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 Брянская Е.В.

Группа ИУ7-62Б

Оценка (баллы) ______________

Преподаватель Градов В.М.

Москва.
2021 г.

Задание

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.

Исходные данные.
Задана система электротехнических уравнений, описывающих разрядный контур, включающий постоянное активное сопротивление R_k, нелинейное сопротивление $R_p(I)$, зависящее от тока I, индуктивность L_k и емкость C_k.

$$\begin{cases} \frac{dI}{dT} = \frac{U-(R_k+R_p(I))I}{L_k}, \\ \frac{dU}{dt} = -\frac{I}{C_k} \end{cases} \quad (1)$$

Начальные условия:
$t = 0, I = I_0, U = U_0$

Здесь I, U - ток и напряжение на конденсаторе.
Сопротивление R_p рассчитать по формуле:

$$R_p = \frac{l_p}{2\pi R^2 \int\limits_0^1 \sigma(T(z))z dz} \quad (2)$$

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	T_0, 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	$\sigma, 1/$
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

Параметры разрядного контура:

R = 0.35 см

l = 12 см

$L_k = 187 \cdot 10^{-6}$ Гн

$C_k = 268 \cdot 10^{-6}$ Ф

$R_k = 0.25$ Ом

$U_{co} = 1400$ В

$I_o = 0..3$ А

$T_w = 2000$ К

Выполнение

Задача решается методом Рунге-Кутта 4ого порядка для системы ОДУ.

$$y_{n+1} = y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}, \qquad z_{n+1} = z_n + \frac{p_1 + 2p_2 + 2p_3 + p_4}{6} \tag{3}$$

где

$$k_1 = hf(x_n, y_n, z_n) \qquad p_1 = h\varphi(x_n, y_n, z_n) \tag{4}$$

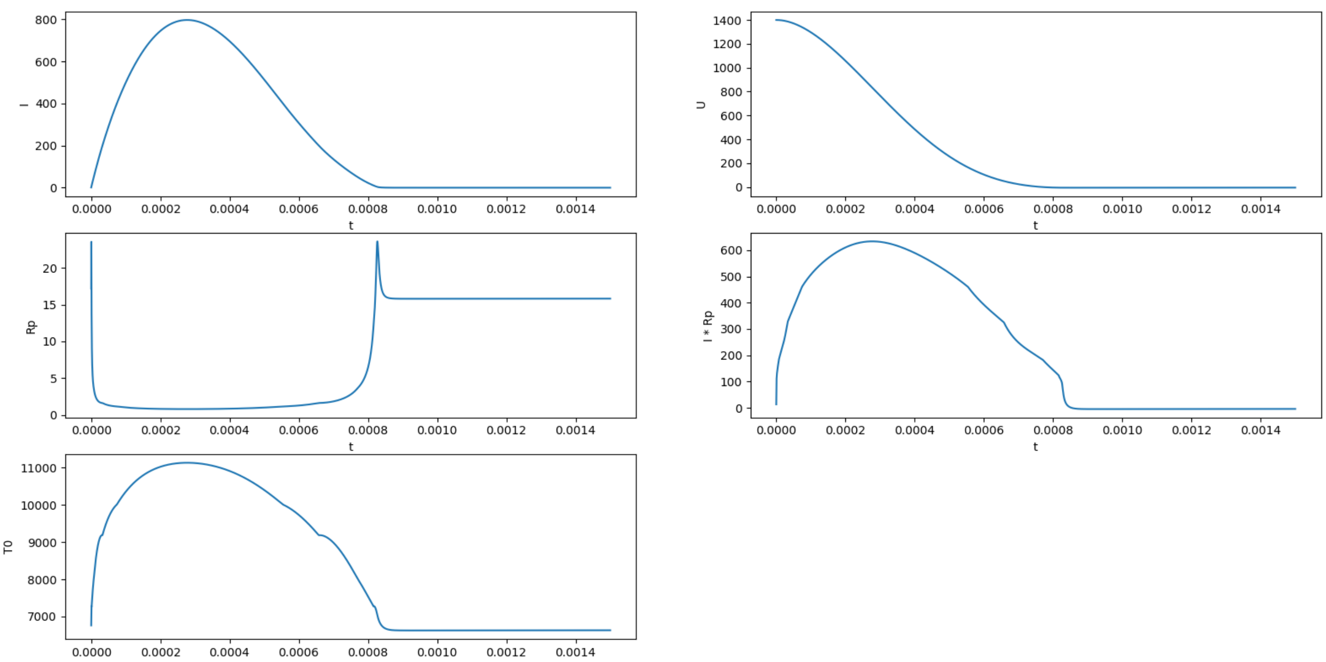
$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{p_1}{2}) \qquad p_2 = h\varphi(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{p_1}{2}) \tag{5}$$

