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»

Отчёт по лабораторной работе №2
По дисциплине «Математическое моделирование»
Тема: «Программно - алгоритмические реализации методов Рунге-Кутта 2-го и 4-го порядков точности при решении системы ОДУ в задаче Коши»

Студент: Барсуков Н.М.

Группа: ИУ7-66

Преподаватель: Градов В.М

Москва, 2020

Содержание

Введение **2**

1. Аналитический раздел **3**
1.1. Исходные данные 3
1.2. Рунге-Кутта 2ого порядка 4
1.3. Рунге-Кутта 4ого порядка 5

2. Технологический раздел **5**
2.1. Минимальные требования 5
2.2. Выбор языка и среды разработки 5
2.3. Листинг 5
2.4. Результаты работы программы 10

3. Вопросы и ответы **13**

1. Аналитический раздел

В данном разделе находится цель работы, описаны исходные данные, приведены методы Рунге-Кутта 1 и 2 порядка **Цель работы:** Получение навыков разработки алгоритмов решения задачи Коши при реализации моделей, построенных на системе ОДУ, с использование методов Рунге-Кутта 2-го и 4-го порядков точности

1.1. Исходные данные

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

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

2) Начальные условия: t = 0, I = I_0, U = U_0 где I $U0$ - ток и напряжение в конденсаторе

3) Сопротивление R_p рассчитать по формуле: $R_p = \frac{I_p}{2\pi R^2 \int_0^1 \sigma(T(z))zdz}$

4) Для функции $T(z)$ применить выражение $T(z) = T_0 + (T_w - T_0)z^m$ параметры T_0, m находятся интерполяцией из табл 1 при известном токе I. Коэффициент электропроводности $\sigma(T)$ зависит от T и рассчитывается интерполяцией из 2.

5) а) R = 0.35m
   б) l_e = 12cm
   в) $L_k = 187 * 10^{-6}$Гн
   г) $C_k = 268 * 10^{-6}$Ф
   д) Rk = 0.25Om
   е) U_co = 1400В
   ж) I_0 = 0.3А
   з) Tw = 2000k

Таблица 1. Таблица 1

T, K	, 1/Ом см
4000,00	0.031
5000,00	0.27
6000,00	43953,00
7000,00	43988,00
8000,00	12.0
9000,00	44093,00
10000,00	44011,00
11000,00	41.1
12000,00	54.1
13000,00	67.7
14000,00	81.5

Таблица 2. Таблица 2

I, A	To, K	m
0.5	6730,00	0.50
1,00	6790,00	0.55
5,00	7150,00	1,70
10,00	7270,00	3,00
50,00	8010,00	11,00
200,00	9185,00	32,00
400,00	10010,00	40,00
800,00	11140,00	41,00
1200,00	12010,00	39,00

1.2. Рунге-Кутта 2ого порядка

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

, где $\alpha = (0.5, 1)$

1.3. Рунге-Кутта 4ого порядка

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

$$U_{n_1} = U_n + \Delta t \frac{q_1 + 2q_2 + 2q_3 + q_4}{6}$$

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

2. Технологический раздел

В данном разделе указаны минимальные системные требования. Описан используемый язык и среда разработки. Приведен листинг программы.

2.1. Минимальные требования

Минимальные системные требования: PC с операционной системой Windows 7/8/10. Требуется устройство ввода: клавиатура, мышь. Устройства вывода: монитор

2.2. Выбор языка и среды разработки

Для достижения поставленной цели, мной был выбран язык Python версии 3.6 по причине удобства для математических вычислений, так же из за болеее лучшего уровня знания данного языка. Использовался sublime и интерпретатор для python.

