

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Тема** Программно-алгоритмическая реализация метода Рунге-Кутты 4-го порядка точности  
при решении задачи Коши для системы ОДУ

Преподаватель    Градов В.М.

Москва — 2021 г.

# Оглавление

<b>1</b>	<b>Теоретические сведения</b>	<b>3</b>
1.1	Метод Рунге-Кутты четвертого порядка точности . . . . .	4
<b>2</b>	<b>Реализация</b>	<b>6</b>
2.1	Код программы . . . . .	6
2.2	Результаты работы программы . . . . .	11
<b>3</b>	<b>Ответы на вопросы</b>	<b>19</b>

## Тема работы

Программно-алгоритмическая реализация метода Рунге-Кутты 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-Кутты 4-го порядка точности.

# 1 Теоретические сведения

Опишем колебательный контур с помощью системы уравнений:

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

Значение  $R_p(I)$  можно вычислить по формуле:

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

т. к.  $z = r/R$ .

Значение  $T(z)$  вычисляется по формуле:

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

Заданы начальные параметры:

$R = 0.35$  см (Радиус трубки)

$l_e = 12$  см (Расстояние между электродами лампы)

$L_k = 187e-6$  Гн (Индуктивность)

$C_k = 268e-6$  Ф (Емкость конденсатора)

$R_k = 0.25$  Ом (Сопротивление)

$U_{c0} = 1400$  В (Напряжение на конденсаторе в начальный момент времени)

$I_0 = 0..3$  А (Сила тока в цепи в начальный момент времени  $t = 0$ )

$T_w = 2000$  К

## 1.1 Метод Рунге-Кутты четвертого порядка ТОЧНОСТИ

Имеем систему уравнений вида:

$$\begin{cases} u'(x) = f(x, u(x)) \\ u(\xi) = \eta \end{cases}$$

Тогда:

$$\begin{aligned} y_{n+1} &= y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \\ k_1 &= h_n f(x_n, y_n) \\ k_2 &= h_n f(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= h_n f(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= h_n f(x_n + h_n, y_n + k_3) \end{aligned}$$

Рассмотрим обобщение формулы на случай двух переменных. Пусть дана система:

$$\begin{cases} u'(x) = f(x, u, v) \\ v'(x) = \varphi(x, u, v) \\ v(\xi) = v_0 \\ u(\xi) = u_0 \end{cases}$$

Тогда:

$$\begin{aligned} y_{n+1} &= y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \\ z_{n+1} &= z_n + \frac{q_1 + 2q_2 + 2q_3 + q_4}{6} \\ k_1 &= h_n f(x_n, y_n, z_n) \\ k_2 &= h_n f(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2}, z_n + \frac{q_1}{2}) \\ k_3 &= h_n f(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2}, z_n + \frac{q_2}{2}) \\ k_4 &= h_n f(x_n + h_n, y_n + k_3, z_n + q_3) \\ q_1 &= h_n \varphi(x_n, y_n, z_n) \\ q_2 &= h_n \varphi(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2}, z_n + \frac{q_1}{2}) \end{aligned}$$

$$q_3 = h_n \varphi(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2}, z_n + \frac{q_2}{2})$$

$$q_4 = h_n \varphi(x_n + h_n, y_n + k_3, z_n + q_3)$$

## 2 Реализация

### 2.1 Код программы

Ниже представлены исходные коды программы на языке **Elixir**.

Листинг 2.1: Основной модуль приложения

```
1 defmodule RungeKutta.Application do
2   use Application
3   import RungeKutta.Runners
4
5   def start(_, _) do
6     run()
7     {:ok, self()}
8   end
9
10  def run() do
11    [from, to, step] =
12      IO.gets("Input boundaries and step using space as separator: ")
13      |> String.trim()
14      |> String.split()
15      |> Enum.map(fn str_float ->
16        {num, _} = Float.parse(str_float)
17        num
18      end)
19
20    run_simple_graphs(from, to, step)
21    run_const_resistance(from, to, step, 0)
22    run_const_resistance(from, to, step, 200)
23  end
24 end
```

Листинг 2.2: Модуль с интерполяционными функциями

```
1 defmodule RungeKutta.InterpolatedFuncs do
2   import RungeKutta.Integral, only: [trapezoid: 4]
3   @step 0.05
4
5   @r 0.35
6   @lp 12
7   @tw 2000
8
9   @t0_I [
10     {0.5, 6730},
11     {1, 6790},
12     {5, 7150},
```

```

13     {10, 7270},
14     {50, 8010},
15     {200, 9185},
16     {400, 10010},
17     {800, 11140},
18     {1200, 12010}
19 ]
20 @m_I [
21     {0.5, 0.5},
22     {1, 0.55},
23     {5, 1.7},
24     {10, 3},
25     {50, 11},
26     {200, 32},
27     {400, 40},
28     {800, 41},
29     {1200, 39}
30 ]
31 @sigma_T [
32     {4000, 0.031},
33     {5000, 0.27},
34     {6000, 2.05},
35     {7000, 6.06},
36     {8000, 12.0},
37     {9000, 19.9},
38     {10000, 29.6},
39     {11000, 41.1},
40     {12000, 54.1},
41     {13000, 67.7},
42     {14000, 81.5}
43 ]
44
45 def m(i) do
46     linear_interpolation(@m_I, i)
47 end
48
49 def t0(i) do
50     linear_interpolation(@t0_I, i)
51 end
52
53 def sigma(z, t0_val, m_val) do
54     t = t0_val + (@tw - t0_val) * :math.pow(z, m_val)
55     linear_interpolation(@sigma_T, t)
56 end
57
58 def rp(i) do
59     t0_val = t0(i)
60     m_val = m(i)

