Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего профессионального 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	Шлюков Алексей Павлович
Группа:	РК6-74Б
Тип задания:	Домашнеее задание
Тема:	Методы формирования математических моделей систем

Студент _____ Шлюков А. П.
подпись, дата Фамилия, И.О.

Преподаватель _____ Боровкова Н.М.
подпись, дата Фамилия, И.О.

Москва, 2025

Содержание

1 Задание

Вариант 19 7

Необходимо составить математическую модель схемы (рис. 6) расширенным узловым методом (7 вариант), разработать программу для анализа переходных про- цессов и сравнить результаты, полученные с помощью программы, с результатами динамического анализа данной схемы в ПА9.

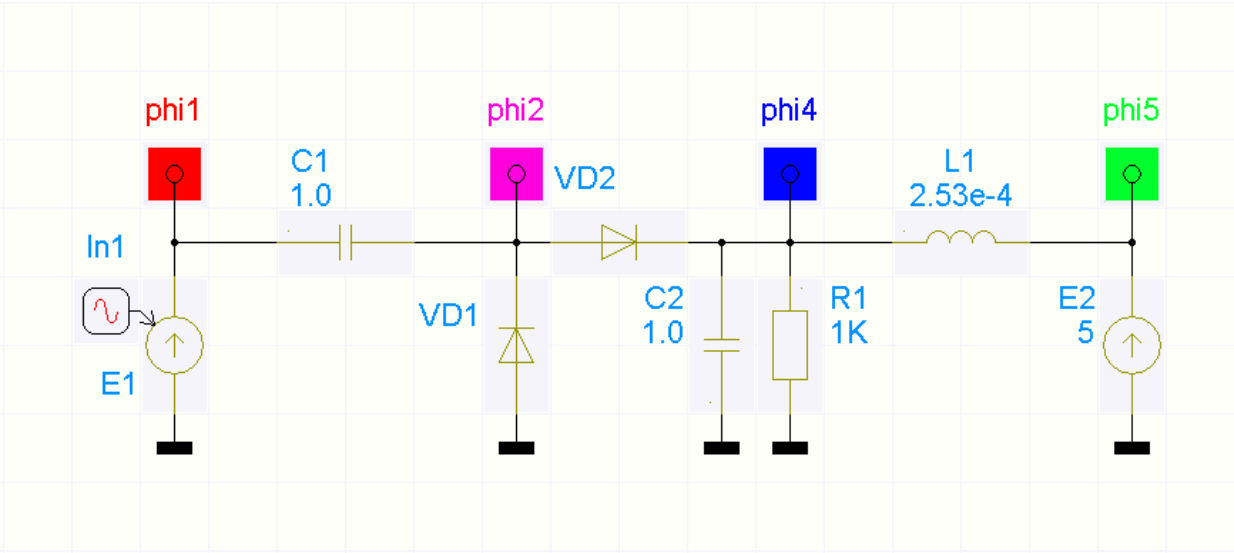

Рис. 1. Схема №19

Параметры системы:

- $E1 = 10 \cdot sin(10^4 \cdot t)$ В;
- $E2 = 5$ В;
- $R1 = 1000$;
- $C1 = C2 = 10^{-6}$ Ф;
- $L = 2.53 \cdot 10^{-4}$ Гн;
- параметры диодов VD1 и VD2 одинаковы:
  - $I_t = 10^{-12}$ А;
  - $C_b = 2 \cdot 10^{-12}$ Ф;
  - $m\varphi_t = 0.026$;
  - $R_u = 10^6$ Ом;
  - $R_b = 20$ Ом.

Параметры расчета:

- время анализа $T_k = 10^{-3}$ с;
- требуемая точность $\varepsilon = 10^{-2}$;
- максимальное число итераций метода Ньютона n $= 7$;
- начальный шаг интегрирования $S_{st} = 10^{-8}$;
- минимальный шаг интегрирования $S_{st} = 10^{-12}$;
- максимальный шаг интегрирования $S_{st} = 10^{-4}$.

2 Составление эквивалентной схемы

Для составления математической модели системы необходимо выполнить замену диода его эвивалентной схемой, состоящей из базовых элементов. Новая схема будет иметь вид (рис.2).

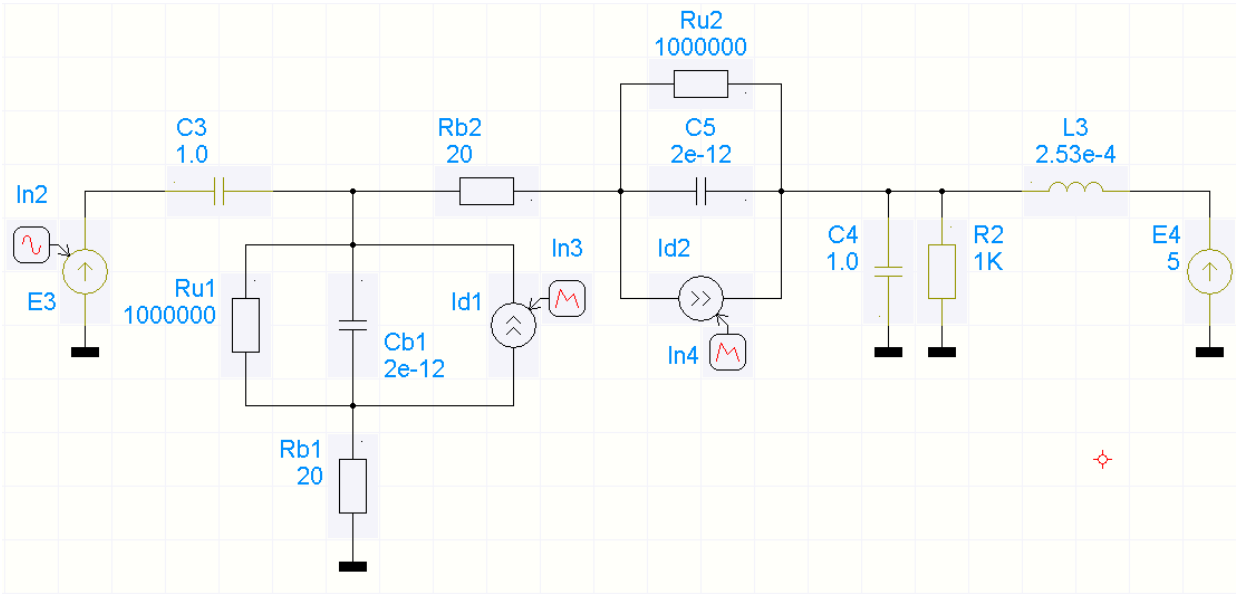

Рис. 2. Эквивалентная схема

При этом ток через диод рассчитывается по следующей формуле:

$$I_d = I_t \left(e^{\frac{U_d}{m\varphi_t}} - 1 \right)$$

где U_d — напряжение на диоде, I_t и $m\varphi_t$ — параметры диода, описанные ранее.

