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 «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 2

Тема Программно-алгоритмическая реализация метода Рунге-Кутта 4-го порядка точности при решении системы ОДУ в задаче Коши

Студент Якуба Д. В.

Группа ИУ7-63Б

Оценка (баллы)

Преподаватель Градов В. М.

Москва.
2021 г.

Лабораторная работа по теме «Программно-алгоритмическая реализация метода Рунге-Кутта 4-го порядка точности при решении системы ОДУ в задаче Коши»

Тема:

Программно-алгоритмическая реализация метода Рунге-Кутта 4-го порядка точности при решении системы ОДУ в задаче Коши.

Цель работы:

Получение навыков разработки алгоритмов решения задачи Коши при реализации моделей, построенных на системе ОДУ, с использованием метода Рунге-Кутта 4-го порядка точности.

Задание:

Задана система электротехнических уравнений, описывающих разрядный контур, включающий постоянное активное сопротивление, нелинейное сопротивление, зависящее от тока, индуктивность и емкость.

$$\begin{cases} \dfrac{dI}{dT} = \dfrac{U - \left(R_k + R_p(I)\right) I}{L_k} \\ \dfrac{dU}{dt} = -\dfrac{I}{C_k} \end{cases}$$

Начальные условия:

$$t = 0, I = I_0, U = U_0$$

Здесь I, U – ток и напряжение на конденсаторе.

Сопротивление R_p рассчитать по формуле:

$$R_p = \frac{l_p}{2\pi R^2 \int_0^1 \sigma\big(T(z)\big) z dz}$$

Для функции T(z) применить выражение $T(z) = T_0 + (T_w - T_0) z^m$.

Параметры T_0, m находятся интерполяцией из таблицы 1 при известном токе I.

Коэффициент электропроводности $\sigma(T)$ зависит от T и рассчитывается интерполяцией из таблицы 2.

Таблица 1

I, A	To, K	m
0.5	6730	0.50
1	6790	0.55
5	7150	1.7
10	7270	3
50	8010	11
200	9185	32
400	10010	40
800	11140	41
1200	12010	39

Таблица 2

T, K	σ , 1/Ом см
4000	0.031
5000	0.27
6000	2.05
7000	6.06
8000	12.0
9000	19.9
10000	29.6
11000	41.1
12000	54.1
13000	67.7
14000	81.5

Таблица 2

Параметры разрядного контура:

R=0.35 см
$l_э$=12 см
L_k=187 10^{-6} Гн
C_k=268 10^{-6} Ф
R_k=0.25 Ом
U_{co}=1400 В
I_o=0..3 А
T_w=2000 К

Для *справки*: при указанных параметрах длительность импульса около 600 мкс, максимальный ток – около 800 А

Описание

Указанная в условии система уравнений решается методом Рунге-Кутта 4-го порядка точности:

$$\begin{cases} y_{n+1} = y_n + \dfrac{k_1 + 2k_2 + 2k_3 + k_4}{6} \\ z_{n+1} = z_n + \dfrac{p_1 + 2p_2 + 2p_3 + p_4}{6} \end{cases}$$

где

$$k_1 = h_n f(x_n, y_n, z_n)$$

$$p_1 = h_n \varphi(x_n, y_n, z_n)$$

$$k_2 = h_n \mathrm{f}\left(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2}, z_n + \frac{p_1}{2}\right)$$

$$p_2 = h_n\varphi\left(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2}, z_n + \frac{p_1}{2}\right)$$

$$k_3 = h_n f\left(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2}, z_n + \frac{p_2}{2}\right)$$

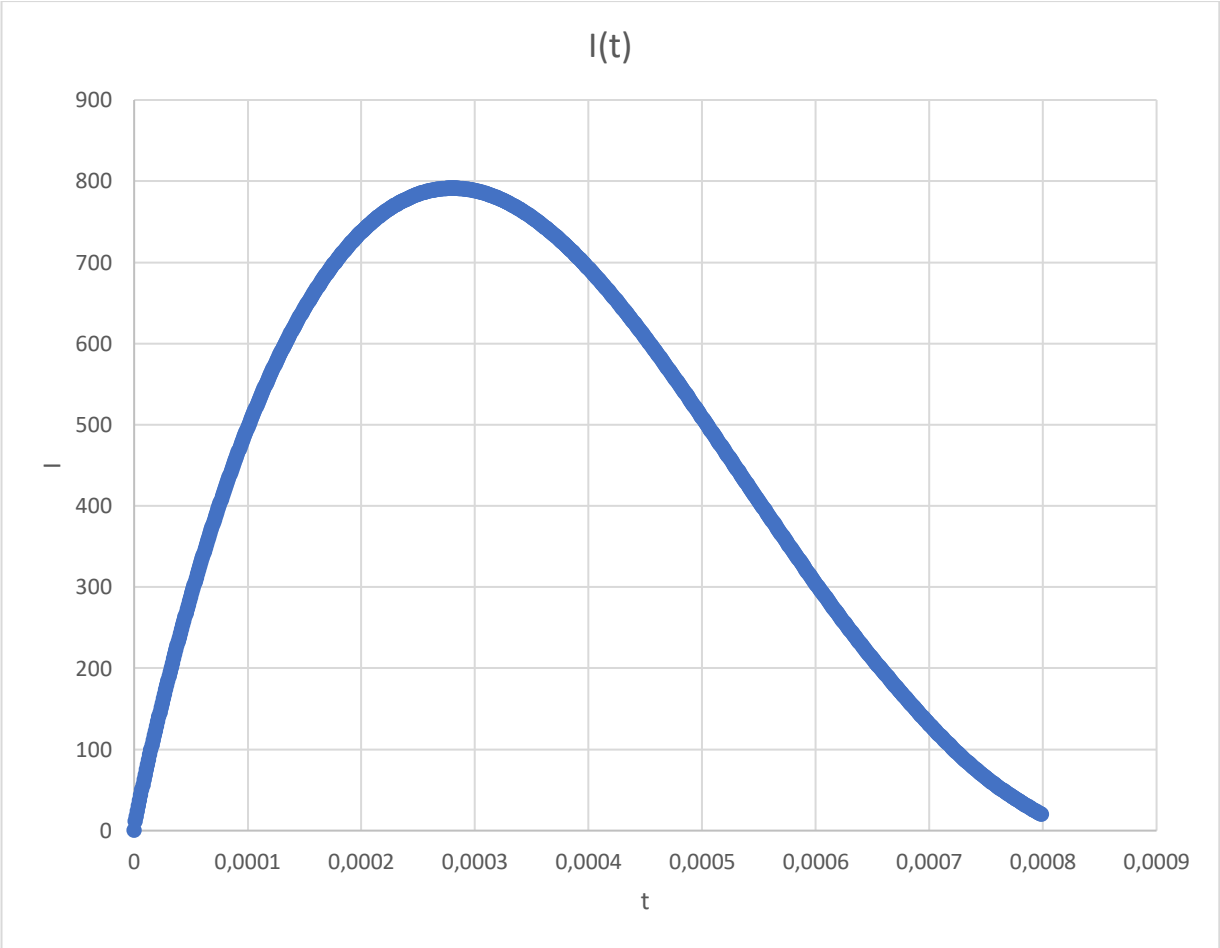
$$p_3 = h_n\varphi\left(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2}, z_n + \frac{p_2}{2}\right)$$