$$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{p_2}{2}) \qquad p_3 = h\varphi(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{p_2}{2}) \tag{6}$$

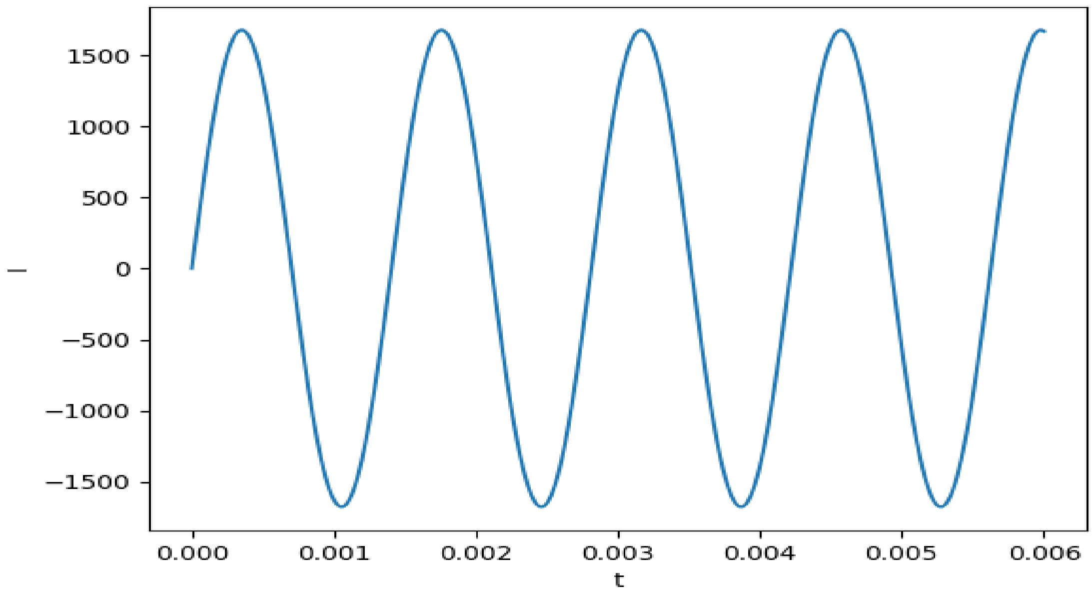
$$k_4 = hf(x_n + \frac{h}{2}, y_n + k_3, z_n + p_3) \qquad p_4 = h\varphi(x_n + \frac{h}{2}, y_n + k_3, z_n + p_3) \tag{7}$$

Результат работы программы.