3 Составление математической модели

Преобразованная схема с указанием узлов рисунке 3.

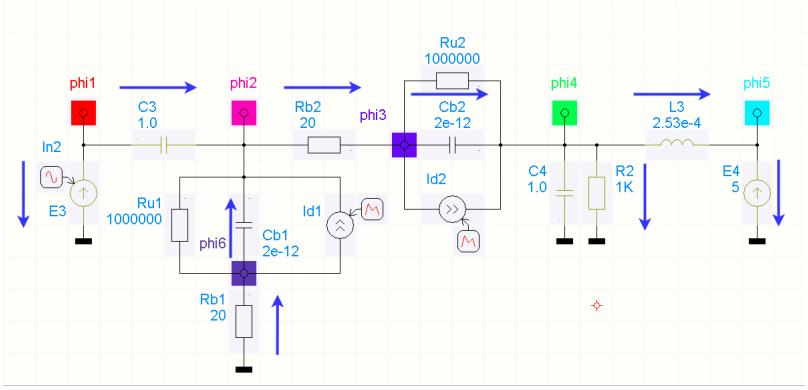

Рис. 3. Направления токов с цепи

Базис математической модели, реализованной расширенным узловым методом, составляют производные переменных состояния, переменные состояния, узловые потенциалов, ток идеальных источников ЭДС. Вектор-базис и вектор невязок:

$$X = \begin{bmatrix} \Delta\frac{dU_{C_1}}{dt} \\ \Delta\frac{dU_{C_2}}{dt} \\ \Delta\frac{dU_{C_{b1}}}{dt} \\ \Delta\frac{dU_{C_{b2}}}{dt} \\ \Delta\frac{dI_L}{dt} \\ \Delta U_{C_1} \\ \Delta U_{C_2} \\ \Delta U_{C_{b1}} \\ \Delta U_{C_{b1}} \\ \Delta I_L \\ \Delta\varphi_1 \\ \Delta\varphi_2 \\ \Delta\varphi_3 \\ \Delta\varphi_4 \\ \Delta\varphi_5 \\ \Delta\varphi_6 \\ \Delta I_{E_1} \\ \Delta I_{E_2} \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{dU_{C_1}}{dt} - \frac{U_{C_1} - U_{C_1}^{n-1}}{\Delta t} \\ \frac{dU_{C_2}}{dt} - \frac{U_{C_2} - U_{C_2}^{n-1}}{\Delta t} \\ \frac{dU_{C_{b1}}}{dt} - \frac{U_{C_{b1}} - U_{C_{b1}}^{n-1}}{\Delta t} \\ \frac{dU_{C_{b2}}}{dt} - \frac{U_{C_{b2}} - U_{C_{b2}}^{n-1}}{\Delta t} \\ \frac{dI_L}{dt} - \frac{I_L - I_L^{n-1}}{\Delta t} \\ U_{C_1} - (\varphi_1 - \varphi_2) \\ U_{C_2} - \varphi_4 \\ U_{Cb_1} - (\varphi_6 - \varphi_2) \\ U_{Cb_2} - (\varphi_3 - \varphi_4) \\ L\frac{I_L}{dt} - (\varphi_4 - \varphi_5) \\ I_{E_1} + C_1\frac{dU_{C_1}}{dt} \\ -C_1\frac{dU_{C_1}}{dt} - C_{b1}\frac{dU_{C_{b1}}}{dt} - \frac{U_{C_{b1}}}{R_u} - I_T(e^{\frac{U_{C_{b1}}}{m\varphi_t}} - 1) + \frac{\varphi_2 - \varphi_3}{R_{b2}} \\ -\frac{\varphi_2 - \varphi_3}{R_{b2}} + C_{b2}\frac{dU_{C_{b2}}}{dt} + \frac{U_{C_{b2}}}{R_u} + I_T(e^{\frac{U_{C_{b2}}}{m\varphi_t}} - 1) \\ -C_{b2}\frac{dU_{C_{b2}}}{dt} - \frac{U_{C_{b2}}}{R_u} - I_T(e^{\frac{U_{C_{b2}}}{m\varphi_t}} - 1) + C_2\frac{dU_{C2}}{dt} + \frac{\varphi_4}{R} + I_L \\ -I_L + I_{E_2} \\ \frac{-\varphi_6}{R_{b2}} + C_{b1}\frac{dU_{C_{b1}}}{dt} + \frac{U_{C_{b1}}}{R_u} + I_T(e^{\frac{U_{C_{b1}}}{m\varphi_t}} - 1) \\ E_1 - \varphi_1 \\ E_2 - \varphi_5 \end{bmatrix}$$