```



```

61     integral = trapezoid(0, 1, @step, fn z -> sigma(z, t0_val, m_val) * z end)
62     @lp / (2 * :math.pi() * @r * @r * integral)
63 end
64
65 def linear_interpolation(table, arg) do
66     {{x1, y1}, {x2, y2}} = find_closest_pair(table, arg)
67     y1 + (y2 - y1) / (x2 - x1) * (arg - x1)
68 end
69
70 def find_closest_pair(table, arg) do
71     table
72     |> Stream.zip(Stream.drop(table, 1))
73     |> Enum.reduce_while(nil, fn {{x1, _}, _} = pair, acc ->
74         cond do
75             x1 >= arg ->
76                 case acc do
77                     nil -> {:halt, pair}
78                     _ -> {:halt, acc}
79                 end
80             true -> {:cont, pair}
81         end
82     end)
83 end
84 end

```

### Листинг 2.3: Модуль с функциями прогонки с разными параметрами

```

1 defmodule RungeKutta.Runners do
2     import RungeKutta.Helper
3     import RungeKutta.Solver
4     alias RungeKutta.Plot
5     import RungeKutta.MainFuncs, only: [f_const: 4]
6
7     def run_simple_graphs(from, to, step) do
8         xs = float_range_generator(from, to, step)
9         {ys, zs} = Enum.unzip(generate_iu(from, to, step, &RungeKutta.MainFuncs.f/3))
10        rpns = generate_rp(from, to, step)
11        t0s = generate_t0(from, to, step)
12        #xs
13        #|> Enum.zip(rpns)
14        #|> Enum.zip(t0s)
15        #|> Enum.zip(ys)
16        #|> Enum.zip(zs)
17        #|> Enum.map(fn {{x, rp}, t0, y, z} ->
18            #IO.puts "x = #{x}, Rp = #{rp}, T0 = #{t0}, I = #{y}, U = #{z};"
19        end)
20        plots = Plot.init_plot_collection()
21        plots = Plot.plot(plots, xs, ys, "I")
22        plots = Plot.plot(plots, xs, zs, "U")

```

```

23 plots = Plot.plot(plots, xs, rpns, "Rp")
24 rp_mul_i = rpns |> Stream.zip(ys) |> Enum.map(fn {rp, i} -> rp * i end)
25 plots = Plot.plot(plots, xs, rp_mul_i, "Rp * I")
26 # yn * zn
27 plots = Plot.plot(plots, xs, t0s, "T0")
28 Plot.show_all(plots)
29 end
30
31 def run_const_resistance(from, to, step, c \\ 0) do
32   xs = float_range_generator(from, to, step)
33   {ys, zs} = Enum.unzip(generate_iu(from, to, step, fn x, y, z -> f_const(x, y, z, c)
34     end))
35   plots = Plot.init_plot_collection()
36   plots = Plot.plot(plots, xs, ys, "I (R = #{c})")
37   plots = Plot.plot(plots, xs, zs, "U (R = #{c})")
38   Plot.show_all(plots)
39 end
end

```

Листинг 2.4: Модуль с методами численного интегрирования

```

1 defmodule RungeKutta.Integral do
2   import RungeKutta.Helper, only: [float_range_map: 5]
3
4   def trapezoid(from, to, step, func) do
5     float_range_map(from, to + step / 2, step, [], fn val, _acc -> func.(val) end)
6     |> Stream.chunk_every(2, 1, :discard)
7     |> Enum.reduce(0, fn [a, b], acc -> acc + step * (a + b) / 2 end)
8   end
9
10  def simpson(from, to, step, func) do
11    n = Float.round((to - from) / step)
12
13    float_range_map(from, to + step / 4, step / 2, [], fn val, _acc -> func.(val) end)
14    |> Enum.reverse()
15    |> (fn vals -> (to - from) / 6 / n * count_all_sums(vals) end).()
16  end
17
18  defp count_all_sums(vals) do
19    all = Enum.reduce(vals, &Kernel.+/2)
20    odds = Stream.drop(vals, 1) |> Stream.take_every(2) |> Enum.reduce(&Kernel.+/2)
21    vals = Stream.drop(vals, 2) |> Enum.reverse()
22    evens = Stream.drop(vals, 2) |> Stream.take_every(2) |> Enum.reduce(&Kernel.+/2)
23    all + odds + 3 * evens
24  end
25 end

