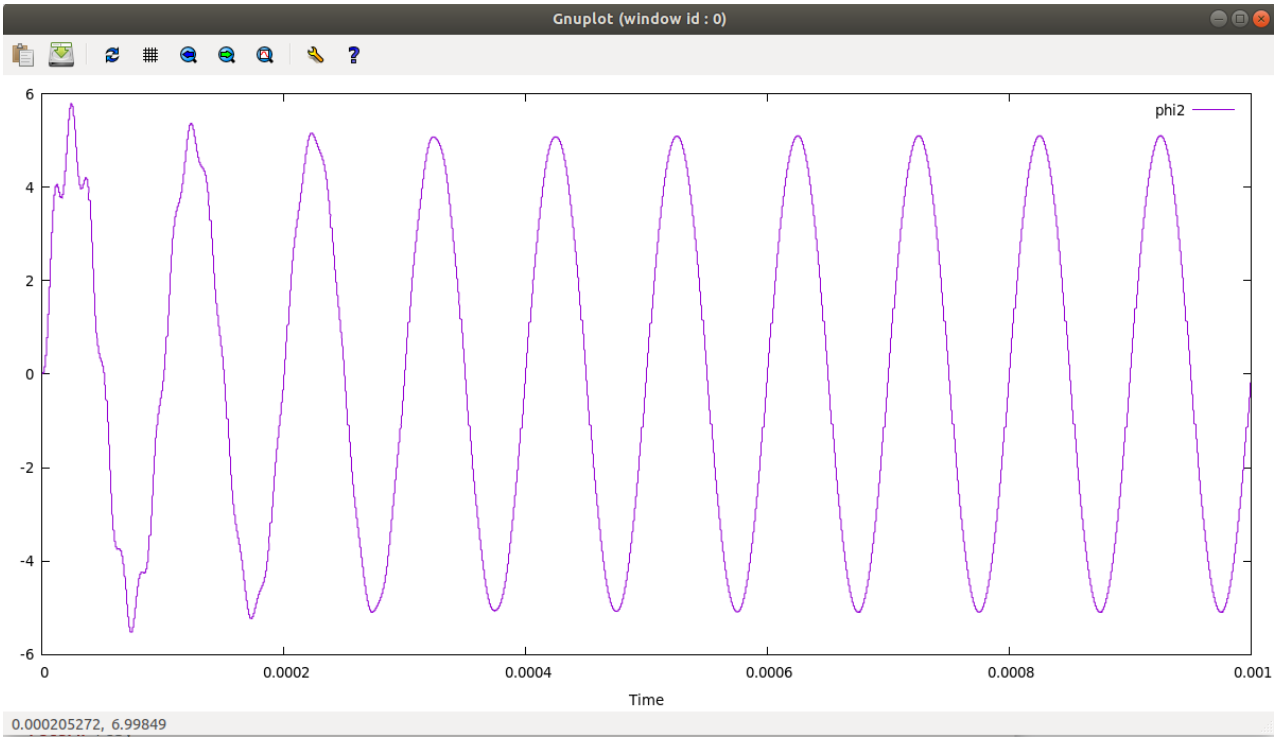

Рисунок 3 — Результат работы программы

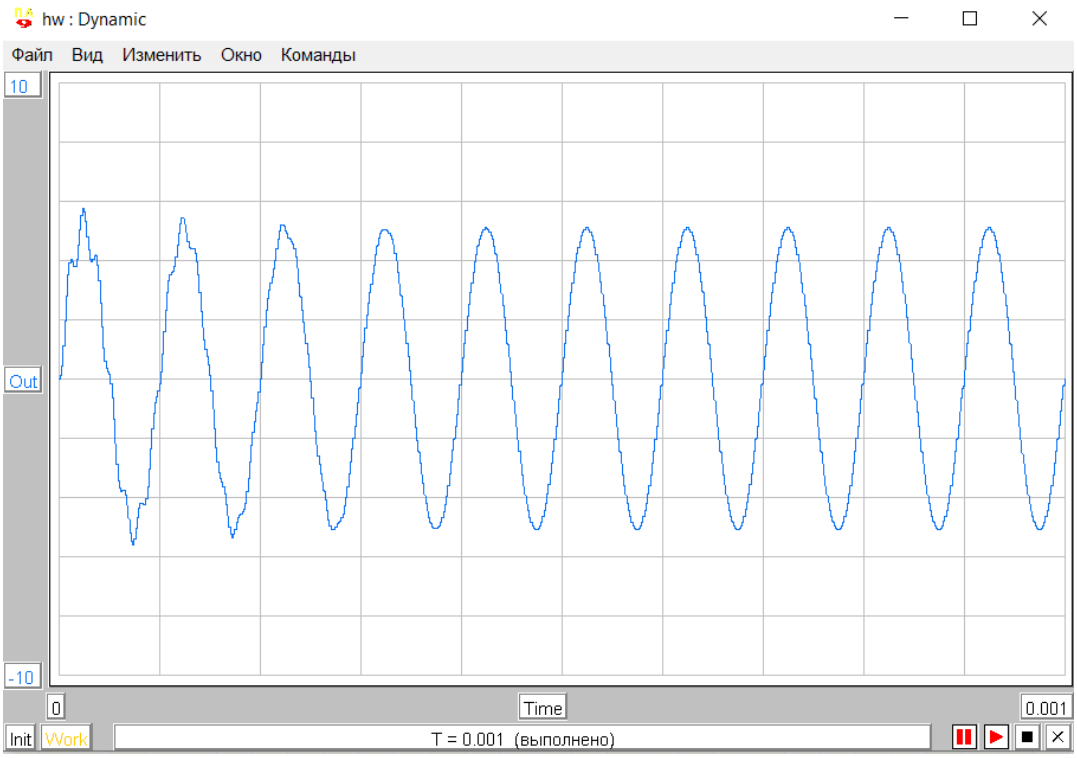

Рисунок 4 — Результат, полученный с помощью ПА9

Исходный код

```
#include <math.h>
#include <string.h>
#include <vector>
#include <string>


typedef std::vector<std::vector<double>> Matrix;
typedef std::vector<double> Vector;

Matrix create_matrix(int rows, int cols) {
    Matrix res;
    res.resize(rows);
    for (int i = 0; i < rows; ++i) {
        res[i].resize(cols, 0);
    }
    return res;
}

void zero(Matrix& m) {
    for (int i = 0; i < m.size(); ++i) {
        std::fill(m[i].begin(), m[i].end(), 0);
    }
}

void print(const Matrix& m) {
    for (int i = 0; i < m.size(); ++i) {
        for (int j = 0; j < m[i].size(); ++j) {
            printf("%-20.8f ", m[i][j]);
        }
        puts("");
    }
}

Vector create_vector(int size) {
    Vector  res;
    res.resize(size, 0);
    return res;
}

void zero(Vector& v) {
    std::fill(v.begin(), v.end(), 0);
}

void print(const Vector& v) {
    for (int i = 0; i < v.size(); ++i) {
        printf("%5f ", v[i]);
    }
    puts("");
}

// it breaks contents of m and b, but they will be recreated on next iteration
anyway...
int gauss(Matrix& m, Vector& b) {
    for (int k = 0; k < m.size(); ++k) {
        if (fabs(m[k][k]) < 1e-17) {
            return 1;
        }
        double diagonal = m[k][k];
        // divide this row by diagonal element
        for (int i = k; i < m.size(); ++i) {
            m[k][i] /= diagonal;
        }
        b[k] /= diagonal;
```

```
            for (int i = k+1; i < m.size(); ++i) {
                double elem = m[i][k];
                for (int j = k; j < m.size(); ++j) {
                    m[i][j] -= elem * m[k][j];
                }
                b[i] -= elem * b[k];
            }
        }
        for (int i = m.size()-2; i >= 0; --i) {
            for (int j = i + 1; j < m.size(); ++j) {
                b[i] -= m[i][j] * b[j];
            }
        }
        return 0;
    }

    void place_element(Matrix& A, int i, int j, double value) {
        if (i >= 0) {
            A[i][i] += value;
        }
        if (j >= 0) {
            A[j][j] += value;
        }
        if (i >= 0 && j >= 0) {
            A[i][j] -= value;
            A[j][i] -= value;
        }
    }

    struct E_AC {
        int i, j;
        double amplitude;
        double freq;
        double phase;

        E_AC(int _i, int _j, double _amplitude, double _freq, double _phase) {
            i = _i; j = _j;
            amplitude = _amplitude;
            freq = _freq;
            phase = _phase;
        }

        double value(double t) {
            return amplitude * sin(freq * t + phase);
        }
    };

    struct Idiode {
        int i, j;
        double It, MFt;
        Idiode(int _i, int _j, double _It, double _MFt) {
            i = _i; j = _j;
            It = _It;
            MFt = _MFt;
        }
        double get_matrix_component(double phi1, double phi2) {