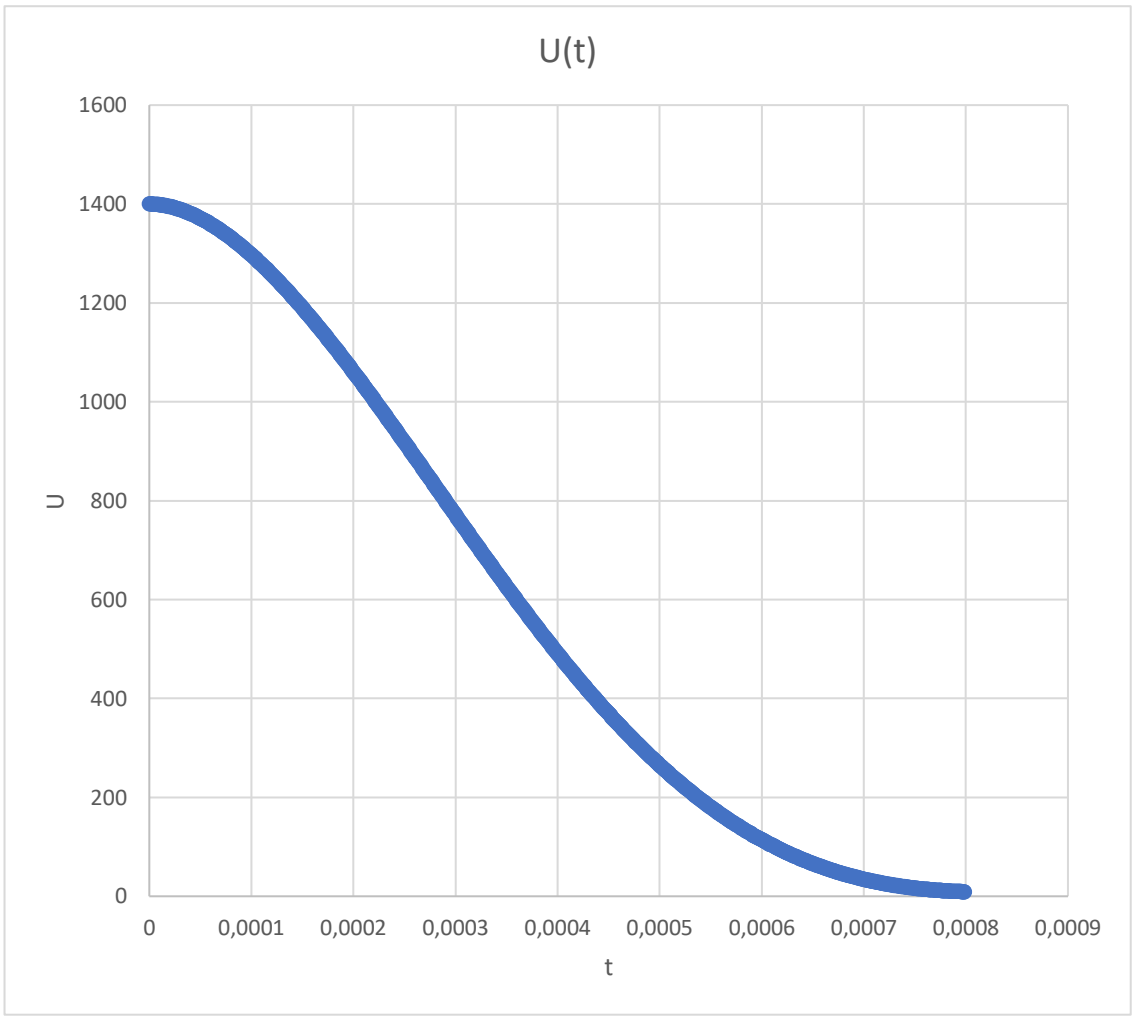
$$k_4 = h_n f(x_n + h_n, y_n + k_3, z_n + p_3)$$

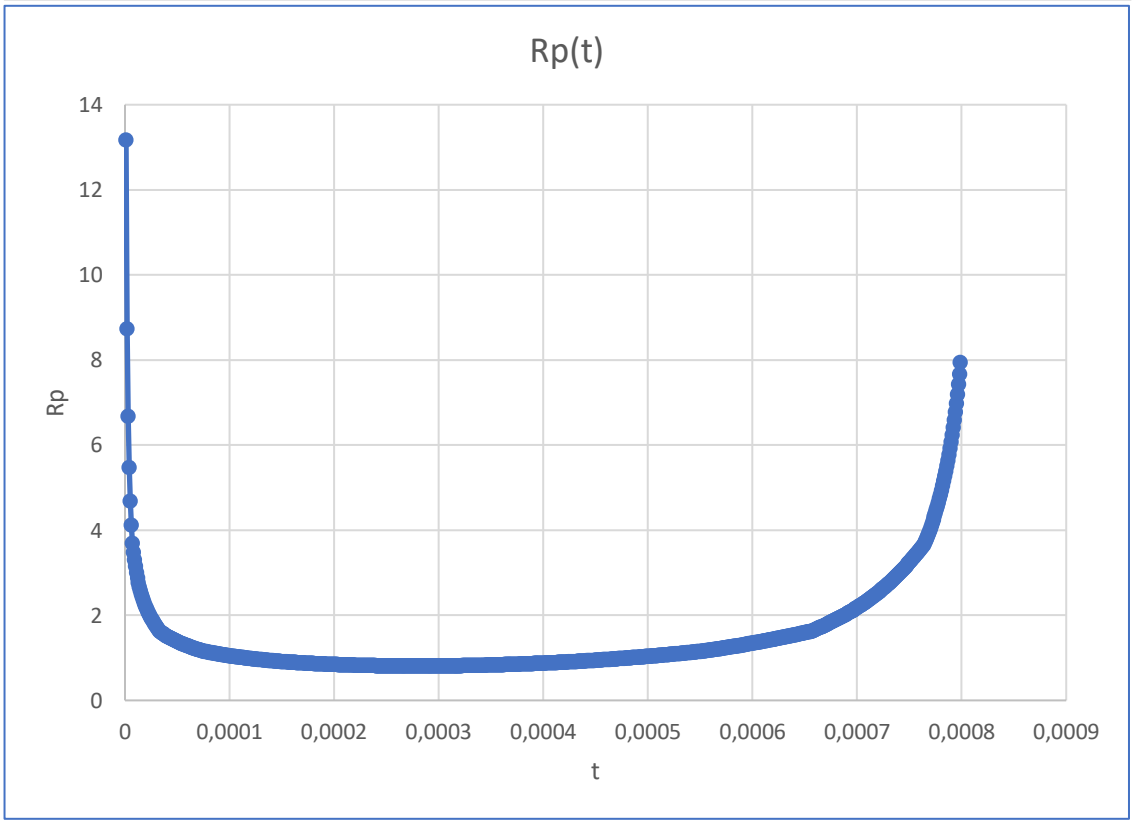
$$p_4 = h_n\varphi(x_n + h_n, y_n + k_3, z_n + p_3)$$