```

Листинг 2.5: Модуль с генерацией искоемых значений

```

1 defmodule RungeKutta.Solver do
2   @u0 1400
3   @i0 0.5
4   import RungeKutta.Helper, only: [float_range_map: 5]
5   import RungeKutta.InterpolatedFuncs, only: [t0: 1, rp: 1]
6   import RungeKutta.MainFuncs, only: [phi: 3]
7
8   def generate_t0(from, to, step) do
9     generate_iu(from, to, step, &RungeKutta.MainFuncs.f/3)
10    |> Stream.map(fn {yn, _zn} -> yn end)
11    |> Enum.map(fn i -> t0(i) end)
12  end
13
14  def generate_iu(from, to, step, f) do
15    float_range_map(from, to, step, [{@i0, @u0}], fn xn, [{yn, zn} | _] ->
16      k1 = step * f.(xn, yn, zn)
17      p1 = step * phi(xn, yn, zn)
18      k2 = step * f.(xn + step / 2, yn + k1 / 2, zn + p1 / 2)
19      p2 = step * phi(xn + step / 2, yn + k1 / 2, zn + p1 / 2)
20      k3 = step * f.(xn + step / 2, yn + k2 / 2, zn + p2 / 2)
21      p3 = step * phi(xn + step / 2, yn + k2 / 2, zn + p2 / 2)
22      k4 = step * f.(xn + step, yn + k3, zn + p3)
23      p4 = step * phi(xn + step, yn + k3, zn + p3)
24
25      {yn + (k1 + 2 * k2 + 2 * k3 + k4) / 6, zn + (p1 + 2 * p2 + 2 * p3 + p4) / 6}
26    end)
27    |> Enum.reverse()
28  end
29
30  def generate_rp(from, to, step) do
31    generate_iu(from, to, step, &RungeKutta.MainFuncs.f/3)
32    |> Stream.map(fn {yn, _zn} -> yn end)
33    |> Enum.map(fn i -> rp(i) end)
34  end
35 end

```

## Листинг 2.6: Модуль с функциями тока и напряжения

```

1 defmodule RungeKutta.MainFuncs do
2   @ck 268.0e-6
3   @rk 0.25
4   @lk 187.0e-6
5   import RungeKutta.InterpolatedFuncs, only: [rp: 1]
6
7   def f(_x, u, v) do
8     rpp = rp(u)
9     (v - (@rk + rpp) * u) / @lk
10  end

```

```

11
12 def f_const(_x, u, v, c) do
13     (v - c * u) / @lk
14 end
15
16 def phi(_x, u, _v) do
17     -u / @ck
18 end
19 end

```

## 2.2 Результаты работы программы

На рисунках 2.1 – 2.5 представлены графики зависимости от времени импульса  $t$ :  $I(t)$ ,  $U(t)$ ,  $R_p(t)$ ,  $I(t) \cdot R_p(t)$ ,  $T_0(t)$  при исходных данных. Интервал:  $[0, 0.0008]$ , шаг  $h = 10^{-6}$ .