На рисунке 4 приведена матрица Якоби (Y) математической модели: Таким образом

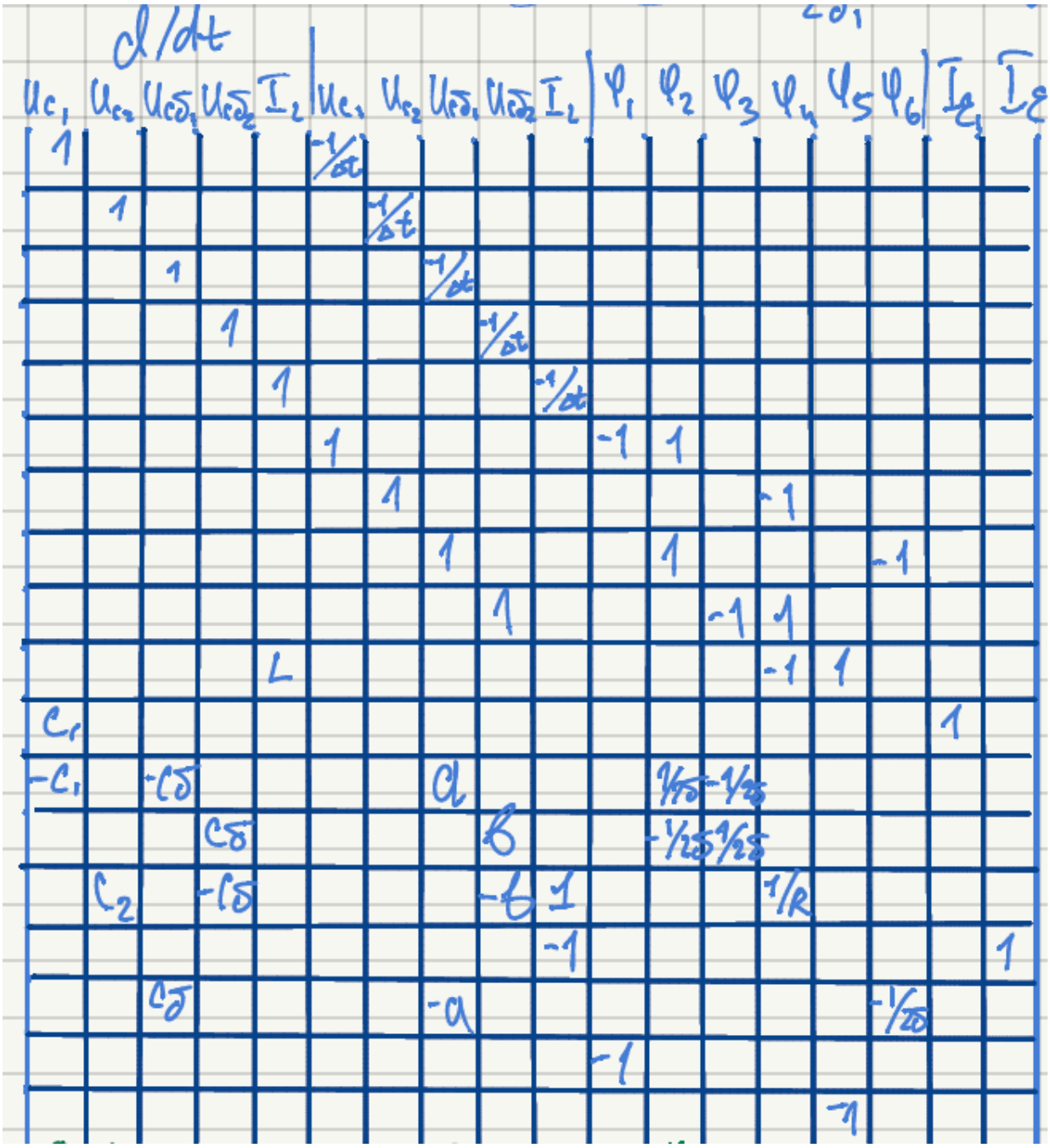

Рис. 4. Якобиан математической модели

получаем систему нелинейных алгебраических уравнений

$$YX = -B,$$

для решения которой ввоспользуемся методом Ньютона

В матрице Якоби (Y) математической модели коэффициенты "a" и "b" имеют следующие значения:

a = -1.0/R_U - I_T/MFT * exp(uCb1/MFT)

Где:

R_U = 1,000,000 Ом - сопротивление утечки диода
I_T = 10^{-12} А - тепловой ток диода
MFT = 0,026 В - температурный потенциал (температурное напряжение)
uCb1 - напряжение на барьерной емкости диода VD1
exp(x) - экспоненциальная функция

b = 1.0/R_U + I_T/MFT * exp(uCb2/MFT)

Где:

R_U = 1,000,000 Ом - сопротивление утечки диода
I_T = 10^{-12} А - тепловой ток диода
MFT = 0,026 В - температурный потенциал
uCb2 - напряжение на барьерной емкости диода VD2

4 Алгоритм расчета переходных процессов

Блок-схема алгоритма решения задачи представлена на рисунке 5

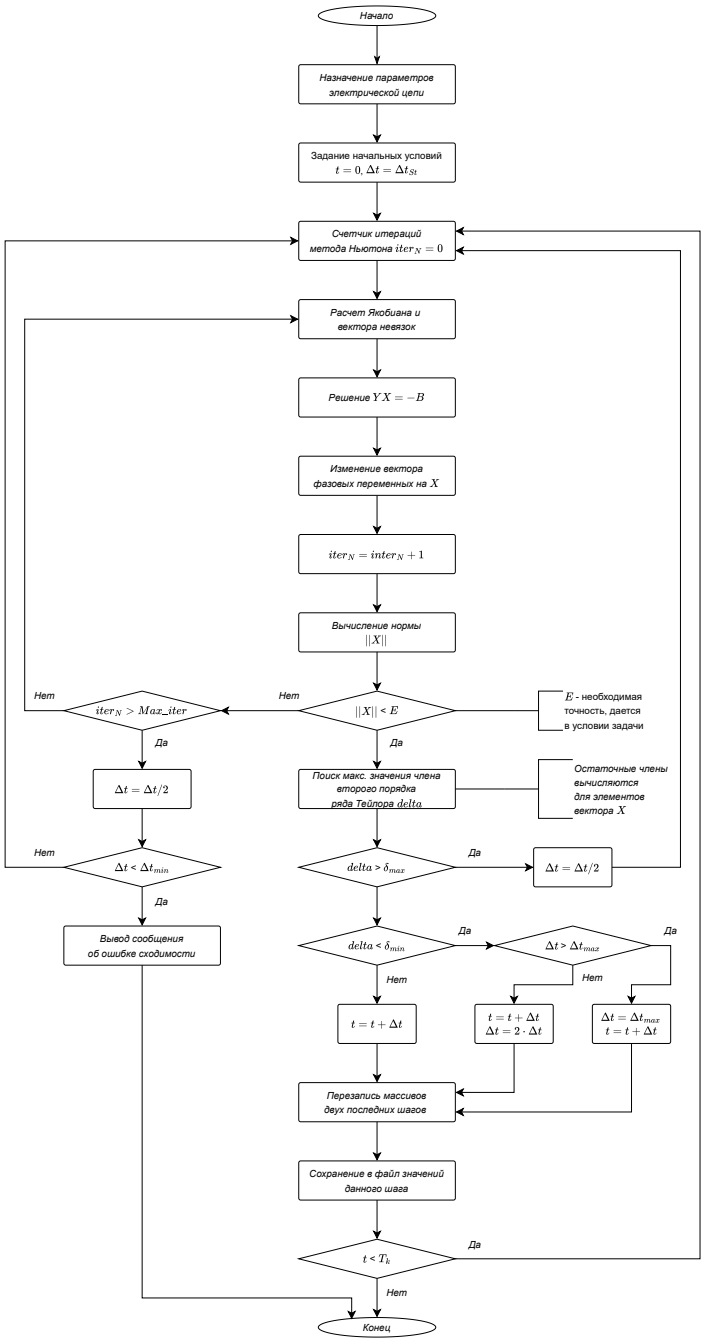

Рис. 5. Блок-схема алгоритма

5 Результаты вычислений

Проведем динамический анализ схемы 6 в ПА9.