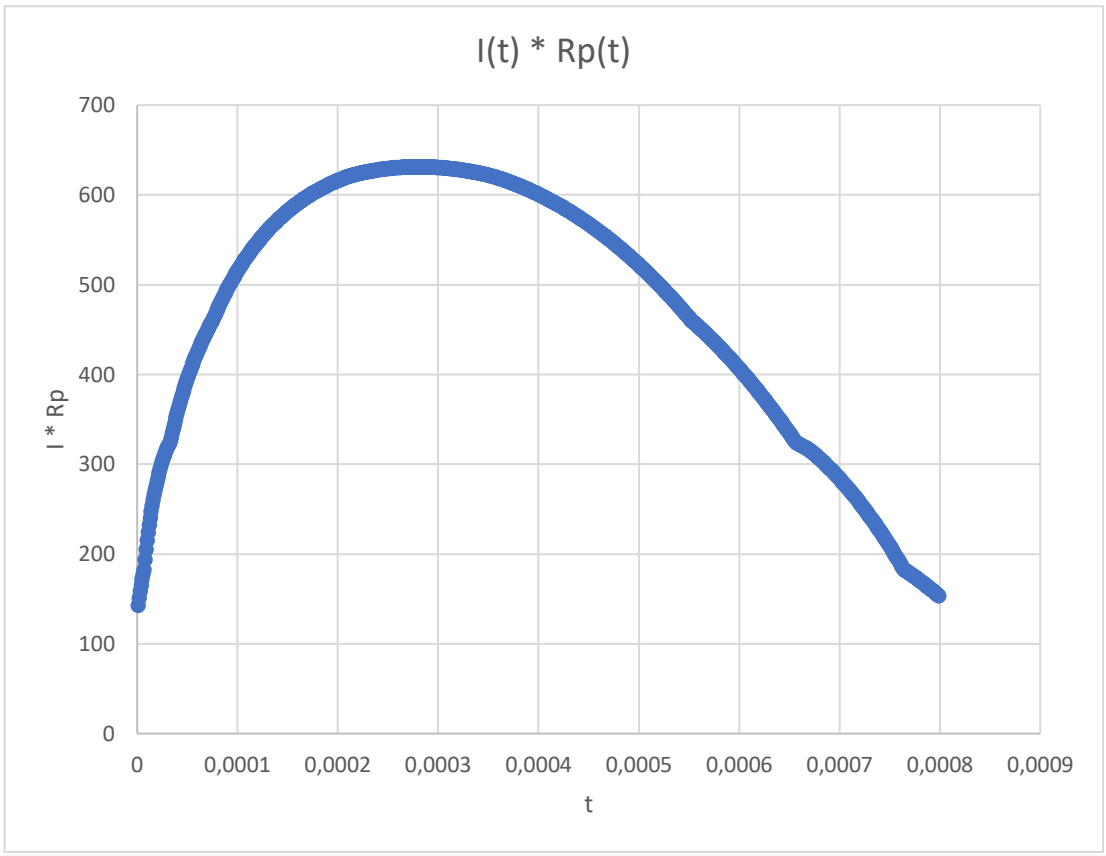
Результат

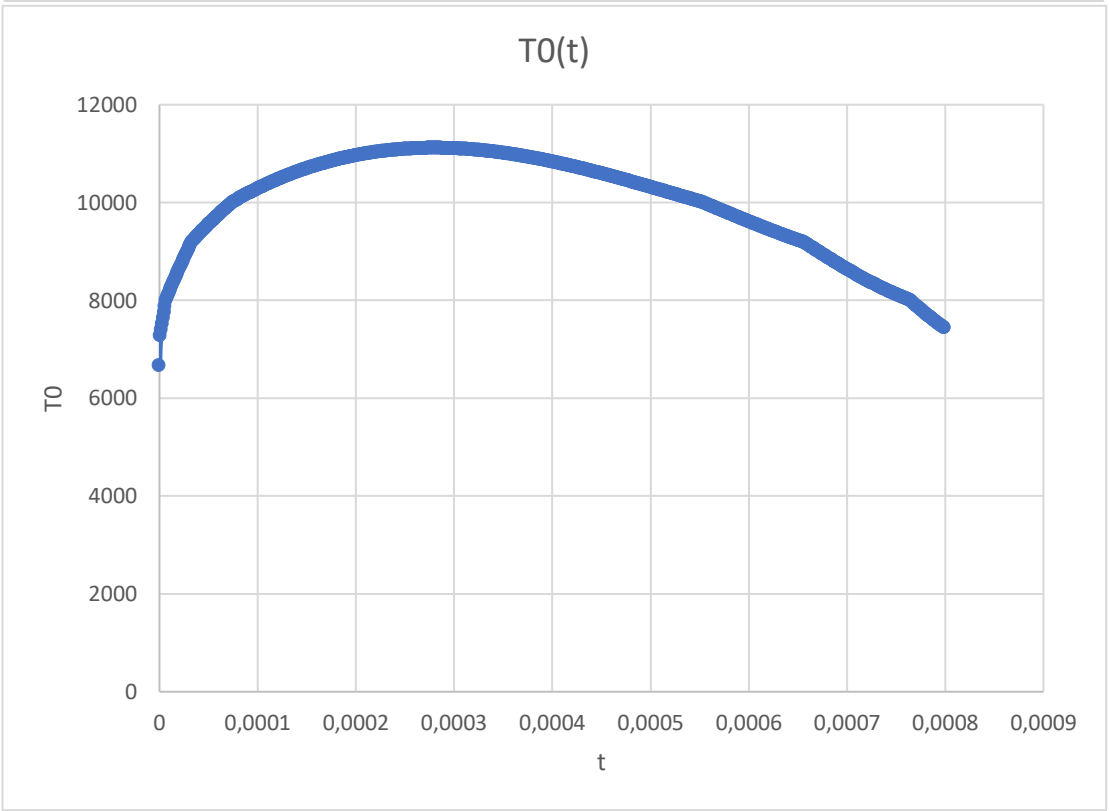
1. Графики зависимости от времени импульса $I(t), U(t), R_p(t), I(t) \cdot R_p(t), T_0(t)$ при заданных выше параметрах.