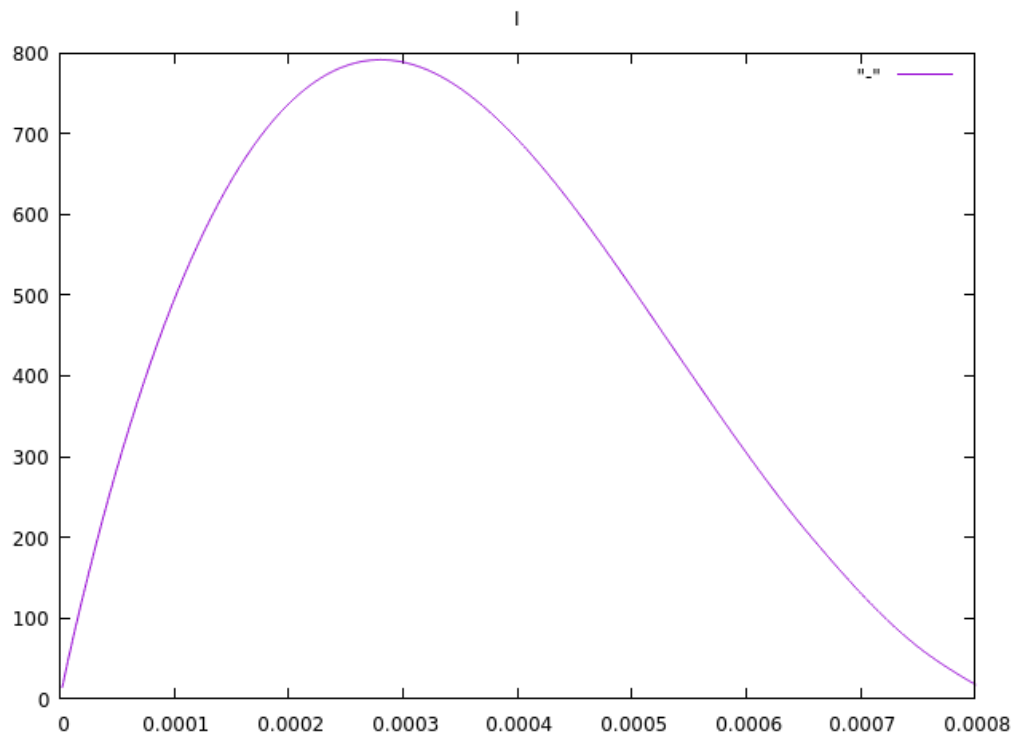


Рис. 2.1: График  $I(t)$

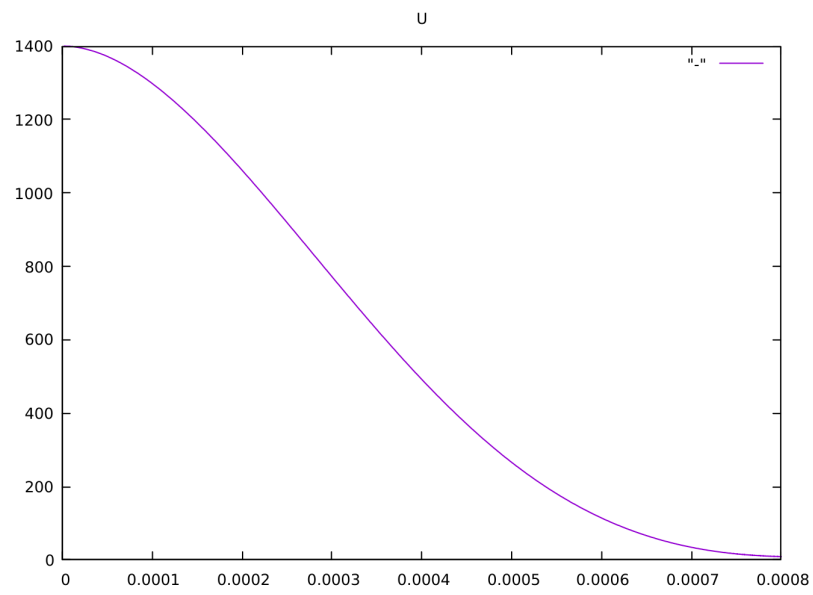


Рис. 2.2: График  $U(t)$

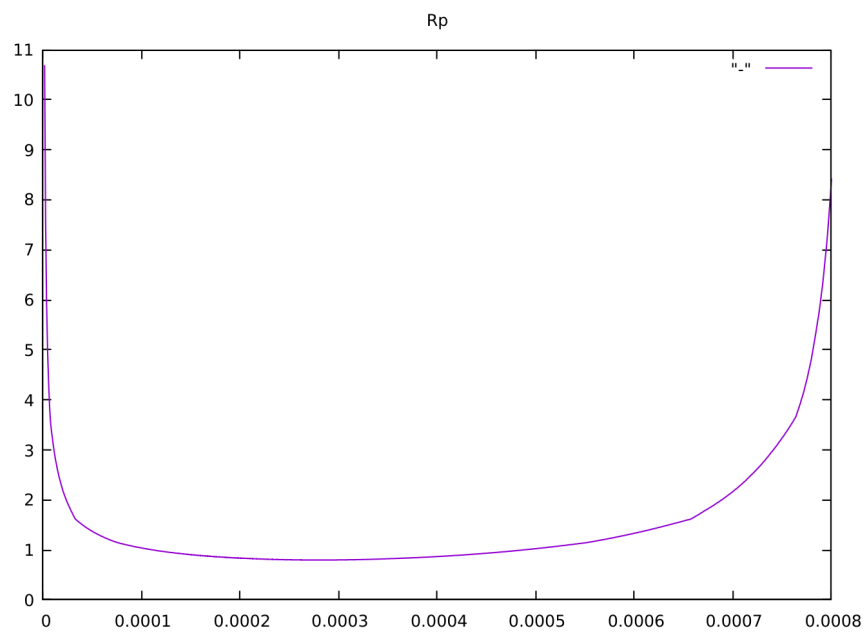


Рис. 2.3: График  $R_p(t)$

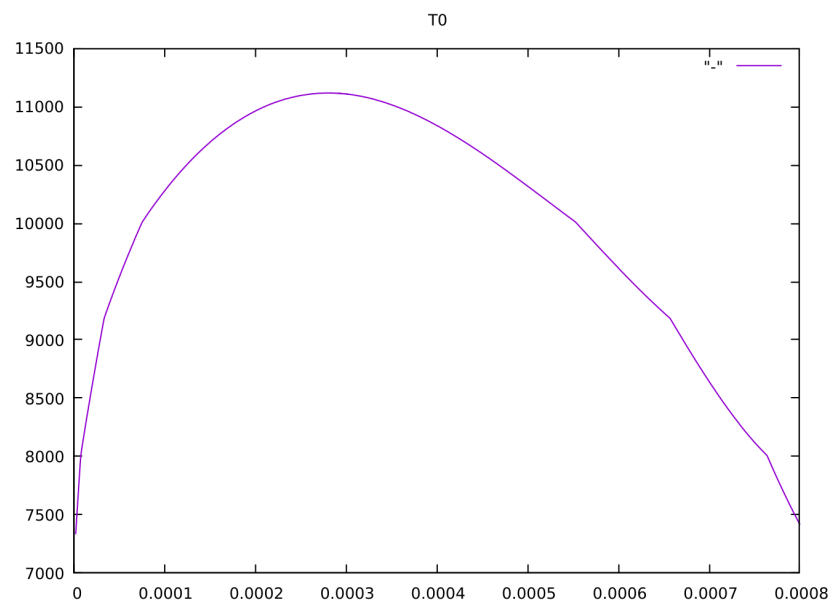


Рис. 2.4: График  $T_0(t)$

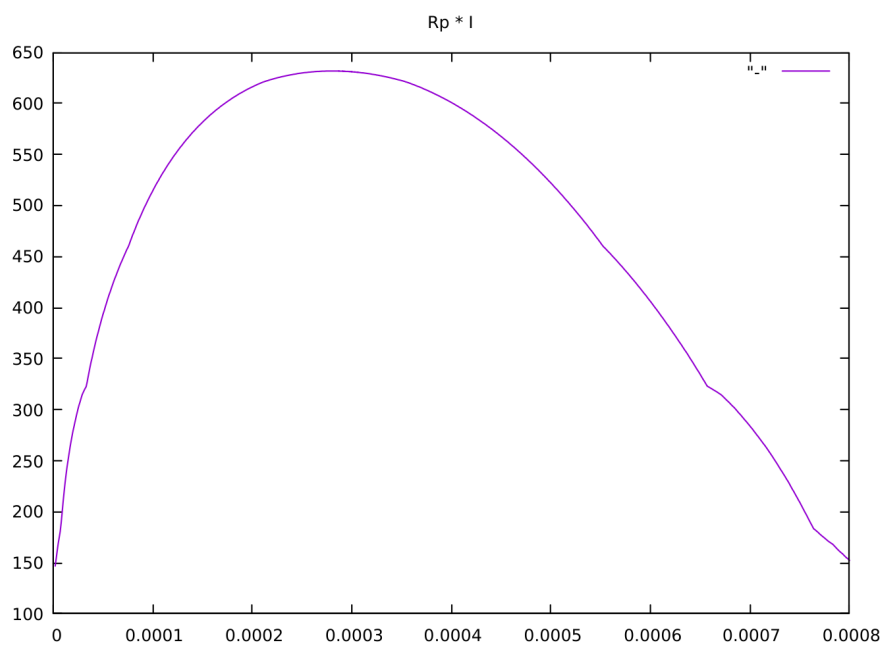


Рис. 2.5: График  $I(t) \cdot R_p(t)$