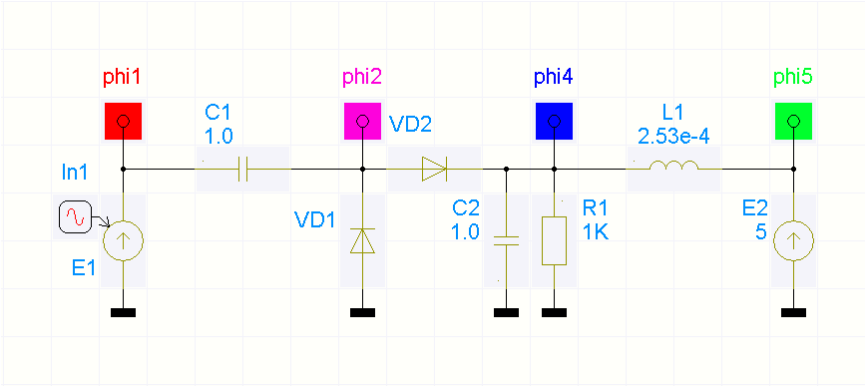

Рис. 6. Схема №19

Статистика расчетов в приложении показывает, что для решения схемы потребовалось 2743 итерации. Для решения задачи собственной программой потребовалось 2565914 итераций.

На рисунках 7 - 11 представлены графики функций узловых потенциалов от времени, полученные в ПА9 и с помощью программы.

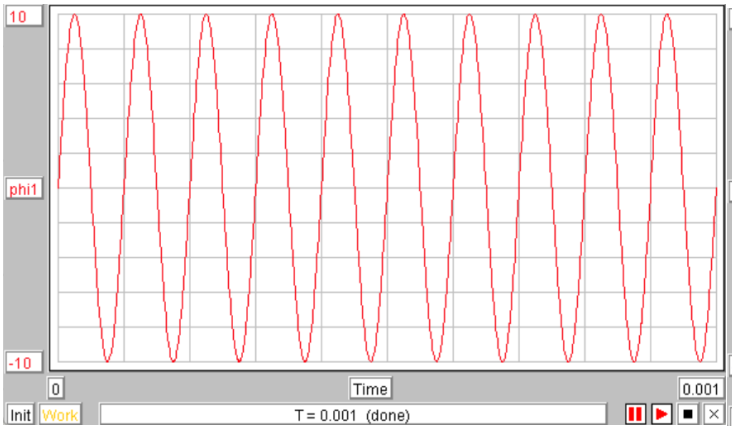

Рис. 7. Потенциал узла 1

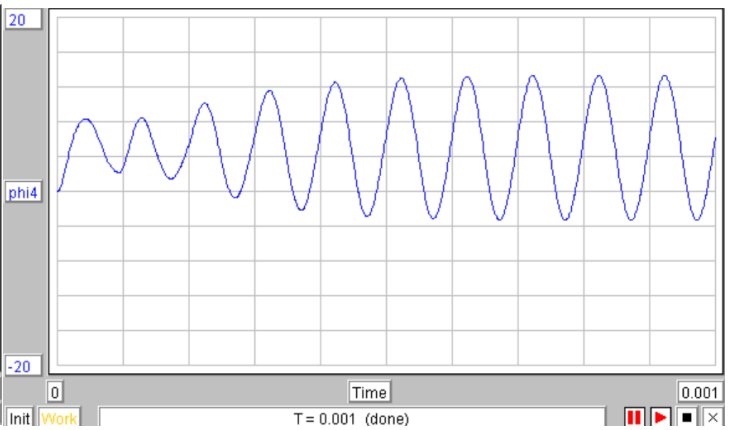

Рис. 9. Потенциал узла 4

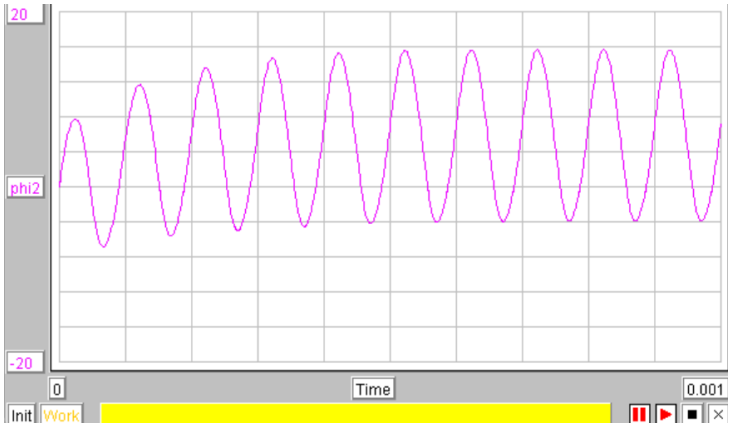

Рис. 8. Потенциал узла 2

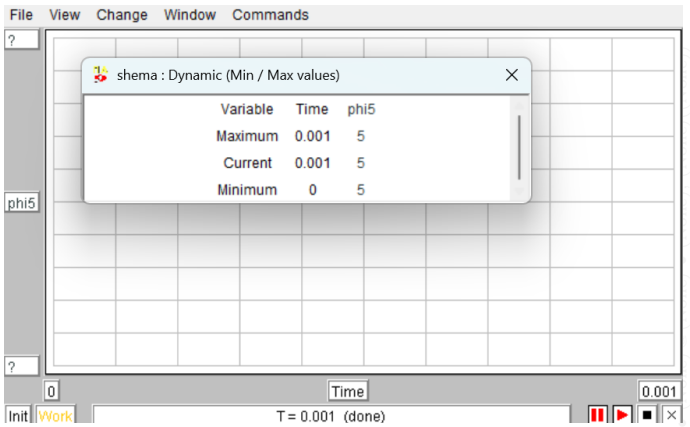

Рис. 10. Потенциал узла 5

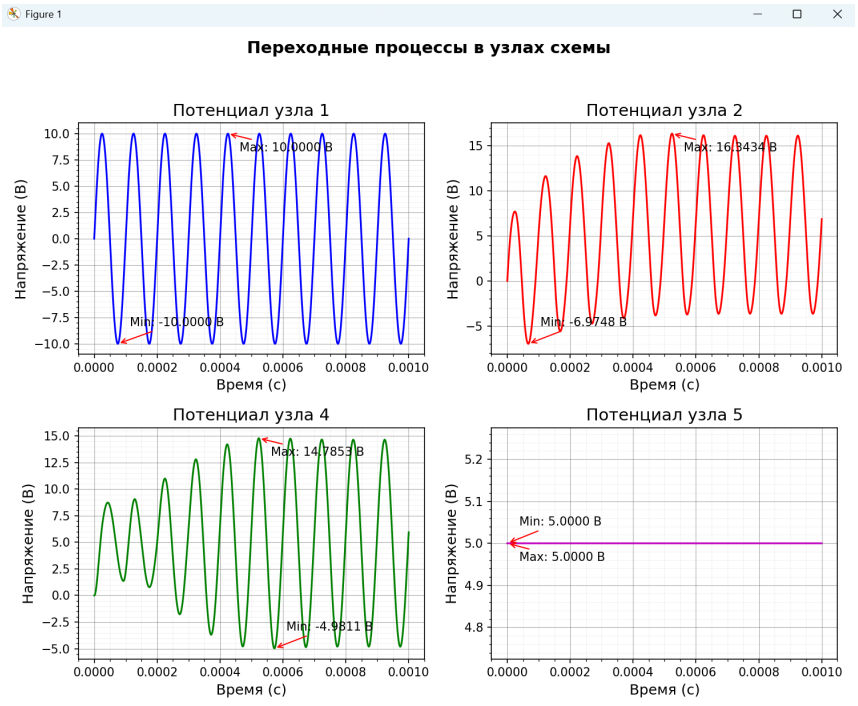

Рис. 11. Вывод

На рисунках 12 - 15 представлены графики напряжения на конденсаторах C1 и C2 и график тока через катушку индуктивности L