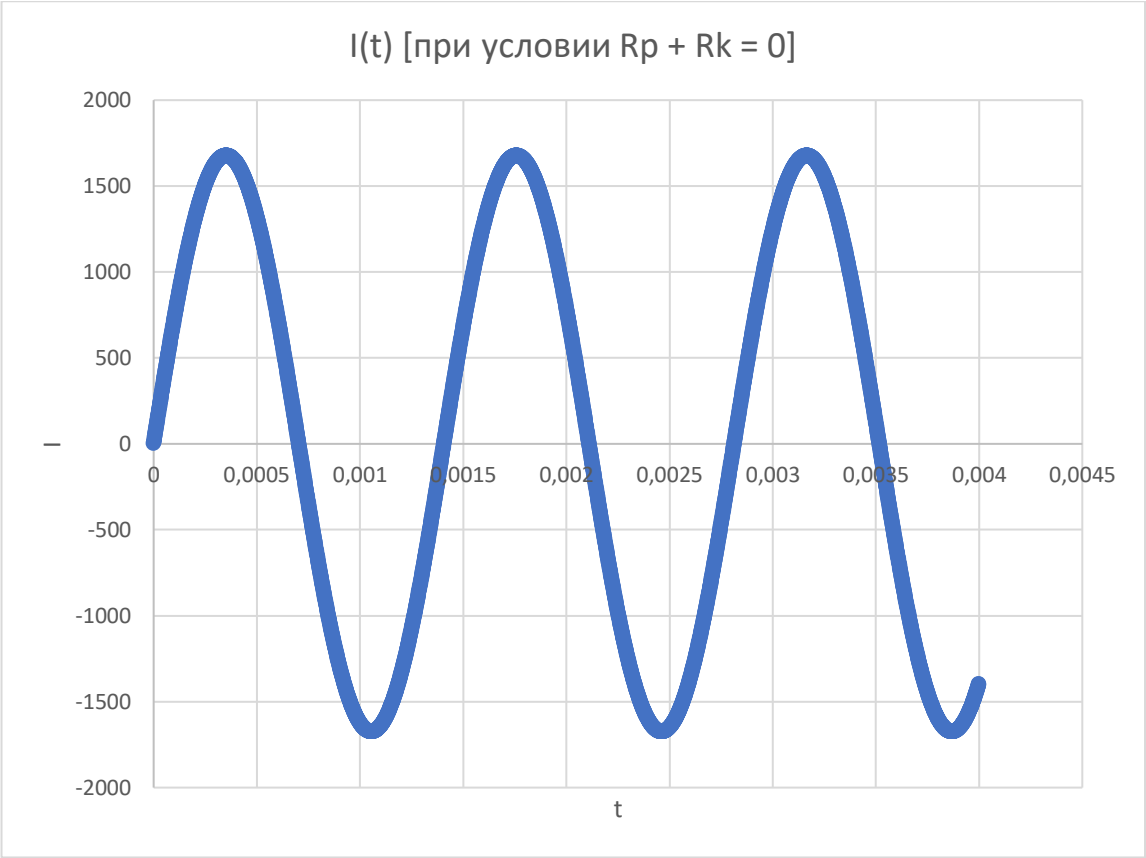

U(t)
1600
1400
1200
1000
800
600
400
200
0
U
0
0,0001
0,0002
0,0003
0,0004
0,0005
0,0006
0,0007
0,0008
0,0009
t


Rp(t)
14
12
10
8
6
4
2
0
Rp
0
0,0001
0,0002
0,0003
0,0004
0,0005
0,0006
0,0007
0,0008
0,0009
t

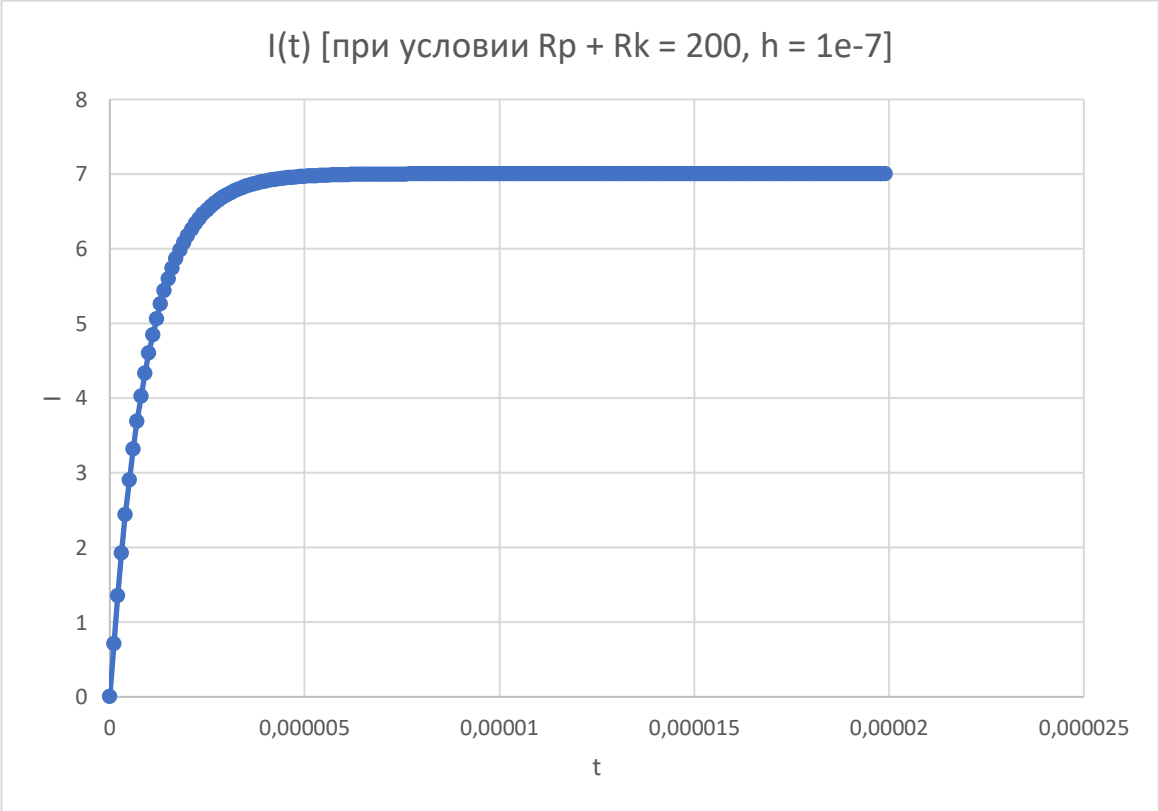

I(t) * Rp(t)
700
600
500
400
300
200
100
0
I * Rp
0
0,0001
0,0002
0,0003
0,0004
0,0005
0,0006
0,0007
0,0008
0,0009
t


T0(t)
12000
10000
8000
6000
4000
2000
0
T0
0
0,0001
0,0002
0,0003
0,0004
0,0005
0,0006
0,0007
0,0008
0,0009
t

2. График зависимости $I(t)$ при $R_k + R_p = 0$. Колебания незатухающие.



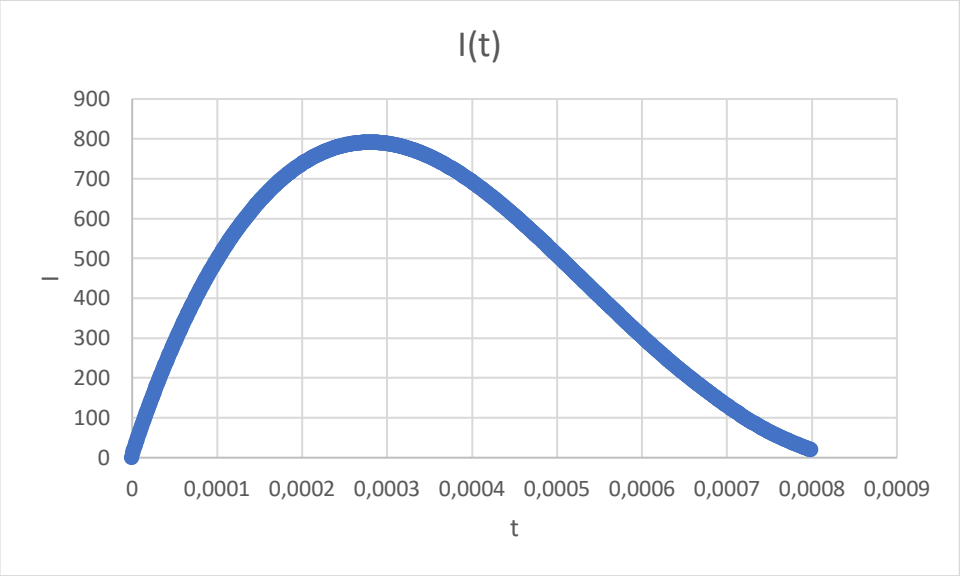
3. График зависимости $I(t)$ при $R_k + R_p = const = 200$ Ом в интервале значений t 0-20 мкс.



