|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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.

Начальные условия:

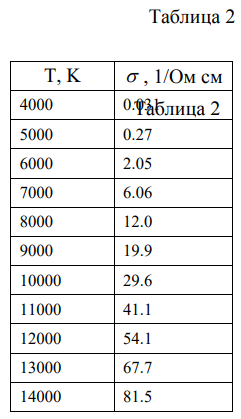
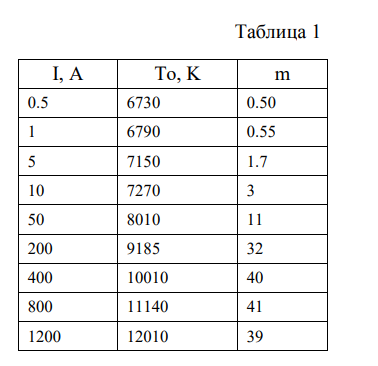
Здесь I, U – ток и напряжение на конденсаторе.

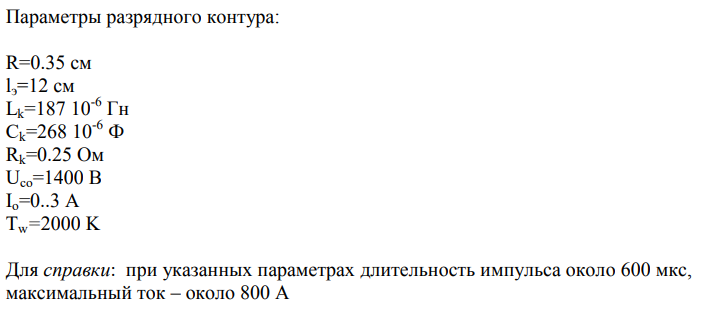
Сопротивление рассчитать по формуле:

Для функции T(z) применить выражение .

Параметры находятся интерполяцией из таблицы 1 при известном токе I.

Коэффициент электропроводности зависит от T и рассчитывается интерполяцией из таблицы 2.





Описание

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

где

Результат

1. Графики зависимости от времени импульса при заданных выше параметрах.

2. График зависимости при . Колебания незатухающие.

3. График зависимости при в интервале значений 0-20 мкс.

4. Результаты исследования влияния параметров контура на длительность импульса апериодической формы.

Изучение параметра :

При

; Соответствует начальному значению t = 0,000048 и конечному значению t = 0,000614. .

При

; Соответствует начальному значению t = 0,000059 и конечному значению t = 0,000853. .

При

; Соответствует начальному значению t = 0,000065 и конечному значению t = 0,001024. .

**Вывод: при возрастании возрастает и .**

При

; Соответствует начальному значению t = 0,000048 и конечному значению t = 0,000614. .

При

; Соответствует начальному значению t = 0,000065 и конечному значению t = 0,000775. .

При

; Соответствует начальному значению t = 0,000091 и конечному значению t = 0,000998. .

**Вывод: при возрастании возрастает и .**

При

; Соответствует начальному значению t = 0,000048 и конечному значению t = 0,000614. .

При

; Соответствует начальному значению t = 0,000041 и конечному значению t = 0,000634. .

При

; Соответствует начальному значению t = 0,000033 и конечному значению t = 0,000712. .

**Вывод: при возрастании возрастает и .**

**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 к нулю, можно вовсе убрать какое бы то ни было сопротивление из цепи, либо задать значение некоторым большим числом.

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

Ответ:

Неявный метод трапеций:

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

Откуда имеем:

Таким образом:

Путём подстановки в выражение для получаем:

Таким образом, имея и можем вычислить значения 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:

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

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 то есть а ток задан в другой момент времени, к примеру, в конце импульса, то есть при . Какой можете предложить алгоритм вычислений?

Ответ: на мой взгляд, очень неэффективным, но рабочим алгоритмом может оказаться вариант с последовательным заданием некоторого значения при и проверка вычисленного при значения тока с заданным в условии. Истинное значение будет определено в том случае, если будет равняться 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)  
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))  
}