            return It/MFt * exp((phi1-phi2)/MFt);
        }
        double get_vector_component(double phi1, double phi2) {
            return It * (exp((phi1-phi2)/MFt) - 1);
        }
    };

    struct C {
        int i, j;
        double val;
```

```
    C(int _i, int _j, double _val) {
        i = _i; j = _j;
        val = _val;
    }
    double get_matrix_component(double dt) { return val/dt; }
    double get_vector_component(double prev, double phi1, double phi2, double
dt) {
        return val/dt*(phi1 - phi2 - prev);
    }
};

struct R {
    int i, j;
    double val;
    R(int _i, int _j, double _val) {
        i = _i, j = _j;
        val = _val;
    }
    double get_matrix_component() { return 1/val; }
    double get_vector_component(double phi1, double phi2) {
        return (phi1 - phi2) / val;
    }
};

struct L {
    int i, j;
    double val;
    L(int _i, int _j, double _val) {
        i = _i; j = _j;
        val = _val;
    }
    double get_matrix_component(double dt) { return dt/val; }
    double get_vector_component(double prev, double phi1, double phi2, double
dt) {
        return prev + dt/val*(phi1 - phi2);
    }
};

double length(const Vector& v) {
    double m = -1;
    for (auto& e : v) {
        if (fabs(e) > m) {
            m = fabs(e);
        }
    }
    return m;
}

// node -1 is ground.
struct Scheme {
    std::vector<C> Cs;
    std::vector<L> Ls;
    std::vector<R> Rs;
    std::vector<Idiode> Idiodes;
    std::vector<E_AC> Es;
    std::vector<int> target_nodes;

    void evaluate(double T, double delta, double e1, double e2) {
        // open files for results
        std::vector<FILE *> files;
        for (auto node : target_nodes) {
            FILE *f = fopen((std::to_string(node) + ".res").c_str(), "w");
            files.push_back(f);
        }

        // find max_i
```

```
        double max_i = -1;
        for (auto& e : Cs) {
            if (e.i > max_i) { max_i = e.i; }
            if (e.j > max_i) { max_i = e.j; }
        }
        for (auto& e : Ls) {
            if (e.i > max_i) { max_i = e.i; }
            if (e.j > max_i) { max_i = e.j; }
        }
        for (auto& e : Rs) {
            if (e.i > max_i) { max_i = e.i; }
            if (e.j > max_i) { max_i = e.j; }
        }
        for (auto& e : Idiodes) {
            if (e.i > max_i) { max_i = e.i; }
            if (e.j > max_i) { max_i = e.j; }
        }
        for (auto& e : Es) {
            if (e.i > max_i) { max_i = e.i; }
            if (e.j > max_i) { max_i = e.j; }
        }

        int count = max_i + 1 + Es.size();
        Matrix A = create_matrix(count, count);
        Vector b = create_vector(count);

        Vector phi_current = create_vector(count);
        Vector phi_previous = create_vector(count);
        double phi0_pp = 0;

        double dt_prev = 0;
        double dt = 1e-8;

        Vector UC = create_vector(Cs.size());
        Vector IL = create_vector(Ls.size());

        // write initial