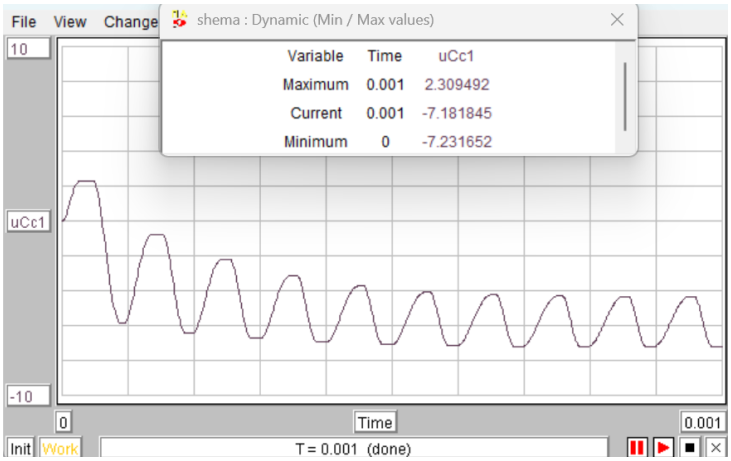

Рис. 12. Напряжение на конденсаторе C1

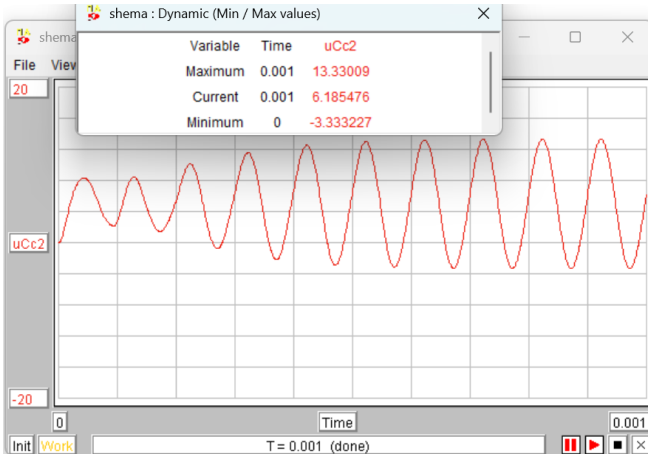

Рис. 13. Напряжение на конденсаторе C2

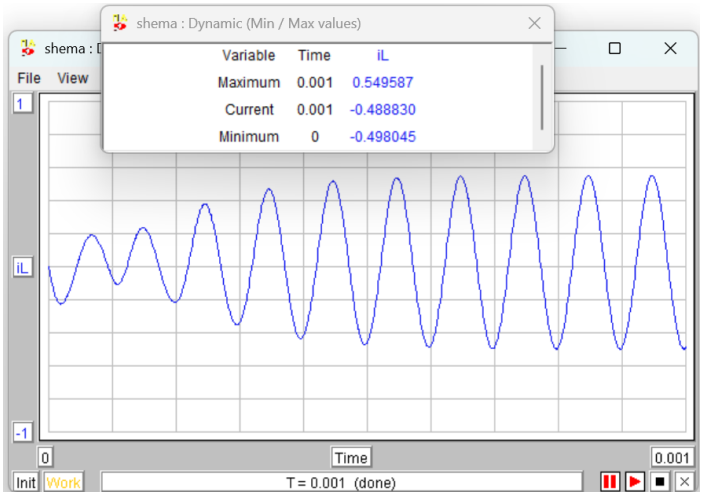

Рис. 14. Ток через индуктивность

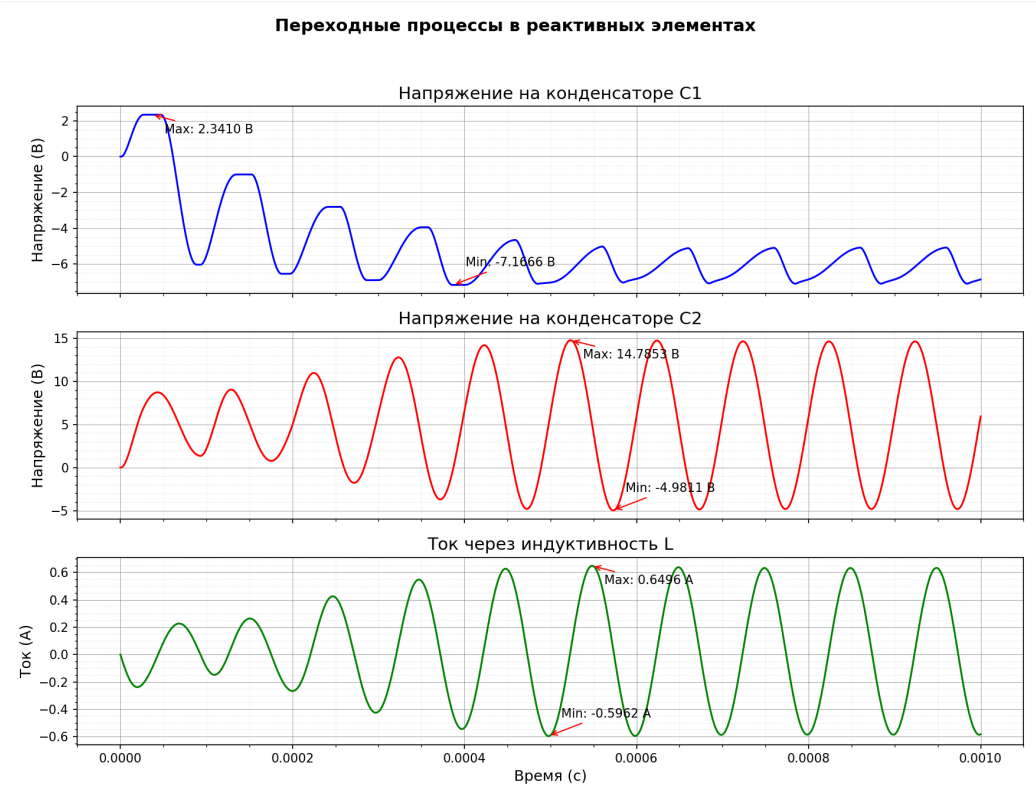

Рис. 15. Вывод программы

7 Листинг программы

Листинг 1. Main.kt

```
fun main() {
    clearFiles()

    var timeDemon = TimeDemon(
        currT = Data.TIME_START,
        deltaT = Data.START_DELTA_TIME
    )
    var prevDeltaT = timeDemon.deltaT

    var initialApproximation = PhaseVariables()
    var pvPrev = PhaseVariables()
    var pvPrevPrev = PhaseVariables()

    val results = ResultLists()

    val addResults: (PhaseVariables, Double) -> Unit = { pv, time ->
        results.phi1List.add(pv.phi1)
        results.phi2List.add(pv.phi2)
        results.phi4List.add(pv.phi4)
        results.phi5List.add(pv.phi5)
        results.timeList.add(time)
    }
    addResults(initialApproximation, timeDemon.currT)

    var prevStateVariables = PrevStateVariables(
        uC1 = 0.0,
        uC2 = 0.0,
        uCb1 = 0.0,
        uCb2 = 0.0,
        iL = 0.0
    )

    var iteration = 0
    while (timeDemon.currT < Data.TIME_END) {
        var newtonMethodResults = newtonMethod(
            td = timeDemon,
            initApprox = initialApproximation,
            prevStateVariables = prevStateVariables
        )
        iteration++

        when (newtonMethodResults.isSuccessful) {
```

```
43              true -> {
44                  val prevTimeDemon = timeDemon.copy()
45                  timeDemon = calculateDeltaT(
46                      td = timeDemon,
47                      prevDeltaT = prevDeltaT,
48                      pv = newtonMethodResults.phaseVariables,
49                      pvPrev = pvPrev,
50                      pvPrevPrev = pvPrevPrev
51                  )
52
53                  if (!timeDemon.success) {
54                      continue
55                  }
56
57                  initialApproximation =
                        predictPhaseVariables(newtonMethodResults.phaseVariables, pvPrev)
58
59                  with(newtonMethodResults.phaseVariables) {
60                      prevStateVariables = PrevStateVariables(
61                          uC1 = uC1,
62                          uC2 = uC2,
63                          uCb1 = uCb1,
64                          uCb2 = uCb2,
65                          iL = iL
66                      )
67                  }
68
69                  pvPrevPrev = pvPrev.copy()
70                  pvPrev = newtonMethodResults.phaseVariables.copy()
71                  if (timeDemon.currT >= Data.TIME_END - timeDemon.deltaT ) {
72                      println("$iteration, t = ${timeDemon.currT}")
73  // addResults(newtonMethodResults.phaseVariables, prevTimeDemon.currT)
74                  }
75              }
76
77              false -> {
78                  timeDemon = deltaTReduction(timeDemon)
79
80                  if (timeDemon.deltaT / 2.0 < Data.MINIMAL_STEP) throw
                        TimeStepException()
81              }
82          }
83      }
84
85      with(results) {
86          printToFile(FileData.PHI1_FILE, phi1List.toDoubleArray())
```

```
87          printToFile(FileData.PHI2_FILE, phi2List.toDoubleArray())
88          printToFile(FileData.PHI4_FILE, phi4List.toDoubleArray())
89          printToFile(FileData.PHI5_FILE, phi5List.toDoubleArray())
90          printToFile(FileData.T_FILE, timeList.toDoubleArray())
91      }
92  }
```

Листинг 2. DataHolders.kt

```
1  import kotlin.math.PI
2  import kotlin.math.sin
3
4  data object Data {
5      const val START_DELTA_TIME = 1e-11
6      const val TIME_START = 0.0
7      const val TIME_END = 1e-3
8      const val MINIMAL_STEP = 1e-18
9      const val MAXIMAL_STEP = 1e-5
10
11     const val N = 18
12     const val NEWTON_STEPS = 7
13
14     const val EPSILON = 5e-2
15     const val DELTA_1 = 1e-2
16     const val DELTA_2 = 1e-3
17 }
18
19 data object Components {
20     const val L = 2.53e-4
21     const val C1 = 1e-6
22     const val C2 = 1e-6
23     const val I_T = 1e-12
24     const val C_B = 2e-12