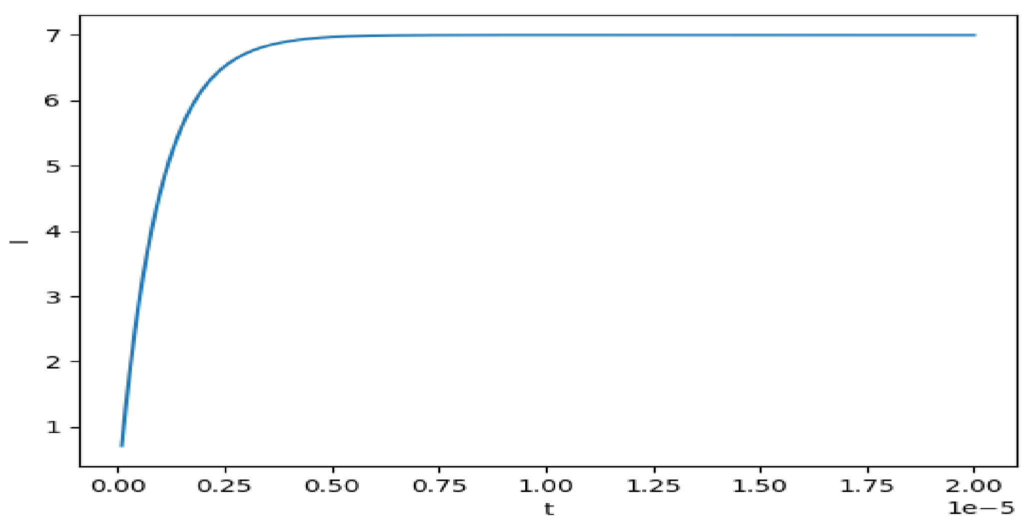
1. Графики зависимости от времени импульса t: $I(t), U(t), R_p(t)$, произведения $I(t) \cdot R_p(t), T_0(t)$ при заданных выше параметрах. Шаг сетки - 10^{-7}.



2. График зависимости $I(t)$ при $R_k + R_p = 0$. Обратить внимание на то, что в этом случае колебания тока будут незатухающими.



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 tимп. апериодической формы. Длительность импульса определяется по

кривой зависимости тока от времени на высоте $35I_{max}$, I_{max} - значение тока в максимуме (см. рисунок).

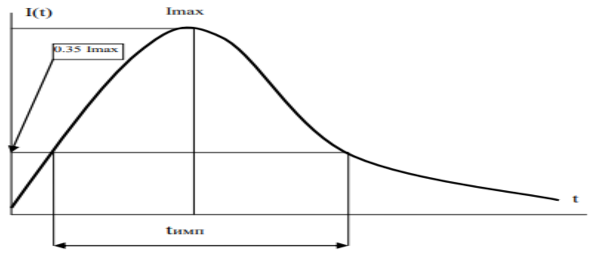


Таблица 3 — Влияние C_k на длительность импульса tимп

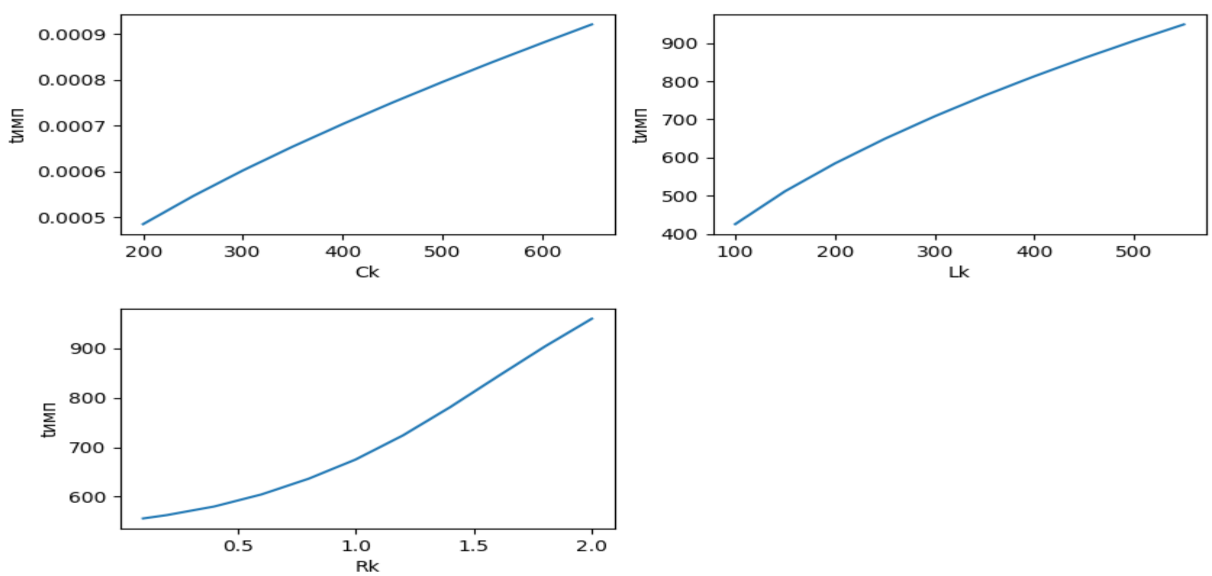
C_k, мкФ	200	250	300	350	400	450	500	550	600	650
tимп, мкс	485.10	546.10	602.10	654.20	703.20	749.90	794.70	838.0	879.70	920.40