На рисунке 2.6 представлен график  $I(t)$ , при  $R_k + R_p = 0$ . Интервал:  $[0, 0.0008]$ , шаг  $h = 10^{-6}$ .

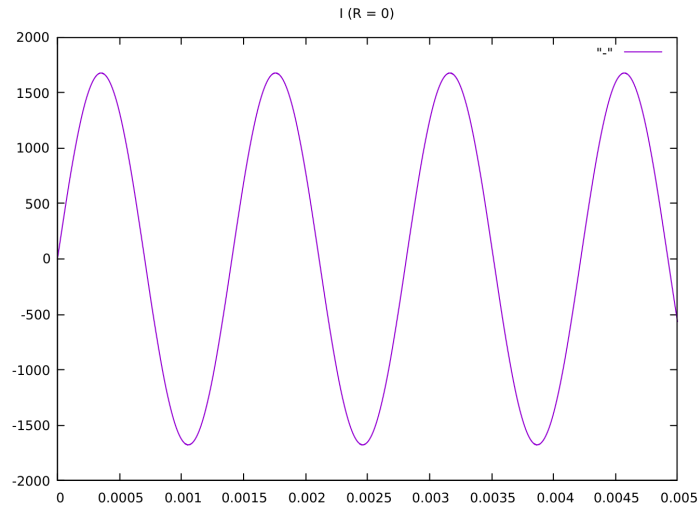


Рис. 2.6: График зависимости  $I(t)$  при  $R = 0$

На рисунке 2.7 представлен график  $I(t)$ , при  $R_k + R_p = 200$ . Интервал:  $[0, 0.00002]$ , шаг  $h = 10^{-7}$ .