        for (int i = 0; i < target_nodes.size(); ++i) {
            fprintf(files[i], "%f %f \n", 0.0f, phi_current[target_nodes[i]]);
        }

        // time iterations
        double t = dt;
        while (t <= T) {
            // set initial
            phi_current = phi_previous;

            bool solved = false;

            // Newton iterations
            int n = 0;
            for (n = 0; n < 7; ++n) {
                // fill matrix
                zero(A);
                for (auto& e : Cs) {
                    place_element(A, e.i, e.j, e.get_matrix_component(dt));
                }
                for (auto& e : Ls) {
                    place_element(A, e.i, e.j, e.get_matrix_component(dt));
                }
                for (auto& e : Rs) {
                    place_element(A, e.i, e.j, e.get_matrix_component());
                }
                for (auto& e : Idiodes) {
                    place_element(A, e.i, e.j,
e.get_matrix_component(phi_current[e.i], phi_current[e.j]));
```

```
                }
                for (int i = 0; i < Es.size(); ++i) {
                    auto E = Es[i];
                    if (E.i >= 0) {
                        A[A.size()-i-1][E.i] -= 1;
                        A[E.i][A.size()-i-1] -= 1;
                    }
                    if (E.j >= 0) {
                        A[A.size()-i-1][E.j] += 1;
                        A[E.j][A.size()-i-1] += 1;
                    }
                }
                // fill vector
                zero(b);
                for (int i = 0; i < Cs.size(); ++i) {
                    auto& e = Cs[i];
                    auto& prev = UC[i];
                    auto component = e.get_vector_component(prev,
phi_current[e.i], phi_current[e.j], dt);
                    b[e.i] += component;
                    b[e.j] -= component;
                }
                for (auto& e : Rs) {
                    auto component = e.get_vector_component(phi_current[e.i],
phi_current[e.j]);
                    b[e.i] += component;
                    b[e.j] -= component;
                }
                for (int i = 0; i < Ls.size(); ++i) {
                    auto& e = Ls[i];
                    auto& prev = IL[i];
                    auto component = e.get_vector_component(prev,
phi_current[e.i], phi_current[e.j], dt);
                    b[e.i] += component;
                    b[e.j] -= component;
                }
                for (auto& e : Idiodes) {
                    auto val = e.get_vector_component(phi_current[e.i],
phi_current[e.j]);
                    b[e.i] += val;
                    b[e.j] -= val;
                }
                for (int i = 0; i < Es.size(); ++i) {
                    auto E = Es[i];
                    b[E.i] -= phi_current[b.size()-i-1];
                    b[E.j] += phi_current[b.size()-i-1];
                    b[b.size()-i-1] += phi_current[E.j]-phi_current[E.i] -
E.value(t);
                }

                // invert b
