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: Решение системы дифференциальных уравнений с помощью метода Рунге-Кутта

Студент: Лаврова А. А.

Группа: ИУ7-65Б

Оценка (баллы) _____

Преподаватель: Градов В.М.

Москва.
2020 г.

Описание задачи:

Дан колебательный контур с газоразрядной трубкой.

Данный контур можно описать с помощью системы уравнений:

$$\begin{cases} L_k \frac{dI}{dt} + (R_k + R_p(I)) \cdot I - U_C = 0 \\ \frac{dU_c}{dt} = -\frac{I}{C_k} \end{cases}$$

Данный контур можно описать с помощью системы уравнений:

$$\begin{cases} L_k \frac{dI}{dt} + (R_k + R_p(I)) \cdot I - U_C = 0 \\ \frac{dU_c}{dt} = -\frac{I}{C_k} \end{cases}$$

Ее необходимо решить и построить графики:
- I(t) - сила тока в цепи
- Uc(t) - напряжение на конденсаторе
- Ucp(t) - напряжение на газоразрядной трубке
- Rp(t) - сопротивление лампы

Значение Rp(I) можно вычислить по формуле:

$$Rp = \frac{l_e}{2\pi \cdot \int_0^R \sigma(T(r)) r dr} = |z = r/R| = \frac{l_e}{2\pi R^2 \cdot \int_0^1 \sigma(T(z)) dz}$$

Значение T(z) вычисляется по формуле:

$$T(z) = (T_w - T_0) \cdot Z^m + T_0$$

Параметры могут меняться из интерфейса:
- Rk = 0.2 Ом (Сопротивление)
- Lk = 60e-6 Гн (Индуктивность)
- Ck = 150e-6 Ф (Емкость конденсатора)
- R = 0.35 см (Радиус трубки)
- T0 = 300 К
- Tw = 2000К
- le = 12 см (Расстояние между электродами лампы)
- Uc0 = 3000 В (Напряжение на конденсаторе в начальный момент времени t = 0)

I0 = 0 .. 2 А (Сила тока в цепи в начальный момент времени t = 0)

Даны таблицы:

I	T_0	m
0.5	6400	0.40
1	6790	0.50
5	7150	1.70
10	7270	3.0
50	8010	11.0
200	9185	32.0
400	10010	40.0
800	11140	41.0
1200	12010	39.0

T	σ
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

Метод Рунге-Кутта 4-ого порядка:

$$I_{n+1} = I_n + \Delta t \cdot \frac{k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4}{6}$$

$$U_{n+1} = U_n + \Delta t \cdot \frac{q_1 + 2 \cdot q_2 + 2 \cdot q_3 + q_4}{6}$$

$$\begin{cases} k_1 = h_n \cdot f(\Delta t, I_n, U_n) \\ q_1 = h_n \cdot \varphi(\Delta t, I_n, U_n) \\ k_2 = h_n \cdot f(\Delta t + \frac{h_n}{2}, I_n + \frac{k_1}{2}, U_n + \frac{q_1}{2}) \\ q_2 = h_n \cdot \varphi(\Delta t + \frac{h_n}{2}, I_n + \frac{k_1}{2}, U_n + \frac{q_1}{2}) \\ k_3 = h_n \cdot f(\Delta t + \frac{h_n}{2}, I_n + \frac{k_2}{2}, U_n + \frac{q_2}{2}) \\ q_3 = h_n \cdot \varphi(\Delta t + \frac{h_n}{2}, I_n + \frac{k_2}{2}, U_n + \frac{q_2}{2}) \\ k_4 = h_n \cdot f(\Delta t + h_n, I_n + k_3, U_n + q_3) \\ q_4 = h_n \cdot \varphi(\Delta t + h_n, I_n + k_3, U_n + q_3) \end{cases}$$

Метод Рунге-Кутта 2-ого порядка:

$$\begin{cases} I_{n+1} = I_n + h_n \cdot [(1-\alpha) \cdot f(\Delta t, I_n, U_n) + \alpha \cdot f(x_n + \frac{h_n}{2\alpha}, I_n + \frac{h_n}{2\alpha} \cdot f(\Delta t, I_n, U_n), U_n + \frac{h_n}{2\alpha} \cdot \varphi(\Delta t, I_n, U_n))] \\ U_{n+1} = U_n + h_n \cdot [(1-\alpha) \cdot \varphi(\Delta t, I_n, U_n) + \alpha \cdot \varphi(x_n + \frac{h_n}{2\alpha}, I_n + \frac{h_n}{2\alpha} \cdot f(\Delta t, I_n, U_n), U_n + \frac{h_n}{2\alpha} \cdot \varphi(\Delta t, I_n, U_n))] \end{cases}$$