2.3. Листинг

```
import math
import numpy
import matplotlib.pyplot as plt
from decimal import Decimal

R = 0.35
Tw = 2000.0
Ck = 268e-6
```

```
Lk = 187e-6
Rk = 0.25  #0.5 to= 200
Uc0 = 1400.0
I0 = 0.5  #0.5 to 3
le = 12.0
I4 = I0
Uc4 = Uc0
I2 = I0
Uc2 = Uc0
T0 = 0.0
m = 0.0

masI = [0.5, 1, 5, 10, 50, 200, 400, 800, 1200]
masT0 = [6730, 6790, 7150, 7270, 8010, 9185,
                 10010, 11140, 12010]
masm = [0.5, 0.55, 1.7, 3, 11, 32, 40, 41, 39]
masT = [4000, 5000, 6000, 7000, 8000, 9000,
                 10000, 11000, 12000, 13000, 14000]
masSigm = [0.031, 0.27, 2.05, 6.06, 12.0, 19.9,
                       29.6, 41.1, 54.1, 67.7, 81.5]


def f(xn, yn, zn, m_Rp):
        return -((Rk + m_Rp) * yn - zn)/Lk


        def phi(xn, yn, zn):
        return -yn/Ck


def trapez(start, end, function):
        step = 0.05#(end - start)/20
        result = 0
        while start <= end:
        left = function(start)
        start += step
        right = function(start)
        result += step*(left + right)/2
        return round(result, 4)


def integrate_polynom(coefs):
        result = [0]
        for i in range (len(coefs)):
```

```
        result.append(coefs[i - 1]/i)
        return result


def polynom_value(coefs, x):
        result = 0
        degree = 1
        for i in range(len(coefs)):
        if(coefs[i] != 0):
        result += coefs[i] * degree
        degree *= x
        return result

def interpolate(x, x_arr, y_arr):
        i = 0
        if (x <= x_arr[0]):
                x0 = x_arr[0]
                x1 = x_arr[1]
                y0 = y_arr[0]
                y1 = y_arr[1]
        elif (x > x_arr[-1]):
                x0 = x_arr[-2]
                x1 = x_arr[-1]
                y0 = y_arr[-2]
                y1 = y_arr[-1]
        else:
                while(x >= x_arr[i]):
                        x0 = x_arr[i - 1]
                        x1 = x_arr[i]
                        y0 = y_arr[i - 1]
                        y1 = y_arr[i]
                        i += 1
        return y0 + (y1 - y0) * x / (x1 - x0)


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


def sigma(T):
        return interpolate(T, masT, masSigm)


def rp(I):
```

```
        m = interpolate(I, masI, masm)
        T0 = interpolate(I, masI, masT0)
        integ = trapez(0, 1, lambda z: sigma(T(z, m, T0)) * z)
        return le / (2 * math.pi * R*R * integ), T0


def second_order(xn, yn, zn, hn, m_Rp):
        alpha = 1
        yn_1 = yn + hn * ((1 - alpha) * f(xn, yn, zn, m_Rp) + alpha \
        * f(xn + hn/(2*alpha),
        yn + hn/(2*alpha) * f(xn, yn, zn, m_Rp),
        zn + hn/(2*alpha) * phi(xn, yn, zn), m_Rp))
        zn_1 = zn + hn * ((1 - alpha) * phi(xn, yn, zn) + alpha \
        * phi(xn + hn/(2*alpha),
        yn + hn/(2*alpha) * f(xn, yn, zn, m_Rp),
        zn + hn/(2*alpha) * phi(xn, yn, zn)))
        return yn_1, zn_1


def fourth_order(xn, yn, zn, hn, m_Rp):
        k1 = hn * f(xn, yn, zn, m_Rp)
        q1 = hn * phi(xn, yn, zn)
        k2 = hn * f(xn + hn/2, yn + k1/2, zn + q1/2, m_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, m_Rp)
        q3 = hn * phi(xn + hn/2, yn + k2/2, zn + q2/2)

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

        yn_1 = yn + (k1 + 2*k2 + 2*k3 + k4)/6
        zn_1 = zn + (q1 + 2*q2 + 2*q3 + q4)/6
        return yn_1, zn_1


def do_plot(pltMasT, mas1, mas2, xlabel,
                        ylabel, name1, name2):
        plt.plot(pltMasT, mas1)
        plt.plot(pltMasT, mas2)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.legend((name1, name2))
        plt.grid(True)
```

```
        plt.show()


if __name__ == "__main__":
        pltMasT = []
        pltMasI4 = []
        pltMasU4 = []
        pltMasRp4 = []
        pltMasI2 = []
        pltMasU2 = []
        pltMasRp2 = []
        pltMasT0 = []
        pltMasT1 = []
        h = 1e-6
for t in numpy.arange(0, 20e-6, h):
        try:
        m_Rp4, T0 = rp(I4)
        m_Rp2, T1 = rp(I2)

        #if t > h:
        pltMasT0.append(T0)
        pltMasT1.append(T1)

        pltMasT.append(t)
        pltMasI4.append(I4)
        pltMasU4.append(Uc4)
        pltMasRp4.append(m_Rp4)
        pltMasI2.append(I2)
        pltMasU2.append(Uc2)
        pltMasRp2.append(m_Rp2)

        I4, Uc4 = fourth_order(t, I4, Uc4, h, m_Rp4)
        I2, Uc2 = second_order(t, I2, Uc2, h, m_Rp2)
        except Exception as e:
        print(e)
        except:
                break


do_plot(pltMasT, pltMasI4, pltMasI2, 't',
                'I', '4th order', '2nd order')
do_plot(pltMasT, pltMasU4, pltMasU2, 't',
                'Uc', '4th order','2nd order')
do_plot(pltMasT, pltMasRp4, pltMasRp2,'t',
```

```
                    'Rp', '4th order', '2nd order')

for i in range(len(pltMasI4)):
        pltMasI4[i] *= pltMasRp4[i]
        pltMasI2[i] *= pltMasRp2[i]

do_plot(pltMasT, pltMasI4, pltMasI2, 't',
                 'I * Rp', '4th order', '2nd order')
do_plot(pltMasT, pltMasT0, pltMasT1, 't',
                 'T0', '4th order', '2nd order')
```