25     const val MFT = 0.026
26     const val R_U = 1_000_000.0
27     const val R_B = 20.0
28     const val R = 1_000.0
29     val getCurrentE1: (Double) -> Double = { t -> 10.0 * sin(2.0 * PI / 1e-4 * t) }
30     const val E2 = 5.0
31 }
32
33 data object FileData {
34     const val PHI1_FILE = "phi1.txt"
35     const val PHI2_FILE = "phi2.txt"
36     const val PHI4_FILE = "phi4.txt"
37     const val PHI5_FILE = "phi5.txt"
```

```
38      const val T_FILE = "t.txt"
39  }
40
41  data class PhaseVariables(
42      val dUC1dt: Double = 0.0,
43      val dUC2dt: Double = 0.0,
44      val dUCB1dt: Double = 0.0,
45      val dUCB2dt: Double = 0.0,
46      val dILdt: Double = 0.0,
47
48      val uC1: Double = 0.0,
49      val uC2: Double = 0.0,
50      val uCb1: Double = 0.0,
51      val uCb2: Double = 0.0,
52      val iL: Double = 0.0,
53
54      val phi1: Double = 0.0,
55      val phi2: Double = 0.0,
56      val phi3: Double = 0.0,
57      val phi4: Double = 0.0,
58      val phi5: Double = 5.0,
59      val phi6: Double = 0.0,
60
61      val iE1: Double = 0.0,
62      val iE2: Double = 0.0,
63  ) {
64      operator fun plus(other: PhaseVariables): PhaseVariables = PhaseVariables(
65          dUC1dt = this.dUC1dt + other.dUC1dt,
66          dUC2dt = this.dUC2dt + other.dUC2dt,
67          dUCB1dt = this.dUCB1dt + other.dUCB1dt,
68          dUCB2dt = this.dUCB2dt + other.dUCB2dt,
69          dILdt = this.dILdt + other.dILdt,
70
71          uC1 = this.uC1 + other.uC1,
72          uC2 = this.uC2 + other.uC2,
73          uCb1 = this.uCb1 + other.uCb1,
74          uCb2 = this.uCb2 + other.uCb2,
75          iL = this.iL + other.iL,
76
77          phi1 = this.phi1 + other.phi1,
78          phi2 = this.phi2 + other.phi2,
79          phi3 = this.phi3 + other.phi3,
80          phi4 = this.phi4 + other.phi4,
81          phi5 = this.phi5 + other.phi5,
82          phi6 = this.phi6 + other.phi6,
83
```

```
84          iE1 = this.iE1 + other.iE1,
85          iE2 = this.iE2 + other.iE2,
86      )
87
88      fun convertFromPVToArray(): DoubleArray = doubleArrayOf(
89          this.dUC1dt,
90          this.dUC2dt,
91          this.dUCB1dt,
92          this.dUCB2dt,
93          this.dILdt,
94
95          this.uC1,
96          this.uC2,
97          this.uCb1,
98          this.uCb2,
99          this.iL,
100
101         this.phi1,
102         this.phi2,
103         this.phi3,
104         this.phi4,
105         this.phi5,
106         this.phi6,
107
108         this.iE1,
109         this.iE2
110     )
111
112     companion object {
113         fun convertFromArrayToPV(results: DoubleArray): PhaseVariables {
114             if (results.size != Data.N) throw ConversionException()
115             return PhaseVariables(
116                 dUC1dt = results[0],
117                 dUC2dt = results[1],
118                 dUCB1dt = results[2],
119                 dUCB2dt = results[3],
120                 dILdt = results[4],
121                 uC1 = results[5],
122                 uC2 = results[6],
123                 uCb1 = results[7],
124                 uCb2 = results[8],
125                 iL = results[9],
126                 phi1 = results[10],
127                 phi2 = results[11],
128                 phi3 = results[12],
129                 phi4 = results[13],
```

```
                    phi5 = results[14],
                    phi6 = results[15],
                    iE1 = results[16],
                    iE2 = results[17]
                )
            }
        }
}

data class PrevStateVariables(
    val uC1: Double,
    val uC2: Double,
    val uCb1: Double,
    val uCb2: Double,
    val iL: Double
)

data class NewtonMethodResults(
    val isSuccessful: Boolean,
    val phaseVariables: PhaseVariables = PhaseVariables()
)

data class TimeDemon(
    val currT: Double,
    val deltaT: Double,
    val success: Boolean = true
)

data class ResultLists(
    val phi1List: MutableList<Double> = mutableListOf(),
    val phi2List: MutableList<Double> = mutableListOf(),
    val phi4List: MutableList<Double> = mutableListOf(),
    val phi5List: MutableList<Double> = mutableListOf(),
    val timeList: MutableList<Double> = mutableListOf(),
)
```