                for (auto& e : b) {
                    e *= -1;
                }

                // solve
                if (gauss(A, b) != 0) {
                    printf("Error: degenerate matrix, dt: %lf\n", dt);
                    exit(-1);
                }

                // update phi
                for (int i = 0; i < phi_current.size(); ++i) {
                    phi_current[i] += b[i];
                }
```

```
                // max element in deltas
                double l = length(b);

                if (l < delta) {
                    double dt_cur = dt;

                    double ddphi = fabs(dt / (dt + dt_prev) * ((phi_current[0]
- phi_previous[0]) - dt / dt_prev * (phi_previous[0] - phi0_pp)));
                    if (ddphi < e1) {
                        dt *= 2;
                    } else if (ddphi > e2) {
                        // ignore this step
                        break;
                    }

                    // Update IL, UC
                    for (int i = 0; i < Cs.size(); ++i) {
                        auto& e = Cs[i];
                        auto phii = e.i >= 0 ? phi_current[e.i] : 0;
                        auto phij = e.j >= 0 ? phi_current[e.j] : 0;
                        UC[i] = phii - phij;
                    }
                    for (int i = 0; i < Ls.size(); ++i) {
                        auto& e = Ls[i];
                        auto phii = e.i >= 0 ? phi_current[e.i] : 0;
                        auto phij = e.j >= 0 ? phi_current[e.j] : 0;
                        IL[i] += dt_cur / e.val * (phii - phij);
                    }

                    // update info
                    dt_prev = dt_cur;
                    t += dt_cur;

                    phi0_pp = phi_previous[0];
                    phi_previous = phi_current;

                    solved = true;

                    // write results
                    for (int i = 0; i < target_nodes.size(); ++i) {
                        fprintf(files[i], "%f %f \n", t,
phi_current[target_nodes[i]]);
                    }

                    break;
                }
            }

            if (!solved) {
                dt /= 2;
            }
        }

        // close files
        for (auto f : files) {
            fclose(f);
        }
    }
};

int main() {
    Scheme scheme;

    // add elements to the scheme. Node -1 is ground.
    scheme.Es.push_back(E_AC(-1, 0, 10, 2*M_PI*1e4, 0));
    scheme.Rs.push_back(R(0, 1, 1e3));
```

```
    scheme.Cs.push_back(C(0, 1, 1e-6));
    scheme.Ls.push_back(L(1, 2, 0.00001));
    scheme.Cs.push_back(C(2, -1, 1e-6));
    scheme.Rs.push_back(R(2, -1, 1e3));
    scheme.Rs.push_back(R(1, 3, 20));
    scheme.Rs.push_back(R(3, 4, 1e6));
    scheme.Cs.push_back(C(3, 4, 2e-9));
    scheme.Idiodes.push_back(Idiode(3, 4, 1e-12,
    0.026));
    scheme.Es.push_back(E_AC(-1, 4, 5, 0, M_PI/2));

    scheme.target_nodes.push_back(2);

    scheme.evaluate(1e-3, 1e-4, 1e-4, 5e-4);
    return 0;
}
```