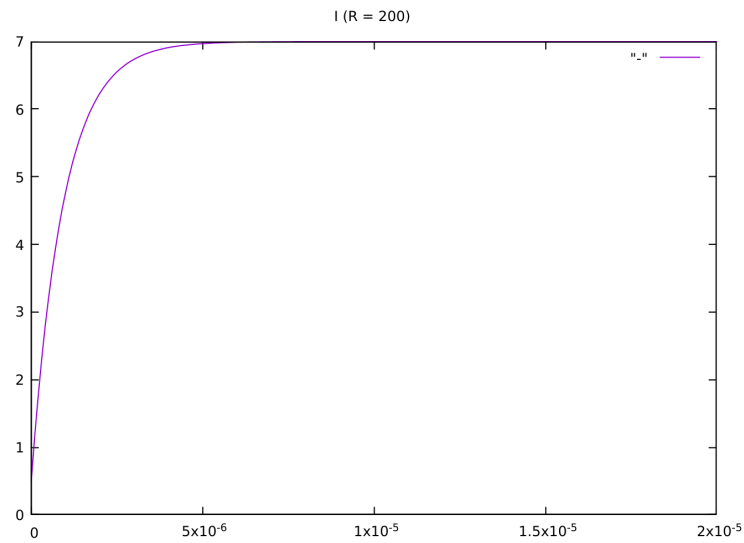


Рис. 2.7: График зависимости  $I(t)$  при  $R = 200$

На рисунках 2.8 - 2.14 представлены результаты исследования влияния параметров контура  $C_k$ ,  $L_k$  и  $R_k$  на длительность импульса  $t$ .

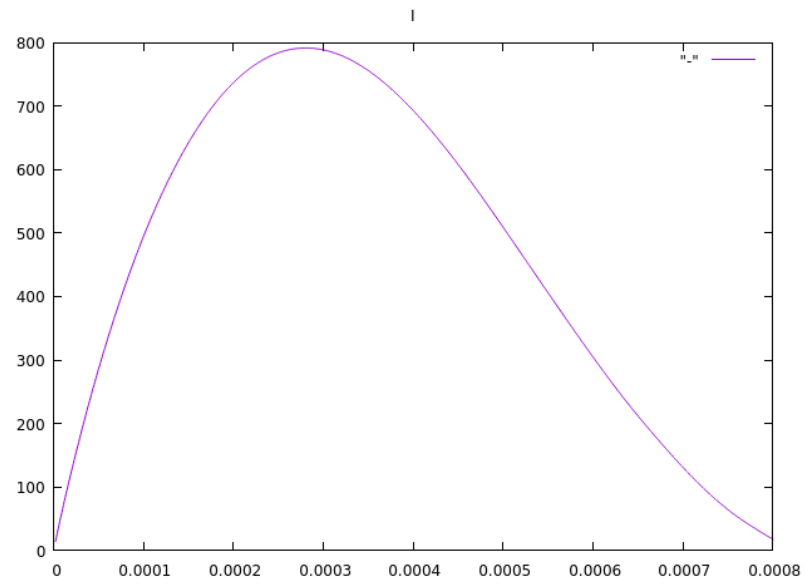


Рис. 2.8: График зависимости  $I(t)$  при начальных параметрах

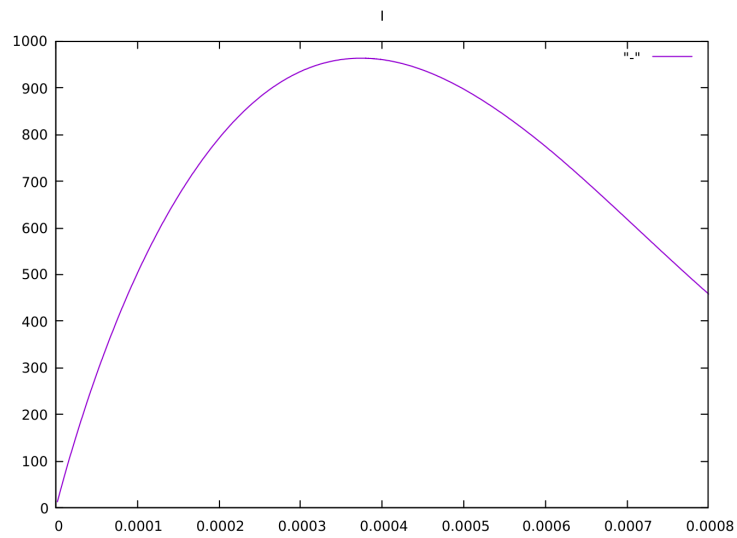


Рис. 2.9: График зависимости  $I(t)$  при увеличении начального значения  $C_k$  в 2 раза.



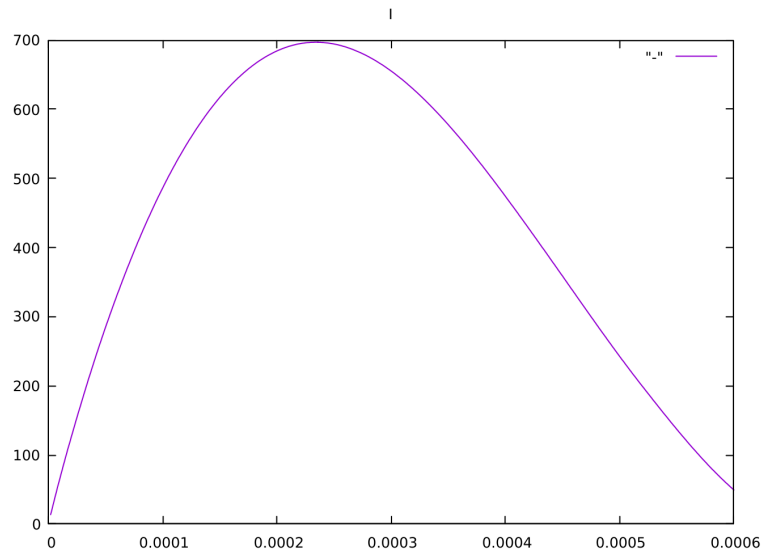


Рис. 2.10: График зависимости  $I(t)$  при уменьшении начального значения  $C_k$  в 1.5 раза.