Таблица 4 — Влияние L_k на длительность импульса tимп

L_k, мкГн	100	150	200	250	300	350	400	450	500	550
tимп, мкс	425.50	511.80	584.70	649.30	708.20	762.50	813.20	861.00	906.20	949.20

Таблица 5 — Влияние R_k на длительность импульса tимп

R_k, Ом	0.1	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
tимп, мкс	556.20	562.90	580.2	604.5	636.3	675.3	723.9	781.01	842.6	903.2	959.7



В результате исследования было выявлено, что при увеличении любого из трёх параметров увеличивается и длительность импульса.

Вопросы при защите лабораторной работы

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

   Можно сравнить результаты работы программы с теми, которые получаются при аналитическом решении, так как представленная задача является слишком сложной для такого подхода, то можно упростить задачу, изменив некоторые условия. Например, при $R_k + R_p = 0$ система представляет собой идеальный колебательный контур, что позволяет оценить правильность работы по ряду характеристик.

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

$$\begin{cases} U_{n+1} = U_n - \dfrac{h}{2C_k}(I_n + I_{n+1}) \\ I_{n+1} = I_n + \dfrac{h}{2L_k}(U_n - (R_k + R_p(I_n))I_n + U_{n+1} - (R_k + R_p(I_{n+1}))I_{n+1}) \end{cases} \tag{8}$$

   Сначала следует найти $R_p(I_{n+1})$, так как он имеет нелинейную зависимость от I, выраженный через интерполяцию таблицы значений, и поэтому выразить I_{n+1} через R_p крайне сложно. В свою очередь, R_p можно найти, подставив в него значение I_{n+1}, найденное с помощью метода Эйлера. После этого, подставив все соответствующие значения, можно получить I_{n+1} по второй формуле из системы выше. И далее уже можно получить U_{n+1}.

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?**

   Это зависит от конкретной задачи, какую точность она требует. Увеличивать порядок точности нужно в том случае, если это существенно (относительно погрешности) повлияет на результат.

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 $t = 0$, т.е. $t = 0, U = U_0$ а ток задан в другой момент времени, к примеру, в конце импульса, т.е. при $t = T, I = I_T$. Какой можете предложить алгоритм вычислений?**

Сначала выберем ток в момент времени $t = 0$, и получим решение реализованным методом Рунге-Кутта для I в $t = T$. Далее оценим невязку с заданной по условию величиной и повторим вычисления для другого тока в $t = 0$. Перебирая значения тока при $t = 0$, нужно свести невязку к минимальному значению.

Но решение поставленной задачи возможно только в случае, если заданные условия ведут только к одному решению. Например, если одним из условий будет то, что $I = 0$ в момент времени $t = 1$ мс, то будет существовать бесконечное количество решений, удовлетворяющих данному условию.