2.4. Результаты работы программы

В данном подразделе приводятся результаты работы программы

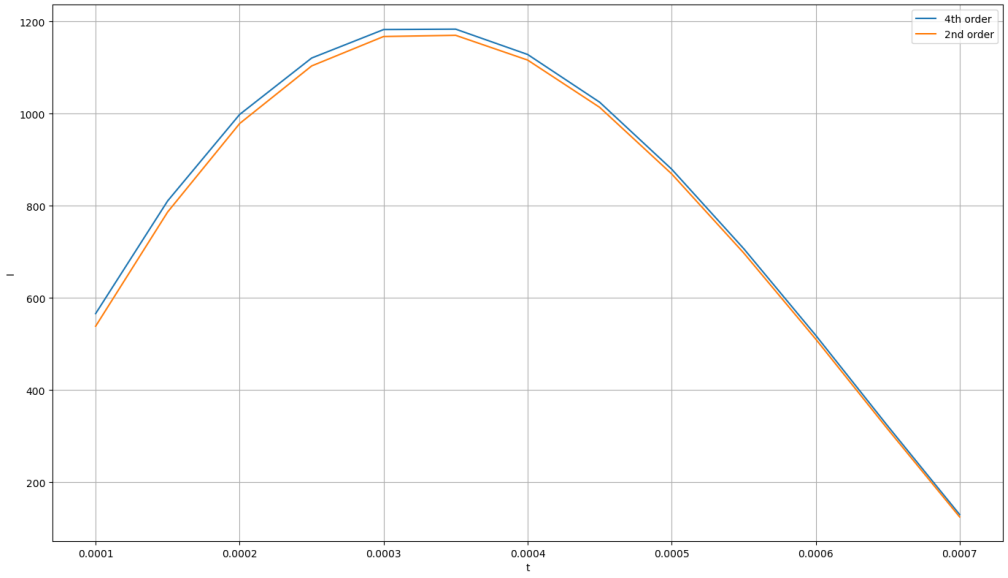

Рис. 1. I(t)

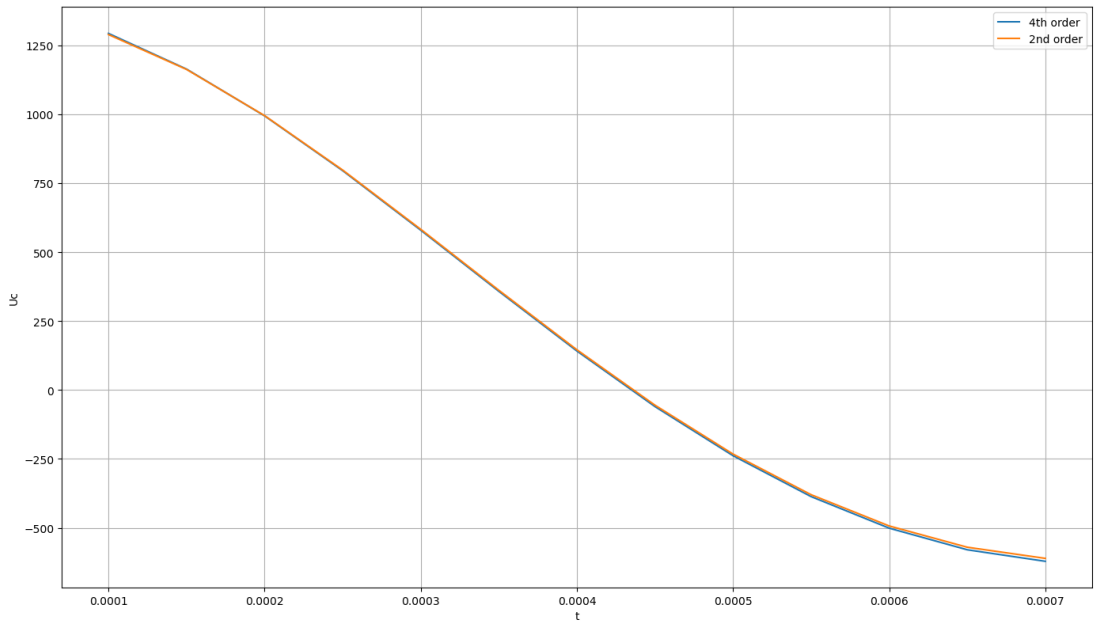

Рис. 2. U(t)