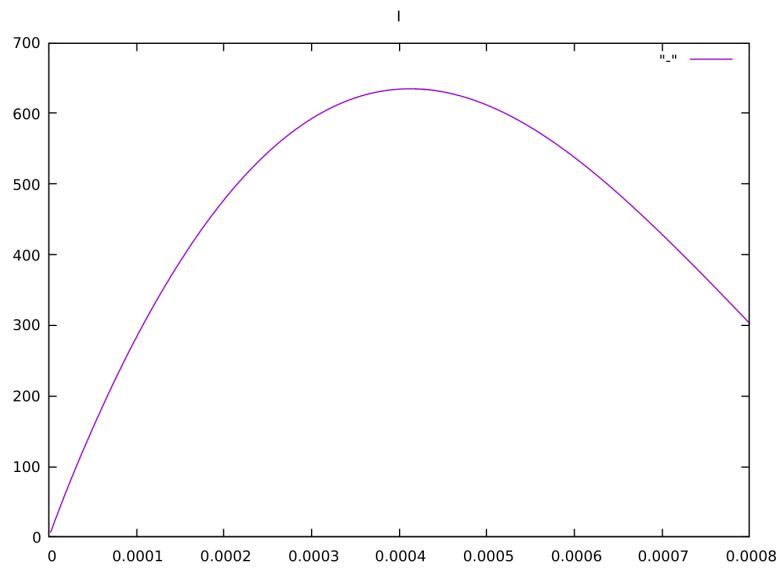


Рис. 2.11: График зависимости  $I(t)$  при увеличении начального значения  $L_k$  в 2 раза

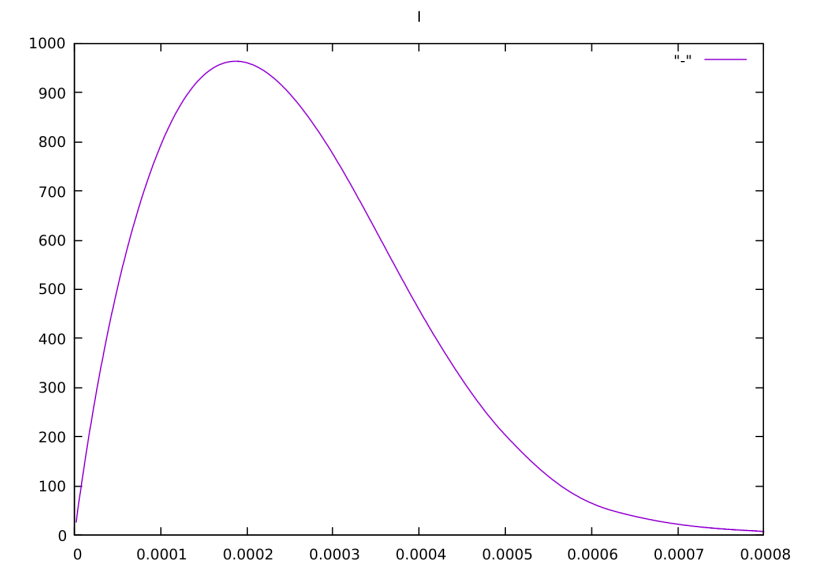


Рис. 2.12: График зависимости  $I(t)$  при уменьшении начального значения  $L_k$  в 2 раза

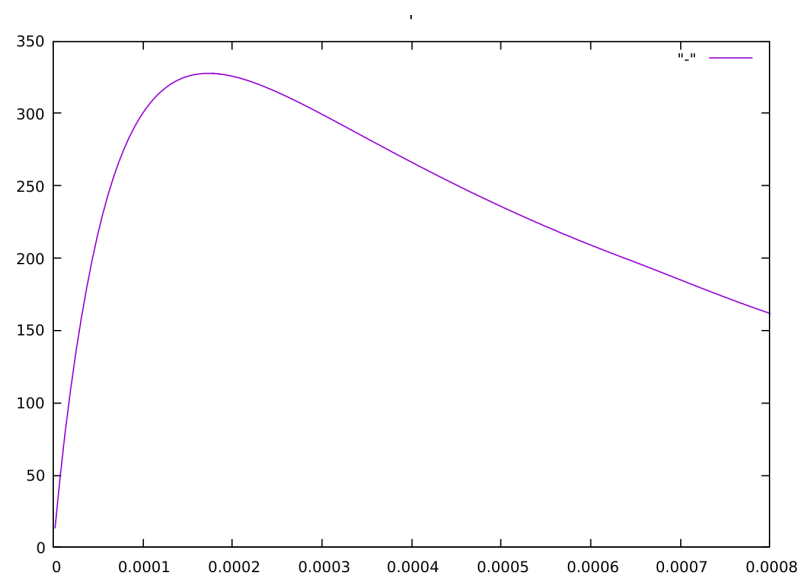


Рис. 2.13: График зависимости  $I(t)$  при увеличении начального значения  $R_k$  в 10 раз