4. Результаты исследования влияния параметров контура C_k, L_k, R_k на длительность импульса $t_{имп}$ апериодической формы.

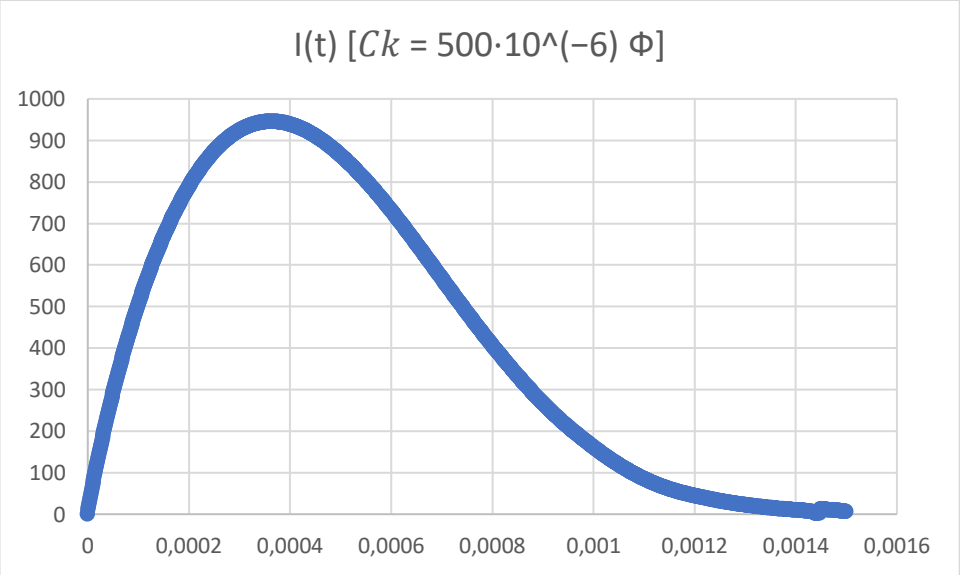
Изучение параметра C_k:

При $C_k = 268 \cdot 10^{-6}$Ф



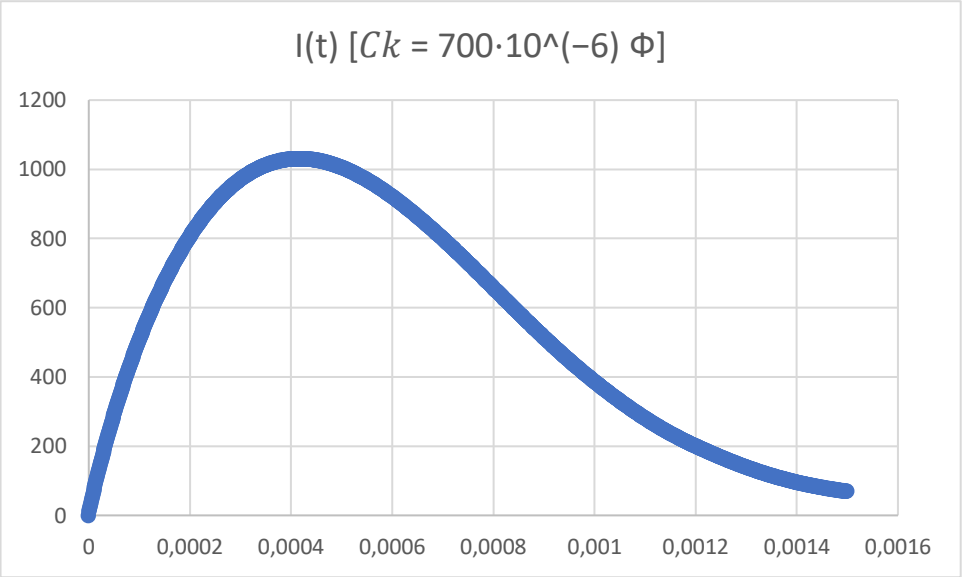
$0.35I_{max} = 276{,}8006171$; Соответствует начальному значению t = 0,000048 и конечному значению t = 0,000614. $t_{имп} = 0{,}000566$.

При $C_k = 500 \cdot 10^{-6}$Ф



$0.35I_{max} = 330{,}765$; Соответствует начальному значению t = 0,000059 и конечному значению t = 0,000853. $t_{имп} = 0{,}000794$.

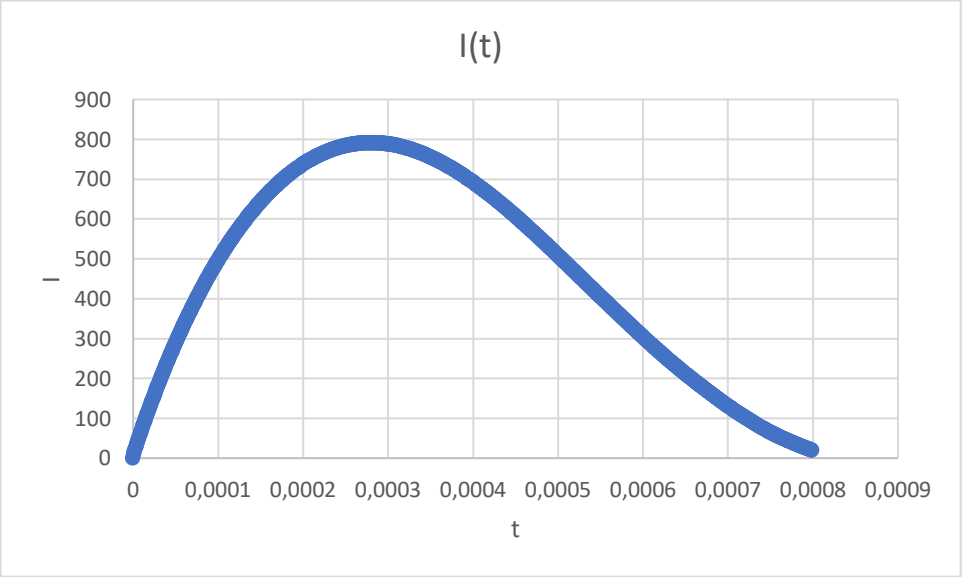
При $C_k = 700 \cdot 10^{-6}$Ф



$0.35I_{max} = 360{,}8429$; Соответствует начальному значению t = 0,000065 и конечному значению t = 0,001024. $t_{имп} = 0{,}000959$.

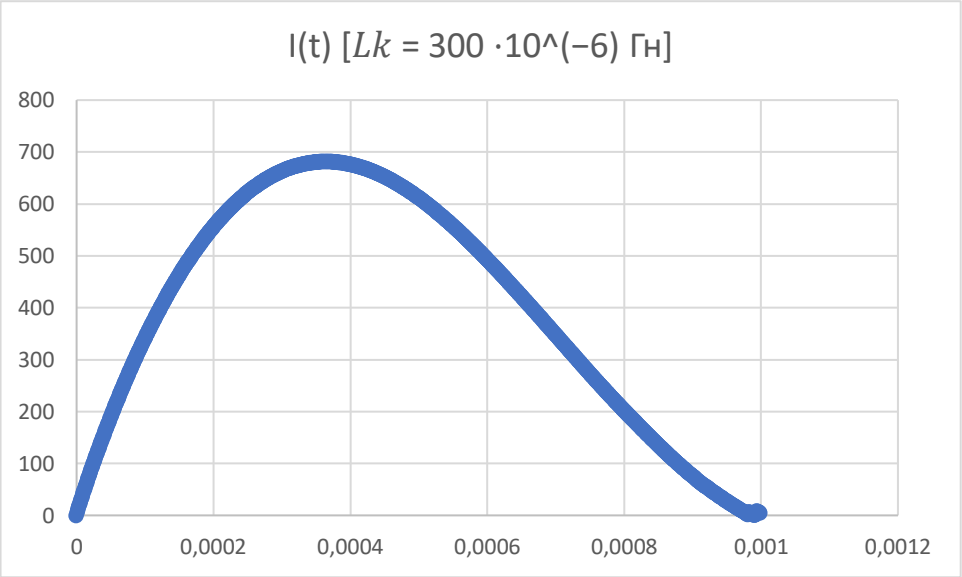
Вывод: при возрастании C_k возрастает и $t_{имп}$.