Код программы

Листинг 1 — Лабораторная работа №2

```
from math import *
import matplotlib.pyplot as plt

table_I = [0.5, 1, 5, 10, 50, 200, 400, 800, 1200]
table_T0 = [6730, 6790, 7150, 7270, 8010, 9185, 10010, 11140, 12010]
table_m = [0.5, 0.55, 1.7, 3, 11, 32, 40, 41, 39]
table_T = [4000,5000,6000,7000,8000,9000,10000,11000,12000,13000,14000]
table_sigma = [0.031,0.27,2.05,6.06,12,19.9,29.6,41.1,54.1,67.7,81.5]

R = 0.35
le = 12
Lk = 187 * 10 ** (-6)
Ck = 268 * 10 ** (-6)
Rk = 0.25
Tw = 2000

T0 = 0
Rp = 0

res_I = []
res_U = []
res_t = []
res_Rp = []
res_IRp = []
res_T0 = []

def find_sigma(z, I, Tw):
    t0 = interpolation(I, 2, table_I, table_T0)
    global T0
    T0 = t0
    m = interpolation(I, 2, table_I, table_m)
    T = T0 + (Tw - T0) * z ** m
    sigma = interpolation(T, 2, table_T, table_sigma)
    return sigma

def find_Rp(I, Tw):
    a, b = 0, 1
```

```python
    n = 100
    dz = (b - a) / n
    intgr = 0
    z = 0

    for j in range(n):
        intgr += z * dz * find_sigma(z, I, Tw)
        z += dz

    return Ie / (2 * pi * R * R * intgr)

def f(I, U):
    global Rp
    Rp = find_Rp(I, Tw)
    return (U - (Rk + Rp) * I) / Lk

def phi(I):
    return -I / Ck

def find_I_U(I, U, h):
    k1 = h * f(I, U)
    p1 = h * phi(I)

    k2 = h * f(I + k1 / 2, U + p1 / 2)
    p2 = h * phi(I + k1 / 2)

    k3 = h * f(I + k2 / 2, U + p2 / 2)
    p3 = h * phi(I + k2 / 2)

    k4 = h * f(I + k3, U + p3)
    p4 = h * phi(I + k3)

    return I + 1 / 6 * (k1 + 2 * k2 + 2 * k3 + k4), \
        U + 1 / 6 * (p1 + 2 * p2 + 2 * p3 + p4)

def find_position(x, data_x, number):
    left = 0
    right = number - 1

    if data_x[0] < data_x[-1]:
```

```
    if x < data_x[left] or x > data_x[right]: return -1
    while left + 1 != right:
      pos_x = int((left + right)/2)
      if data_x[pos_x] <= x:
        left = pos_x
      else:
        right = pos_x
    return left
  else:
    if x < data_x[right] or x > data_x[left]: return -1
    while left + 1 != right:
      pos_x = int((left + right)/2)
      if data_x[pos_x] >= x:
        left = pos_x
      else:
        right = pos_x
    return left

def find_range(pos_x, degree, data_x, data_y, number):
  half = int(degree / 2)
  left = pos_x - half
  right = pos_x + (degree - half)
  if left < 0:
    right += -left
    left = 0
  elif right > number - 1:
    left -= right - (number - 1)
    right = number - 1

  return data_x[left: right + 1], data_y[left: right + 1]

def find_polynomial(degree, nodes_x, nodes_y):
  div_diff, old_arr = [0] * (degree + 1), nodes_y
  new_arr = [0] * degree

  div_diff[0] = old_arr[0]
  for i in range(degree):
    for j in range(degree - i):
      new_arr[j] = (old_arr[j] - old_arr[j+1]) / \
      (nodes_x[j] - nodes_x[j+i+1])
```

```
      div_diff[i+1] = new_arr[0]
      old_arr = new_arr

   return lambda x: nwtn_polynom(x, degree, div_diff, nodes_x)

def nwtn_polynom(x, degree, div_diff, nodes_x):
   y = div_diff[0]
   x_pl = 1
   for i in range(1, degree + 1):
      x_pl *= (x - nodes_x[i-1])
      y += x_pl * div_diff[i]
   return y

def interpolation(x, degree, data_x, data_y):
   pos = find_position(x, data_x, len(data_x))
   nodes_x, nodes_y = find_range(pos,degree,data_x, data_y, len(data_x))
   f = find_polynomial(degree, nodes_x, nodes_y)
   return f(x)

def show_graph():
   /* Построение графиков */

def main():
   Uc = 1400
   I, t = 0, 0
   h = 10 ** (-7)

   while t < 1.5 * 10 ** (-3):
      I, Uc = find_I_U(I, Uc, h)
      t += h

      res_I.append(I)
      res_U.append(Uc)
      res_t.append(t)
      res_T0.append(T0)
      res_Rp.append(Rp)

   for i in range(len(res_Rp)):
      res_IRp.append(res_I[i] * res_Rp[i])

```

```python
    show_graph()

if __name__ == '__main__':
    main()
```