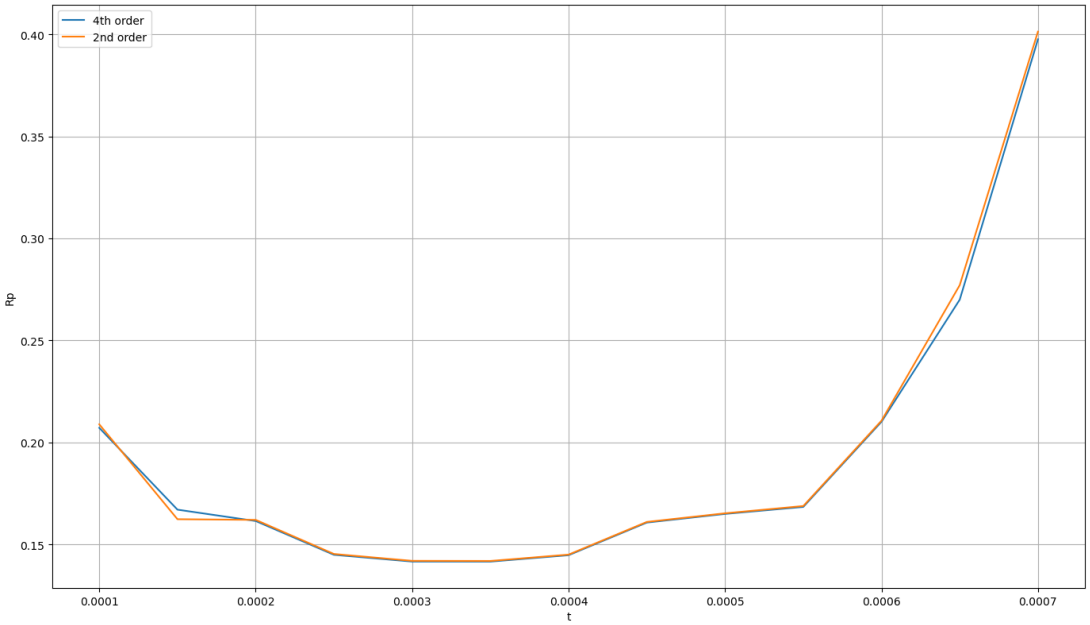

Рис. 3. Rp(t)

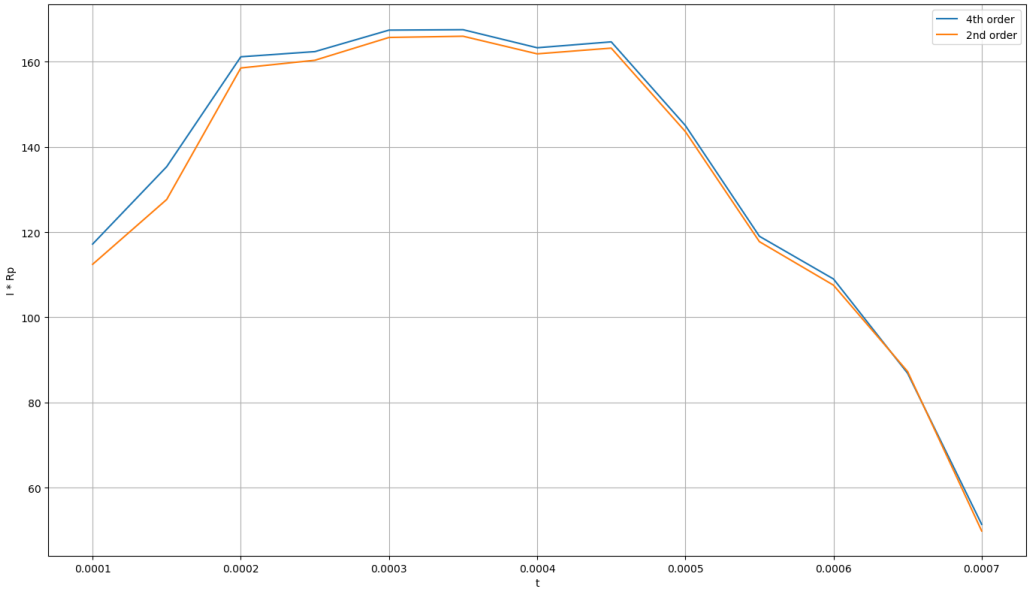

Рис. 4. I(t) * Rp(t)