Листинг программы:

```
from matplotlib import pyplot as plt
from numpy import arange
from math import pi
from scipy import integrate
from scipy.interpolate import InterpolatedUnivariateSpline


table1 = [[0.5, 6400, 0.4],
          [1.0, 6790, 0.55],
          [5.0, 7150, 1.7],
          [10.0, 7270, 3],
          [50.0, 8010, 11],
          [200.0, 9185, 32],
```

```
        [400.0, 10010, 40],
        [800.0, 11140, 41],
        [1200.0, 12010, 39]]


table2 = [[4000,  0.031],
        [5000, 0.27],
        [6000, 2.05],
        [7000, 6.06],
        [8000, 12],
        [9000, 19.9],
        [10000, 29.6],
        [11000, 41.1],
        [12000, 54.1],
        [13000, 67.7],
        [14000, 81.5]]


def f(x, y, z, Rp):
    return -((Rk + Rp) * y - z) / Lk


def phi(x, y, z):
    return -y / Ck


def interpolate(x, x_mas, y_mas):
    order = 1
    s = InterpolatedUnivariateSpline(x_mas, y_mas, k=order)
    return float(s(x))


def T(z):
    return T0 + (Tw - T0) * z ** m


def sigma(T):
    T_from_table = []
    for i in range (len(table2)):
        T_from_table.append(table2[i][0])

    sigm_from_table = []
    for j in range(len(table2)):
        sigm_from_table.append(table2[j][1])

    return interpolate(T, T_from_table, sigm_from_table)


def Rp(l):
    global m
    global T0

    l_from_table = []
    for i in range(len(table1)):
        l_from_table.append(table1[i][0])

    T0_from_table = []
    for j in range(len(table1)):
        T0_from_table.append(table1[j][1])

    m_from_table = []
    for z in range(len(table1)):
        m_from_table.append(table1[z][2])
```

```
    m = interpolate(I, I_from_table, m_from_table)
    T0 = interpolate(I, I_from_table, T0_from_table)

    func = lambda z: sigma(T(z)) * z
    integral = integrate.quad(func, 0, 1)
    Rp = Le / (2 * pi * R ** 2 * integral[0])

    return Rp


def runge_kutta_second_order(xn, yn, zn, hn, Rp):
    alpha = 0.5
    y_next = yn + hn * ((1 - alpha) * f(xn, yn, zn, Rp) + alpha * f(xn + hn / (2 * alpha),
             yn + hn / (2 * alpha) * f(xn, yn, zn, Rp),
             zn + hn / (2 * alpha) * phi(xn, yn, zn), Rp))

    z_next = zn + hn * ((1 - alpha) * phi(xn, yn, zn) + alpha * phi(xn + hn / (2 * alpha),
             yn + hn / (2 * alpha) * f(xn, yn, zn, Rp),
             zn + hn / (2 * alpha) * phi(xn, yn, zn)))

    return y_next, z_next


def runge_kutta_fourth_order(xn, yn, zn, hn, Rp):
    k1 = hn * f(xn, yn, zn, Rp)
    q1 = hn * phi(xn, yn, zn)

    k2 = hn * f(xn + hn / 2, yn + k1 / 2, zn + q1 / 2, Rp)
    q2 = hn * phi(xn + hn / 2, yn + k1 / 2, zn + q1 / 2)

    k3 = hn * f(xn + hn / 2, yn + k2 / 2, zn + q2 / 2, Rp)
    q3 = hn * phi(xn + hn / 2, yn + k2 / 2, zn + q2 / 2)

    k4 = hn * f(xn + hn, yn + k3, zn + q3, Rp)
    q4 = hn * phi(xn + hn, yn + k3, zn + q3)

    y_next = yn + (k1 + 2 * k2 + 2 * k3 + k4) / 6
    z_next = zn + (q1 + 2 * q2 + 2 * q3 + q4) / 6

    return y_next, z_next

def do_plot(pltMasT, mas1, xlabel, ylabel):
    plt.plot(pltMasT, mas1)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.grid(True)
    plt.show()
    plt.savefig(ylabel, format='png')
```