При $L_k = 187 \cdot 10^{-6}$Гн



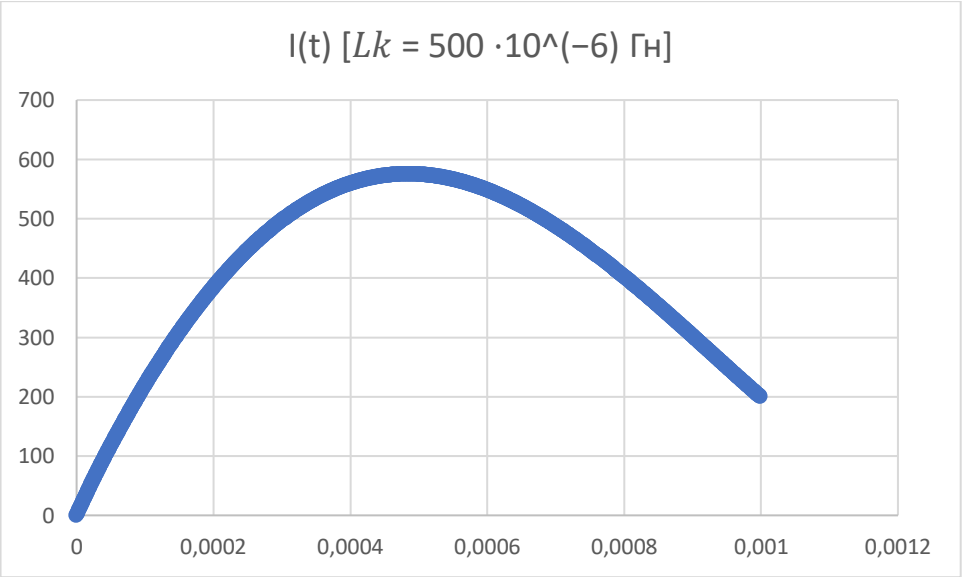
$0.35I_{max} = 276{,}8006171$; Соответствует начальному значению t = 0,000048 и конечному значению t = 0,000614. $t_{имп} = 0{,}000566$.

При $L_k = 300 \cdot 10^{-6}$Гн



$0.35I_{max} = 238{,}5789$; Соответствует начальному значению t = 0,000065 и конечному значению t = 0,000775. $t_{имп} = 0{,}000710$.

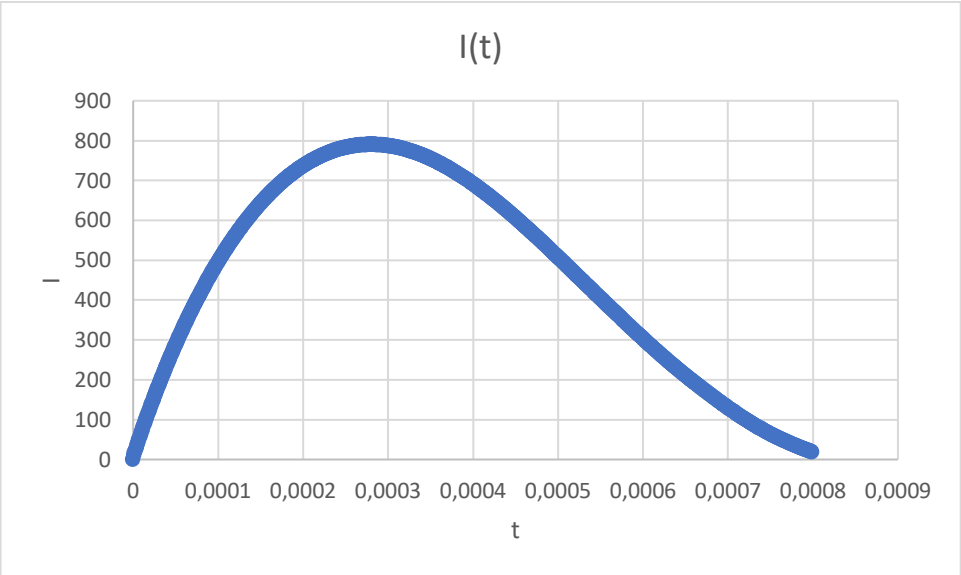
При $L_k = 500 \cdot 10^{-6}$Гн



$0.35I_{max} = 201{,}4575$; Соответствует начальному значению t = 0,000091 и конечному значению t = 0,000998. $t_{имп} = 0{,}000907$.

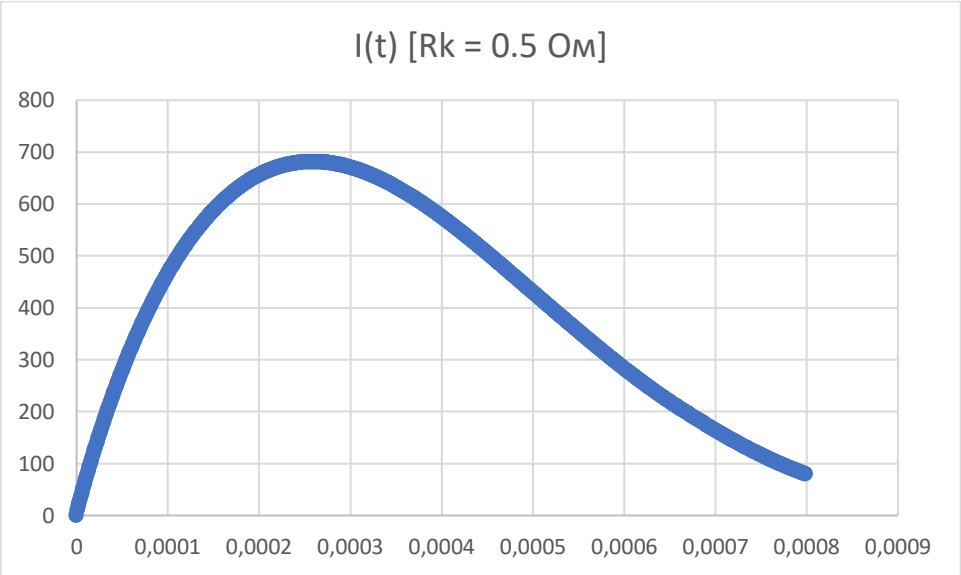
Вывод: при возрастании L_k возрастает и $t_{имп}$.