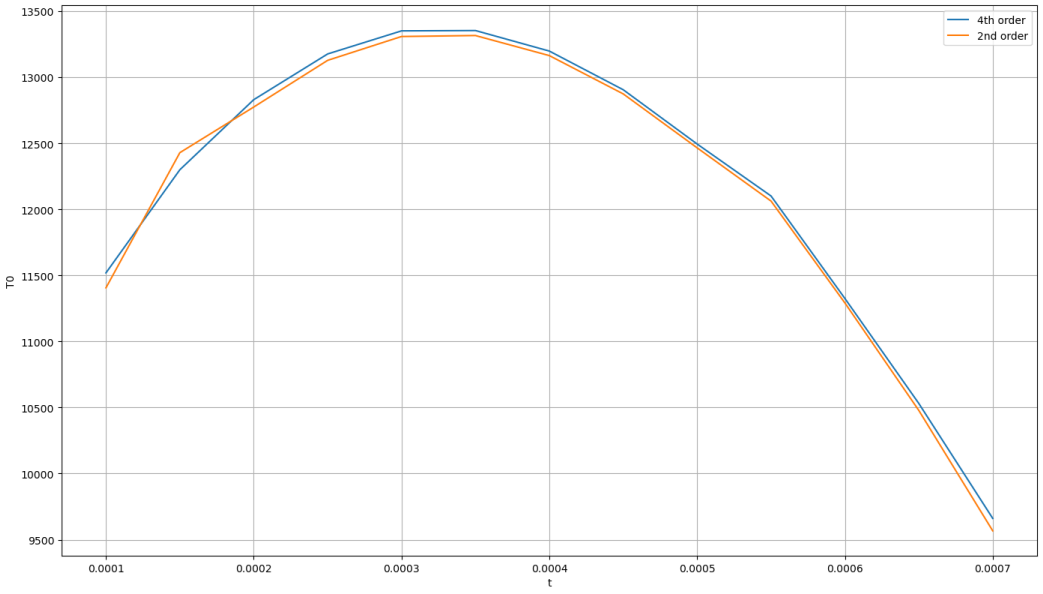

Рис. 5. T0(t)

3. Вопросы и ответы

1) а) Вопрос: Какие способы тестирования программы можно предложить?

   б) Ответ: 1) Приравнять друг к другу знание Rk и Rp к 0. Потери в контуре будут отсутствовать. На графики будем видеть незатухающую синусоиду 2) Сравнить результаты методов, с малым шагом

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

   б) Ответ:

$$\text{Возьмем систему: } \begin{cases} \frac{dl}{dt} & = \frac{U-(R_k+R_p(I))I}{L_k} \\ \frac{dU}{dy} & = -\frac{I}{C_k} \end{cases} \text{ Запишем выражение:} \quad (1)$$

   Запишем выражение:

$$I_{n+1} = I_n + \Delta t \frac{f(I_n, U_{n+1})}{2} \quad (2)$$

$$U_{n+1} = U_n + \Delta t \frac{g(l_n) + g(l_{n+1})}{2} \quad (3)$$

   Подставим выражение f и g:

$$l_{n+1} = l_n + \Delta t \frac{U_n - (R_k + R_p(I_n))I_n + U_{n+1} - R_k + Rp(l_{n+1}))I_{n+1}}{2L_k} \quad (4)$$

$$U_{n+1} = U_n - \Delta \frac{l_n + l_{n+1}}{2C_k} \quad (5)$$

   Подставив U_{n+1} из второго уравнения в первое, решим его относительно l_{n+1}

$$l_{n+1} = \frac{-2C_k R_p(I_n) I_n \Delta t + 4C_k I_k I_n - 2C_k I_n R_k \Delta t + 4C_k U_n \Delta t - I_n \Delta t^2}{den} \quad (6)$$

   Данное уравнение решается методом простых итераций $x^{(s)} = f(x^(s-1))$ Получив l_{n+1} подставим его значение в уравнение $U_{n+1} = U_n - \Delta t \frac{l_n + l_{n+1}}{2C_k}$

3) а) Вопрос: Из каких соображений проводится выбор того или иного метода, учитывая, что чем выше порядок точности метода, тем он более сложен?

   б) Ответ: Все исходит из необходимых нам условий времени и точности вычисления результата. Точные методы требуют больше аппаратного времени на вычисление, в то время как их "коллеги"вычисляются быстрее, но с меньшей точностью. Нужно искать золотую середину.