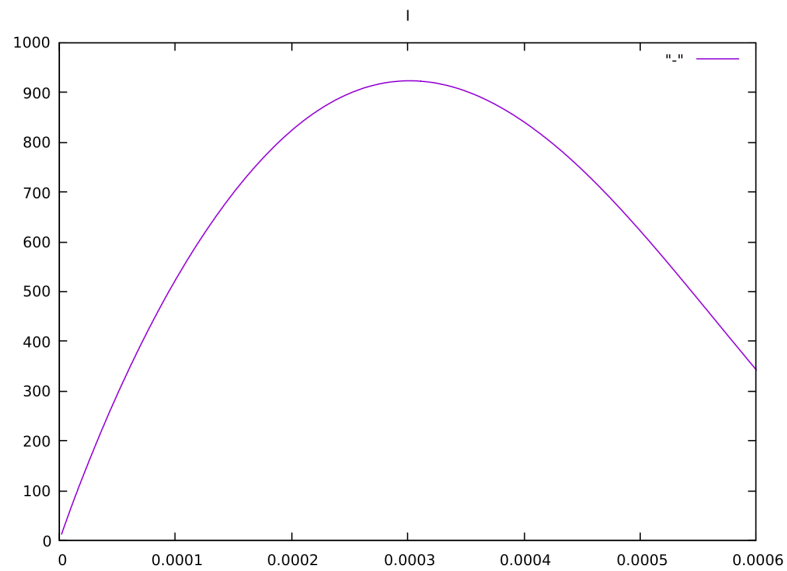


Рис. 2.14: График зависимости  $I(t)$  при уменьшении начального значения  $R_k$  в 10 раз

Выводы:

- увеличение  $C_k$  приводит к увеличению длительности импульса  $t$ ;
- уменьшение  $C_k$  приводит к уменьшению длительности импульса  $t$ ;
- увеличение  $L_k$  приводит к увеличению длительности импульса  $t$ ;
- уменьшение  $L_k$  приводит к уменьшению длительности импульса  $t$ ;
- увеличение  $R_k$  приводит к увеличению длительности импульса  $t$ ;
- уменьшение  $R_k$  приводит к уменьшению длительности импульса  $t$ ;

### 3 Ответы на вопросы

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

В цепи мы можем влиять на 3 параметра (ёмкость, индуктивность, сопротивление), при чем влияние каждого параметра по отдельности мы рассмотрели в п.2. При этом особый интерес представляет изменение сопротивления (потому что изменение ёмкости и индуктивности по большей мере влияют на изменение длительности импульса (так, при нулевом сопротивлении, график всегда будет представлен синусоидой, а изменение параметров будет влиять только на значение ее экстремумов и период)) путем добавления/удаления/изменения объектов, обладающих сопротивлением (лампа, резистор), при этом: если  $R_k$  мало, то возникнут затухающие колебания, если велико  $R_k$  — апериодическое затухание.

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

$$U_{n+1} = U_n + \frac{h}{2} + f(x_n, u_n) + f(x_{n+1}, u_{n+1}) + O(h^2) \quad (3.1)$$

$$\begin{cases} \frac{dI}{dT} = \frac{U - (R_k + R_p(I))I}{L_k} \\ \frac{dU}{dt} = -\frac{I}{C_k} \end{cases} \quad (3.2)$$

$$I_{n+1} = I_n + \frac{h}{2} \left( \frac{U_n - (R_k + R_p(I_n))I_n}{L_k} + \frac{U_{n+1} - (R_k + R_p(I_{n+1}))I_{n+1}}{L_k} \right) \quad (3.3)$$

$$U_{n+1} = U_n + \frac{h}{2} \left( -\frac{I_n}{C_k} - \frac{I_{n+1}}{C_k} \right) = U_n - \frac{h}{2} \left( \frac{I_n + I_{n+1}}{C_k} \right) \quad (3.4)$$

Подставляя (3.4) в (3.3), имеем:

$$I_{n+1} = I_n + \frac{h}{2L_k} (2U_n - (R_k + R_p(I_n) + \frac{h}{2C_k})I_n - (R_k + R_p(I_{n+1}) + \frac{h}{2C_k})I_{n+1}) \quad (3.5)$$

**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?**

Выбор численного метода проводится исходя из требований поставленной задачи, объема вычислений, а также от свойств функций, используемых в вычислениях.

Оценивается погрешность для частного случая вида правой части дифференциального уравнения:  $\varphi(x, \nu) = \varphi(x)$

Так, если  $\varphi(x, \nu)$  непрерывна и ограничена и ограничены и непрерывны её четвертые производные, то наилучший результат достигаем при использовании метода Рунге-Кутты 4 порядка:

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

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

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

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

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

Однако в случае если  $\varphi(x, \nu)$  не имеет таких производных, то четвертый порядок схемы не может быть достигнут и стоит применять более простые схемы.