Листинг 3. Utilities.kt

```
import java.io.File
import kotlin.math.abs
import kotlin.math.exp
import kotlin.math.sqrt

fun gauss(a: Array<DoubleArray>, b: DoubleArray): DoubleArray {
    if (a.size != b.size) throw MatrixDimensionException()

```

```
    val x = DoubleArray(b.size) { 0.0 }

    for (j in 0..<a.size − 1) {
        val ajj = a[j][j]
        for (i in j + 1..<a.size) {
            val coeff = a[i][j] / ajj
            for (k in j..<a.size) {
                a[i][k] −= coeff ∗ a[j][k]
            }
            b[i] −= coeff ∗ b[j]
        }
    }
    for (i in a.size − 1 downTo 0) {
        var sum = 0.0
        for (ind in a.size − 1 downTo i + 1) {
            sum += a[i][ind] ∗ x[ind]
        }
        x[i] = (b[i] − sum) / a[i][i]
    }
    return x
}

fun createJacobiMatrix(deltaT: Double, uCb1: Double, uCb2: Double): Array<DoubleArray> {
    val jacobi = Array<DoubleArray>(Data.N) { DoubleArray(Data.N) { 0.0 } }

    with(Components) {
        val a = −1.0 / R_U − I_T / MFT ∗ exp(uCb1 / MFT)
        val b = 1.0 / R_U + I_T / MFT ∗ exp(uCb2 / MFT)

        jacobi[0][0] = 1.0
        jacobi[0][5] = −1.0 / deltaT
        jacobi[1][1] = 1.0
        jacobi[1][6] = −1.0 / deltaT
        jacobi[2][2] = 1.0
        jacobi[2][7] = −1.0 / deltaT
        jacobi[3][3] = 1.0
        jacobi[3][8] = −1.0 / deltaT
        jacobi[4][4] = 1.0
        jacobi[4][9] = −1.0 / deltaT

        jacobi[5][5] = 1.0
        jacobi[5][10] = −1.0
        jacobi[5][11] = 1.0

        jacobi[6][6] = 1.0
        jacobi[6][13] = −1.0
```

```
55
56          jacobi[7][7] = 1.0
57          jacobi[7][11] = 1.0
58          jacobi[7][15] = −1.0
59
60          jacobi[8][8] = 1.0
61          jacobi[8][12] = −1.0
62          jacobi[8][13] = 1.0
63
64          jacobi[9][4] = L
65          jacobi[9][13] = −1.0
66          jacobi[9][14] = 1.0
67
68          jacobi[10][0] = C1
69          jacobi[10][16] = 1.0
70
71          jacobi[11][0] = −C1
72          jacobi[11][2] = −C_B
73          jacobi[11][7] = a
74          jacobi[11][11] = 1.0 / R_B
75          jacobi[11][12] = −1.0 / R_B
76
77          jacobi[12][3] = C_B
78          jacobi[12][8] = b
79          jacobi[12][11] = −1.0 / R_B
80          jacobi[12][12] = 1.0 / R_B
81
82          jacobi[13][1] = C2
83          jacobi[13][3] = −C_B
84          jacobi[13][8] = −b
85          jacobi[13][9] = 1.0
86          jacobi[13][13] = 1.0 / R
87
88          jacobi[14][9] = −1.0
89          jacobi[14][17] = 1.0
90
91          jacobi[15][2] = C_B
92          jacobi[15][7] = −a
93          jacobi[15][15] = 1.0 / R_B
94
95          jacobi[16][10] = −1.0
96          jacobi[17][14] = −1.0
97      }
98      return jacobi
99  }
100
```

```
fun createVector(
    td: TimeDemon,
    pvApprox: PhaseVariables,
    prevStateVariables: PrevStateVariables
): DoubleArray {
    // распаковкаобъектаприближенийфазовыхпеременныхвотдельныезначения
    val (dUC1dt, dUC2dt, dUCB1dt, dUCB2dt, dILdt, uC1, uC2,
        uCB1, uCB2, iL, phi1, phi2, phi3, phi4, phi5, phi6, iE1, iE2) = pvApprox

    val (uC1Prev, uC2Prev, uCB1Prev, uCB2Prev, iLPrev) = prevStateVariables

    val vector = DoubleArray(Data.N) { 0.0 }

    val deltaT = td.deltaT

    with(Components) {
        vector[0] = dUC1dt − (uC1 − uC1Prev) / deltaT
        vector[1] = dUC2dt − (uC2 − uC2Prev) / deltaT
        vector[2] = dUCB1dt − (uCB1 − uCB1Prev) / deltaT
        vector[3] = dUCB2dt − (uCB2 − uCB2Prev) / deltaT
        vector[4] = dILdt − (iL − iLPrev) / deltaT

        vector[5] = uC1 − (phi1 − phi2)
        vector[6] = uC2 − phi4
        vector[7] = uCB1 − (phi6 − phi2)
        vector[8] = uCB2 − (phi3 − phi4)
        vector[9] = L ∗ dILdt − (phi4 − phi5)

        vector[10] = iE1 + C1 ∗ dUC1dt
        vector[11] = −C1 ∗ dUC1dt − C_B ∗ dUCB1dt − uCB1 / R_U − I_T ∗
            (exp(uCB1 / MFT) − 1.0) + (phi2 − phi3) / R_B
        vector[12] = −(phi2 − phi3) / R_B + C_B ∗ dUCB2dt + uCB2 / R_U + I_T ∗
            (exp(uCB2 / MFT) − 1.0)
        vector[13] = −(C_B ∗ dUCB2dt + uCB2 / R_U + I_T ∗ (exp(uCB2 / MFT) −
            1.0)) + C2 ∗ dUC2dt + phi4 / R + iL
        vector[14] = −iL + iE2
        vector[15] = phi6 / R_B + C_B ∗ dUCB1dt + uCB1 / R_U + I_T ∗ (exp(uCB1 /
            MFT) − 1.0)

        vector[16] = getCurrentE1(td.currT) − phi1
        vector[17] = E2 − phi5
    }

    return vector
}
```

```
143 fun newtonMethod(
144     td: TimeDemon, initApprox: PhaseVariables,
145     prevStateVariables: PrevStateVariables
146 ): NewtonMethodResults {
147     var n = 0
148 
149     var currApprox = initApprox.copy()
150     while (n < Data.NEWTON_STEPS) {
