## 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

**Введение:**

Инженеру часто приходится иметь дело с техническими системами и технологическими процессами, характеристики которых непрерывным образом меняются во времени. Соответствующие явления, как правило, подчиняются физическим законам, которые формулируются в виде дифференциальных уравнений. Одной из основных математических задач, которые приходится решать для таких уравнений, является задача Коши (начальная задача). Чаще всего к ней приходят тогда, когда известно начальное состояние физической величины системы в некоторый момент времени и требуется предсказать её поведение в момент времени .

**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

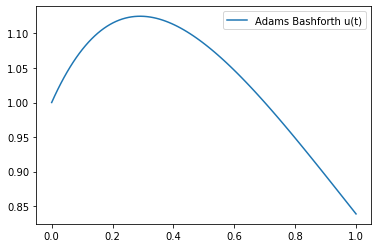
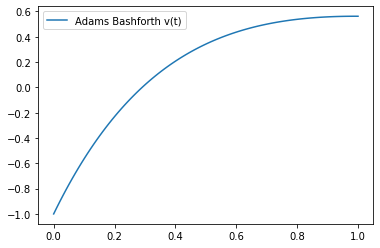
Здесь

Для разгона метода Адамса-Башфорта 3 порядка требуется вычислить еще 2 решения, помимо начального, так как метод является трехшаговым. Будем использовать для этой цели 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:

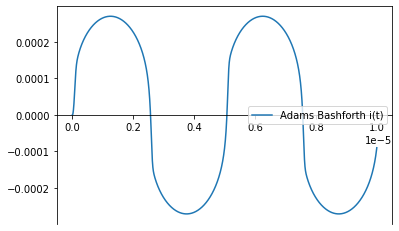
Это действительно графики решений u(t), v(t)

*Переходим к решению основной задачи.*

**Алгоритм поиска оптимального шага:**

1. Выбираем стартовый шаг, например, 0.01 от длины отрезка времени, охватывающего два периода колебаний
2. Вычисляем два вектора решений – две сеточные функции, одна с шагом h, другая с шагом h/2
3. По правилу Рунге находим максимальное значение погрешности в узле сетки с шагом h
4. Если заданная точность не достигнута, уменьшаем шаг вдвое и переходим к п.2
5. Если точность достигнута с избытком, увеличиваем шаг в 1.5 раза и переходим к п.2
6. Если точность достигнута, решение исходной задачи найдено

**Результаты:**

****График решения для двух периодов колебаний – зависимость тока от времени при 2ом варианте исходных данных.

Найденный оптимальный шаг и погрешность решения для каждого варианта исходных данных (решение вычислялось с точностью ):

1. Оптимальный шаг: 2.44e-12 Погрешность: 7.56e-09
2. Оптимальный шаг: 6.59e-12 Погрешность: 5.54e-09
3. Оптимальный шаг: 1.22e-11 Погрешность: 7.68e-10
4. Оптимальный шаг: 2.44e-11 Погрешность: 6.17e-09
5. Оптимальный шаг: 6.10e-13 Погрешность: 1.95e-10
6. Оптимальный шаг: 4.07e-11 Погрешность: 4.43e-09

**Вывод:**

Смоделирована зависимость силы тока в колебательном контуре от времени. Решение найдено с заданной точностью для всех вариантов исходных данных.

**Код программы:**

**import** numpy  
**import** matplotlib.pyplot **as** plt  
  
*# класс входных данных***class** InputData:  
 **def** \_\_init\_\_(self, R0, L, C, f, k):  
 self.L = L \* 10 \*\* -6  
 self.C = C \* 10 \*\* -6  
 self.R0 = R0  
 self.f = f  
 self.k = k  
 self.U = 1  
 self.w = 2 \* numpy.pi \* f  
 self.period = 1 / f  
  
 R = **lambda** self, i: self.R0 \* (1 + self.k \* i \*\* 2)  
  
  
*# задача Коши***def** f(y, t):  
 u, v = y  
 *# замена u = i  
 # замена v = di/dt* R = data.R(u)  
 du = v  
 dv = (-R \* v - u / data.C + data.U \* data.w \* numpy.cos(data.w \* t)) / data.L  
 **return** numpy.array([du, dv], dtype=numpy.float64)  
  
*# 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]  
 **for** i **in** range(1, len(u\_2h)):  
 r\_u\_data.append(numpy.abs((u\_h[2 \* i] - u\_2h[i])) / (2 \*\* 3 - 1))  
  
 **return** max(r\_u\_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, dtype=numpy.float64)  
 u, v = y\_data[:, 0], y\_data[:, 1]  
 **return** numpy.array([u, v], dtype=numpy.float64)  
  
*# 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)  
 print(  
 **"Error:"**,  
 **"{:.2e}"**.format(err),  
 **"\nStep: "**,  
 **"{:.2e}"**.format(h),  
 )  
 **if** err > EPSILON:  
 print(**"Decreasing...\n"**)  
 **return** get\_optimal\_h(h / 2, y0)  
 **elif** err < EPSILON / 10:  
 print(**"Increasing...\n"**)  
 **return** get\_optimal\_h(1.5 \* h, y0)**else**:  
 show\_graph(h, y\_data\_h[0], **"i(t)"**)  
 **return** h, err  
  
*# все варианты входных данных*allData = {  
 1: InputData(R0=2, L=1, C=0.001, f=10 \*\* 6, k=8 \* 10 \*\* 10), *# 60sec* 2: InputData(R0=3, L=5, C=0.01, f=5 \* 10 \*\* 6, k=2 \* 10 \*\* 14), *# 14sec* 3: InputData(R0=5, L=10, C=0.1, f=10 \*\* 5, k=10 \*\* 14), *# 120sec* 4: InputData(R0=3, L=50, C=0.047, f=2 \* 10 \*\* 5, k=5 \* 10 \*\* 10), *# 30sec* 5: InputData(R0=7.5, L=2, C=0.0047, f=2 \* 10 \*\* 6, k=4 \* 10 \*\* 15), *# 120sec* 6: InputData(R0=1, L=10, C=0.068, f=6 \* 10 \*\* 4, k=7 \* 10 \*\* 12), *# 60sec*}  
data = allData[2] *# выбираем вариант*EPSILON = 10 \*\* -8  
y0 = numpy.array([0, 0], dtype=numpy.float64)  
t\_0 = 0  
t\_n = data.period \* 2 *# охватываем два периода колебаний*h = (t\_n - t\_0) / 100  
  
h\_opt, err = get\_optimal\_h(h, y0)  
print(**"Оптимальный шаг: "**, **"{:.2e}"**.format(h\_opt))  
print(**"Погрешность: "**, **"{:.2e}"**.format(err))