## Национальный исследовательский университет «МЭИ»

## (Московский Энергетический Институт)

**Кафедра математического и компьютерного моделирования**

## Численные методы

## Курсовая работа

**«ВЫНУЖДЕННЫЕ КОЛЕБАНИЯ ТОКА**

**В ЭЛЕКТРИЧЕСКОМ КОЛЕБАТЕЛЬНОМ КОНТУРЕ»**

*Выполнил:* Солонин Е. В. А-14-19

*Преподаватель:* Амосова О. А.

Вариант 8

2021 Москва

**Постановка задачи:**

С помощью стандартной замены переменных это уравнение приводится к системе обыкновенных дифференциальных уравнений. Рассматривается задача Коши для данной системы уравнений. Используя правило Рунге апостериорной оценки погрешности, составить адаптивную программу вычисления решения с заданной точностью. Решить численно полученную задачу, построить графики зависимости решения от времени.

Метод решения задачи: Адамса-Башфорта 3 порядка

**Теоретический материал:**

Метод Адамса-Башфорта 3 порядка:

Метод Рунге-Кутты 3 порядка:

**Приведение исходного уравнения к системе:**

Замена

Здесь

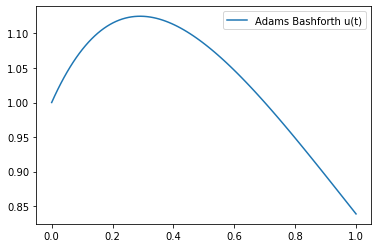
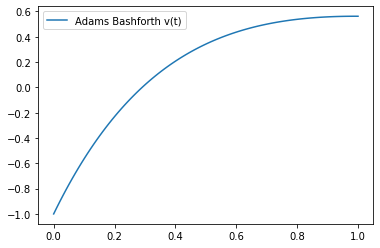
Метод Адамса-Башфорта 3 порядка – неявный, следовательно для разгона требуется вычислить еще 2 решения, помимо начального.

Будем использовать метод Рунге-Кутты 3 порядка

Построим тестовые примеры, чтобы убедиться в правильности работы формул методов.

**Тестовый пример:**

1. Решение:

****Результат работы программы для этого примера с шагом 0.01:

Это действительно графики решений u(t), v(t)

**Код программы:**

**import** numpy  
**import** matplotlib.pyplot **as** plt  
  
R\_ = **lambda** R0, k, i: R0 \* (1 + k \* i \*\* 2)  
  
*# задача Коши***def** f(y, t):  
 u, v = y  
 *# замена u = i  
 # замена v = di/dt* R = R\_(R0, k, u)  
 du = v  
 dv = (-R \* v - u / C + U \* w \* numpy.cos(w \* t)) / L  
 **return** numpy.array([du, dv])  
  
  
*# метод Рунге-Кутты 3 порядка***def** Runge\_Kutta\_3(y, t, h):  
 k1 = f(y, t) \* h  
 k2 = f(y + k1 / 2, t + h / 2) \* h  
 k3 = f(y - k1 + 2 \* k2, t + h) \* h  
 y = y + (k1 + 4 \* k2 + k3) / 6  
 t = t + h  
 **return** y  
  
  
*# метод Адамса-Башфорта 3 порядка***def** Adams\_Bashforth(y, t, i, h):  
 f\_1 = f(y[i - 1], t[i - 1])  
 f\_2 = f(y[i - 2], t[i - 2])  
 f\_3 = f(y[i - 3], t[i - 3])  
 **return** y[i - 1] + h / 12 \* (23 \* f\_1 - 16 \* f\_2 + 5 \* f\_3)  
  
  
*# расчет числа точек N для шага h*get\_n = **lambda** h: int(numpy.ceil((t\_n - t\_0) / h))  
  
*# максимальное значение погрешности по правилу Рунге***def** Runge(y\_data\_h, y\_data\_2h):  
 u\_h, v\_h = y\_data\_h  
 u\_2h, v\_2h = y\_data\_2h  
 r\_u\_data = [0]  
 r\_v\_data = [0]  
 **for** i **in** range(1, len(u\_2h)):  
 r\_u\_data.append(numpy.abs((u\_h[2 \* i] - u\_2h[i])) / (2 \*\* 3 - 1))  
  
 **for** i **in** range(1, len(v\_2h)):  
 r\_v\_data.append(numpy.abs((v\_h[2 \* i] - v\_2h[i])) / (2 \*\* 3 - 1))  
  
 **return** max(max(r\_u\_data), max(r\_v\_data))  
  
  
*# получить массив решений по методу Адамса-Башфорта 3 порядка  
# с помощью стартовых решений по методу Рунге-Кутты 3 порядка***def** get\_y\_data(h, y0):  
 n = get\_n(h)  
 t\_data = numpy.linspace(t\_0, t\_n, n)  
 y\_data = [y0]  
 y\_data.append(Runge\_Kutta\_3(y\_data[0], t\_0, h))  
 y\_data.append(Runge\_Kutta\_3(y\_data[1], t\_0 + h, h))  
  
 **for** i **in** range(3, n):  
 y\_data.append(Adams\_Bashforth(y\_data, t\_data, i, h))  
  
 y\_data = numpy.array(y\_data)  
 u, v = y\_data[:, 0], y\_data[:, 1]  
 **return** numpy.array([u, v])  
  
  
*# построение графика***def** show\_graph(h, y\_data, label):  
 fig, axs = plt.subplots()  
 n = get\_n(h)  
 t\_data = numpy.linspace(t\_0, t\_n, n)  
 plt.plot(t\_data, y\_data, label=**"Adams Bashforth "** + label)  
 plt.legend()  
 *# axs.spines['bottom'].set\_position('center')* plt.show()  
  
  
*# адаптивная процедура поиска оптимального шага  
# если точность не достигнута - уменьшаем шаг в 2 раза  
# если точность превышает заданную на порядок - увеличиваем шаг в 1.5 раза  
# иначе - нашли оптимальный шаг***def** get\_optimal\_h(h, y0):  
 y\_data\_h = get\_y\_data(h, y0)  
 y\_data\_2h = get\_y\_data(2 \* h, y0)  
 err = Runge(y\_data\_h, y\_data\_2h)  
 **if** err > epsilon:  
 **return** get\_optimal\_h(h / 2, y0)  
 **elif** err < epsilon / 10:  
 **return** get\_optimal\_h(1.5 \* h, y0)  
 **else**:  
 **return** h  
  
  
*# исходные данные*L = 50  
R0 = 3  
C = 0.047  
f\_ = 2 \* 10 \*\* 5  
k = 5 \* 10 \*\* 10  
w = 2 \* numpy.pi \* f\_  
U = 1  
  
epsilon = 10 \*\* -9  
t\_0 = 0  
t\_n = 1  
h = 0.01  
y0 = numpy.array([0, 0])  
  
  
y\_data\_h = get\_y\_data(h, y0)  
y\_data\_2h = get\_y\_data(2 \* h, y0)  
  
show\_graph(h, y\_data\_h[0], **"i(t)"**)  
  
*# h\_opt = get\_optimal\_h(h, y0)  
# print("Оптимальный шаг: ", h\_opt)*