При $R_k = 0.25$ Ом



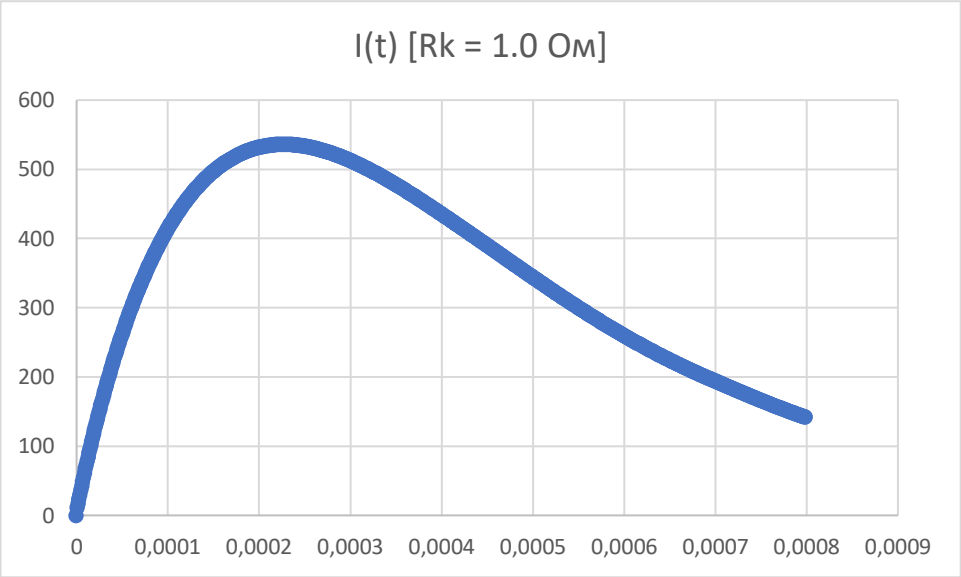
$0.35I_{max} = 276{,}8006171$; Соответствует начальному значению t = 0,000048 и конечному значению t = 0,000614. $t_{имп} = 0{,}000566$.

При $R_k = 0.5$ Ом



$0.35I_{max} = 238{,}5942$; Соответствует начальному значению t = 0,000041 и конечному значению t = 0,000634. $t_{имп} = 0{,}000593$.

При $R_k = 1$ Ом



$0.35I_{max} = 187{,}6424$; Соответствует начальному значению t = 0,000033 и конечному значению t = 0,000712. $t_{имп} = 0{,}000679$.

Вывод: при возрастании R_k возрастает и $t_{имп}$.

Таким образом, все рассматриваемые параметры при увеличении позволяют «растянуть» кривую, тем самым увеличивая так называемую длительность импульса.

Контрольные вопросы

1. Какие способы тестирования программы, кроме указанного в п.2, можете предложить ещё?

Ответ: от реализованного программного обеспечения требуется, чтобы оно выводило результаты, соответствующие законам физики, поэтому наилучшими способами тестирования будет сравнение реального поведения представленного контура и моделируемого. Наравне с приравниваем значения R_p к нулю, можно вовсе убрать какое бы то ни было сопротивление из цепи, либо задать значение R_k некоторым большим числом.

2. Получите систему разностных уравнений для решения сформулированной задачи неявным методом трапеций. Опишите алгоритм реализации полученных уравнений.

Ответ:

Неявный метод трапеций:

$$u(t+\tau) = u(t) + \int_0^t u'(t+\tau)d\tau$$

Если в данной формуле рассматриваемый интеграл заменить формулой трапеций, то получим:

$$u(t+\tau) = u(t) + \frac{\tau}{2}[u'(t) + u'(t+\tau)] + O(\tau^2)$$

Откуда имеем:

$$u_{n+1} = u_n + \frac{\tau}{2}[f(t_n, u_n) + f(t_{n+1}, u_{n+1})]$$

Таким образом:

$$\begin{cases} \dfrac{dI}{dT} = \dfrac{U - \left(R_k + R_p(I)\right) I}{L_k} \\ \dfrac{dU}{dt} = -\dfrac{I}{C_k} \end{cases}$$

$$I_{n+1} = I_n + \frac{h}{2}\left[\frac{U_n - \left(R_k + R_p(I_n)\right) I_n}{L_k} + \frac{U_{n+1} - \left(R_k + R_p(I_{n+1})\right) I_{n+1}}{L_k}\right]$$

$$U_{n+1} = U_n + \frac{h}{2}\left[-\frac{I_n}{C_k} - \frac{I_{n+1}}{C_k}\right]$$

Путём подстановки U_{n+1} в выражение для I_{n+1} получаем:

$$I_{n+1} = I_n + \frac{h}{2L_k}\left[2U_n - \left(R_k + R_p(I_n)\right) I_n - \left(R_k + R_p(I_{n+1})\right) I_{n+1} - \frac{h}{2}\left[\frac{I_n + I_{n+1}}{C_k}\right]\right]$$

Таким образом, имея I_0 и U_0 можем вычислить значения I_n, U_n при некоторых заданных параметрах.

3.Из каких соображений проводится выбор численного метода того или иного порядка точности, учитывая, что чем выше порядок точности метода, тем он более сложен и требует, как правило, больших ресурсов вычислительной системы?

Ответ: оценка погрешности для частного случая вида правой части дифференциального уравнения:

$$\varphi(x, v) \equiv \varphi(x)$$

Выбор той или иной из приведенных схем для решения конкретной задачи определяется следующими соображениями. Если функция $\varphi(x, v)$ правой части уравнения непрерывна и ограничена, а также непрерывны и ограничены её четвертые производные, то наилучший результат достигается при использовании схемы:

$$y_{n+1} = y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

$$k_1 = h_n \varphi(x_n, y_n)$$

$$k_2 = h_n \varphi\left(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = h_n \varphi\left(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = h_n \varphi(x_n + h_n, y_n + k_3)$$

В том случае, когда функция не имеет названых выше производных, предельный (четвертый) порядок указанной выше схемы не может быть достигнут, и целесообразным оказывается применение более простых схем.

4. Можно ли метод Рунге-Кутта применить для решения задачи, в которой часть условий задана на одной границе, а часть – на другой? Например, напряжение по-прежнему задано при $t = 0$, то есть $t = 0, U = U_0$, а ток задан в другой момент времени, к примеру, в конце импульса, то есть при $t = T, I = I_T$. Какой можете предложить алгоритм вычислений?

Ответ: на мой взгляд, очень неэффективным, но рабочим алгоритмом может оказаться вариант с последовательным заданием некоторого значения при $t = 0: I = I_0$ и проверка вычисленного при $t = T$ значения тока с заданным в условии. Истинное значение I_0 будет определено в том случае, если $I[\text{при } t = T] = I_T$ будет равняться $I[\text{при } t = T] = I_{T\text{ вычисленное}}$ с некоторой заданной точностью. Подобный метод перебора может потребовать большого количества ресурсов, поэтому более верным решением, если это некритично, будет использование методов Рунге-Кутта низких порядков.