151 
152         val jacobi = createJacobiMatrix(
153             deltaT = td.deltaT,
154             uCb1 = initApprox.uCb1,
155             uCb2 = initApprox.uCb2
156         )
157         val vectorForNewton = createVector(
158             td = td,
159             pvApprox = currApprox,
160             prevStateVariables = prevStateVariables
161         ).apply {
162             for (i in this.indices) {
163                 this[i] *= -1.0
164             }
165         }
166 
167         val gaussResults = gauss(jacobi, vectorForNewton)
168         val deltas = PhaseVariables.convertFromArrayToPV(gaussResults)
169         var newApprox = currApprox + deltas
170         currApprox = newApprox
171 
172         if (calculateVectorNorm(deltas) < Data.EPSILON) break
173 
174         n++
175     }
176     if (n >= 7) {
177         return NewtonMethodResults(false)
178     }
179 
180     return NewtonMethodResults(true, currApprox)
181 }
182 
183 private fun findMaxValue(values: DoubleArray): Double {
184     var max = 0.0
185     values.forEach { component ->
186         if (component > max) max = component
187     }
188     return max
```

```
189 }
190
191 private fun calculateVectorNorm(pv: PhaseVariables): Double {
192     val array = pv.convertFromPVToArray()
193     val result = array.fold(0.0) { acc: Double, value -> acc + value * value }
194     return sqrt(result)
195 }
196
197 fun predictPhaseVariables(
198     pvPrev: PhaseVariables,
199     pvPrevPrev: PhaseVariables,
200 ): PhaseVariables {
201     val prevPrevArray = pvPrevPrev.convertFromPVToArray()
202     val newValues = pvPrev
203         .convertFromPVToArray()
204         .mapIndexed { i, value -> 2.0 * value - prevPrevArray[i] }
205         .toDoubleArray()
206     return PhaseVariables.convertFromArrayToPV(newValues)
207 }
208
209 fun calculateDeltaT(
210     td: TimeDemon, prevDeltaT: Double,
211     pv: PhaseVariables, pvPrev: PhaseVariables, pvPrevPrev: PhaseVariables
212 ): TimeDemon {
213     val currT = td.currT
214     val deltaT = td.deltaT
215
216     val secondRemainder: (Double, Double, Double) -> Double = { curr, prev, prevPrev ->
217         abs((curr * prevDeltaT - prev * (deltaT + prevDeltaT) + prevPrev * deltaT) / (2 *
                prevDeltaT))
218     }
219
220     val delta: Double = findMaxValue(
221         PhaseVariables(
222             phi1 = secondRemainder(pv.phi1, pvPrev.phi1, pvPrevPrev.phi1),
223             phi2 = secondRemainder(pv.phi2, pvPrev.phi2, pvPrevPrev.phi2),
224             phi3 = secondRemainder(pv.phi3, pvPrev.phi3, pvPrevPrev.phi3),
225             phi4 = secondRemainder(pv.phi4, pvPrev.phi4, pvPrevPrev.phi4),
226             phi5 = secondRemainder(pv.phi5, pvPrev.phi5, pvPrevPrev.phi5),
227             phi6 = secondRemainder(pv.phi6, pvPrev.phi6, pvPrevPrev.phi6),
228         ).convertFromPVToArray()
229     )
230
231     return if (delta > Data.DELTA_1) {
232         TimeDemon(currT = currT, deltaT = deltaT / 2, success = false)
233     } else {
```

```
234             if (delta > Data.DELTA_2) {
235                 TimeDemon(currT = currT + deltaT, deltaT = deltaT, success = true)
236             } else {
237                 val newDeltaT = if (deltaT > Data.MAXIMAL_STEP) Data.MAXIMAL_STEP
                        else deltaT * 2
238                 TimeDemon(currT = currT + deltaT, deltaT = newDeltaT, success = true)
239             }
240         }
241 }
242
243
244 fun deltaTReduction(timeDemon: TimeDemon) =
245     timeDemon.copy(deltaT = timeDemon.deltaT / 2.0)
246
247 fun printToFile(name: String, values: DoubleArray) {
248     val file = File(name)
249     for (i in values.indices) {
250         file.appendText("${values[i]}\n")
251     }
252 }
253
254 fun clearFiles() {
255     val files = listOf(
256         File(FileData.PHI1_FILE), File(FileData.PHI2_FILE),
257         File(FileData.PHI4_FILE), File(FileData.PHI5_FILE), File(FileData.T_FILE)
258     )
259     files.forEach { file -> if (file.exists()) file.delete() }
260 }
```

Листинг 4. Exceptions.kt

```
1 interface MimaprException {
2     val error: String
3 }
4
5 data class MatrixDimensionException(
6     override val error: String = "Invalid size of matrix"
7 ) : RuntimeException(), MimaprException
8
9 data class TimeStepException(
10     override val error: String = "Step is now less than the minimum possible"
11 ) : RuntimeException(), MimaprException
12
13 data class ConversionException(
14     override val error: String = "Gauss results don't match the required size"
15 ): RuntimeException(), MimaprException
```