Код программы

```
import kotlin.math.PI
import kotlin.math.absoluteValue
import kotlin.math.pow
import kotlin.math.round

var curT0 = 0.0

// Linear interpolation. Works only for (sort #'> table)
```

```kotlin
fun linearInterpolation(table: List<Pair<Double, Double>>, findX: Double): Double
{
    if (findX <= table[0].first) return table[0].second +
            (table[1].second - table[0].second) / (table[1].first - table[0].first) *
(findX - table[0].first)

    var firstDot: Pair<Double, Double>? = null
    var secondDot: Pair<Double, Double>? = null

    var curPairInd: Int = 0
    while ((curPairInd < table.size) && (firstDot == null))
    {
        if (table[curPairInd].first >= findX)
        {
            firstDot = table[curPairInd - 1]
            secondDot = table[curPairInd]
        }
        curPairInd++
    }

    if (firstDot == null)
    {
        firstDot = table[curPairInd - 2]
        secondDot = table[curPairInd - 1]
    }

    return firstDot.second +
            (secondDot!!.second - firstDot.second) / (secondDot.first -
firstDot.first) * (findX - firstDot.first)
}

// trapezodial integration. Be careful with arguments: leftLimit > rightLimit and
fragNum > 0
fun trapezodialIntegrationWithTable(
    leftLimit: Double,
    rightLimit: Double,
    fragNum: Int,
    table: List<Pair<Double, Double>>) : Double
{
    val step: Double = (rightLimit - leftLimit) / fragNum
    var curX: Double = leftLimit
    var outSum: Double = 0.0
    var fInter: Double?
    var sInter: Double?
    for (ind in 0 until fragNum)
    {
        fInter = linearInterpolation(table, curX)
        sInter = linearInterpolation(table, curX + step)
        outSum += (fInter + sInter) / 2.0

        curX += step
    }

    return round((outSum * step) * 1e5) / 1e5
}

// trapezodial integration. Be careful with arguments: leftLimit > rightLimit and
fragNum > 0
fun trapezodialIntegrationWithFunction(
    leftLimit: Double,
    rightLimit: Double,
    fragNum: Int,
```

```
    func: (Double) -> Double) : Double
{
    val step: Double = (rightLimit - leftLimit) / fragNum.toDouble()
    var outSum: Double = 0.0

    var curZ: Double = 0.0
    for (ind in 0 until fragNum)
    {
        outSum += (func(curZ) + func(curZ + step)) / 2.0 * step
        curZ += step
    }

    return outSum
}

fun T(z: Double, curT: Double, Tw: Double, curM: Double): Double
{
    return curT + (Tw - curT) * z.pow(curM)
}

fun sigma(T: Double, Tsigma_Table: List<Pair<Double, Double>>): Double
{
    return linearInterpolation(Tsigma_Table, T)
}

fun findNonLinearResistance(IT0_Table: List<Pair<Double, Double>>,
                            Im_Table: List<Pair<Double, Double>>,
                            Tsigma_Table: List<Pair<Double, Double>>,
                            Tw: Double, amperage: Double,
                            Ie: Double, Res: Double): Double
{
    val currentT0: Double = linearInterpolation(IT0_Table, amperage)
    curT0 = currentT0
    val currentM: Double = linearInterpolation(Im_Table, amperage)
    val getSigma = fun(z: Double): Double { return sigma(T(z, currentT0, Tw,
currentM), Tsigma_Table) * z }

    val integral = trapezodialIntegrationWithFunction(
        0.0, 1.0, 40, getSigma)

    return Ie / (2 * PI * Res * Res * integral)
}

fun fFunction(curA: Double, curU: Double, parameters: Map<String, Double>,
              Rp: Double): Double
{
    return (curU - (parameters["Rk"]!! + Rp /* 0 */ /* 200 */) * curA) /
parameters["Lk"]!!
}

fun phiFunction(curA: Double, Ck: Double): Double
{
    return - curA / Ck
}

fun getNextAmperageVoltage(curA: Double,
                           curU: Double,
                           parameters: Map<String, Double>,
                           Rp: Double,
                           step: Double): Pair<Double, Double>
{
    val Ck = parameters["Ck"]!!
```

```
    val f1 = step * fFunction(curA, curU, parameters, Rp)
    val phi1 = step * phiFunction(curA, Ck)
    val f2 = step * fFunction(curA + f1 / 2, curU + phi1 / 2, parameters, Rp)
    val phi2 = step * phiFunction(curA + f1 / 2, Ck)
    val f3 = step * fFunction(curA + f2 / 2, curU + phi2 / 2, parameters, Rp)
    val phi3 = step * phiFunction(curA + f2 / 2, Ck)
    val f4 = step * fFunction(curA + f3, curU + phi3, parameters, Rp)
    val phi4 = step * phiFunction(curA + f3, Ck)

    return Pair(curA + (f1 + 2 * f2 + 2 * f3 + f4) / 6, curU + (phi1 + 2 * phi2 + 2 *
phi3 + phi4) / 6)
}

fun main()
{
    val IT0_Table: List<Pair<Double, Double>> =
        listOf(
            Pair(0.5, 6730.0),
            Pair(1.0, 6790.0),
            Pair(5.0, 7150.0),
            Pair(10.0, 7270.0),
            Pair(50.0, 8010.0),
            Pair(200.0, 9185.0),
            Pair(400.0, 10010.0),
            Pair(800.0, 11140.0),
            Pair(1200.0, 12010.0)
        )

    val Im_Table: List<Pair<Double, Double>> =
        listOf(
            Pair(0.5, 0.5),
            Pair(1.0, 0.55),
            Pair(5.0, 1.7),
            Pair(10.0, 3.0),
            Pair(50.0, 11.0),
            Pair(200.0, 32.0),
            Pair(400.0, 40.0),
            Pair(800.0, 41.0),
            Pair(1200.0, 39.0))

    val Tsigma_Table: List<Pair<Double, Double>> =
        listOf(
            Pair(4000.0, 0.031),
            Pair(5000.0, 0.27),
            Pair(6000.0, 2.05),
            Pair(7000.0, 6.06),
            Pair(8000.0, 12.0),
            Pair(9000.0, 19.9),
            Pair(10000.0, 29.6),
            Pair(11000.0, 41.1),
            Pair(12000.0, 54.1),
            Pair(13000.0, 67.7),
            Pair(14000.0, 81.5)
        )

    val parameters: Map<String, Double> =
        mapOf("R" to 0.35, "Ie" to 12.0, "Lk" to 187 * 1e-6, "Ck" to 268 * 1e-6, "Rk"
to 0.25,
            "Uco" to 1400.0, "Tw" to 2000.0)

    var curT: Double = 0.0
```

```
    val step: Double = 1e-6
    var currentAmperage: Double = 0.0
    var currentVoltage: Double = 1400.0

    val outTableIT: MutableList<Pair<Double, Double>> = mutableListOf()
    val outTableUT: MutableList<Pair<Double, Double>> = mutableListOf()
    val outTableRpT: MutableList<Pair<Double, Double>> = mutableListOf()
    val outTableT0: MutableList<Pair<Double, Double>> = mutableListOf()
    val outTableIRpT: MutableList<Pair<Double, Double>> = mutableListOf()

    var curRp: Double

    while (curT < 8e-4)
    {
        curRp = findNonLinearResistance(IT0_Table, Im_Table, Tsigma_Table,
parameters["Tw"]!!, currentAmperage, parameters["Ie"]!!, parameters["R"]!!)

        outTableIT.add(Pair(currentAmperage, curT))
        outTableUT.add(Pair(currentVoltage, curT))
        outTableRpT.add(Pair(curRp, curT))
        outTableT0.add(Pair(curT0, curT))
        outTableIRpT.add(Pair(currentAmperage * curRp, curT))

        val curPair = getNextAmperageVoltage(currentAmperage, currentVoltage,
parameters, curRp, step)
        currentAmperage = curPair.first
        currentVoltage = curPair.second
        curT += step
    }

    for (i in outTableIT)
        println("%.6f".format(i.first))
}
```