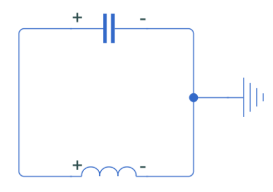
Дана электрическая схема, состоящая из идеального конденсатора ёмкостью *C* = 1 Ф, идеального индуктора индуктивностью *L* = 1 Гн и заземления. Пусть в начальный момент времени конденсатор заряжен до 1 В, а ток в цепи отсутствует.



Постройте соответствующую электрической схеме задачу Коши для системы обыкновенных дифференциальных уравнений и напишите программу для её решения любым численным методом на интервале *t* ∈ [0; 100].

Приведите графики изменения тока и напряжения конденсатора. Обоснуйте правильность полученного решения как с точки зрения физики процесса, так и с точки зрения математики.

# Решение.

Будем считать заряд конденсатора *q* положительным, если знаки заряда на обкладка совпадают с показанными на рисунке, а силу тока положительной, если ток направлен против часовой стрелки. Согласно закону Ома, в замкнутом контуре сумма падений напряжения равна сумме действующих ЭДС. На конденсаторе – напряжение , на катушке наводится самоиндукция , в точке заземление нет ни падения напряжения, ни ЭДС (т.е. заземление не влияет на колебательный контур).

.

Поскольку , имеем

или

с начальными условиями , .

Аналитическое решение этой задачи хорошо известно:

где – собственная частота колебаний.

Для численного решения исходный диффур второго порядка будем сводить к системе 2-х дифференциальных уравнений первого порядка через переобозначения , . Система примет вид

с соответствующим вектором начальных условий . Каждое уравнение системы , , переписывается через разностною схему.

В качестве основного решателя выбрана схема Рунге-Кутты 4-го порядка как самая популярная.

На одном шаге вектор-функция вычисляется 4 раза:

,

,

,

.

У метода 4-й порядок точности: суммарная ошибка имеет порядок .

Программировать будем на Python 3. Нам потребуется пакеты для работы с массивами numpy и рисования графиков matplotlib:

import numpy as np  
import matplotlib.pyplot as plt

Реализуем функцию runge\_kutta\_4(f, T, U0), которая возвращает сетку решений . Входные параметры функции:

* f – вектор-функция правых частей системы,
* T – вектор с разбиением по независимой переменной
* U0 – вектор начальных значений ,

Программный код метода-функции runge\_kutta\_4(f, T, U0) представлен ниже.

def runge\_kutta\_4(f, T, U0):  
 U = np.zeros((len(T), len(U0)))  
 U[0] = U0  
  
 h = T[1] - T[0]  
  
 for i in range(1, len(T)):  
 k1 = f(T[i - 1], U[i - 1])  
 k2 = f(T[i - 1] + h / 2, U[i - 1] + h \* k1 / 2)  
 k3 = f(T[i - 1] + h / 2, U[i - 1] + h \* k2 / 2)  
 k4 = f(T[i - 1] + h, U[i - 1] + h \* k3)  
  
 U[i] = U[i - 1] + h \* (k1 + 2 \* k2 + 2 \* k3 + k4) / 6  
  
 return U

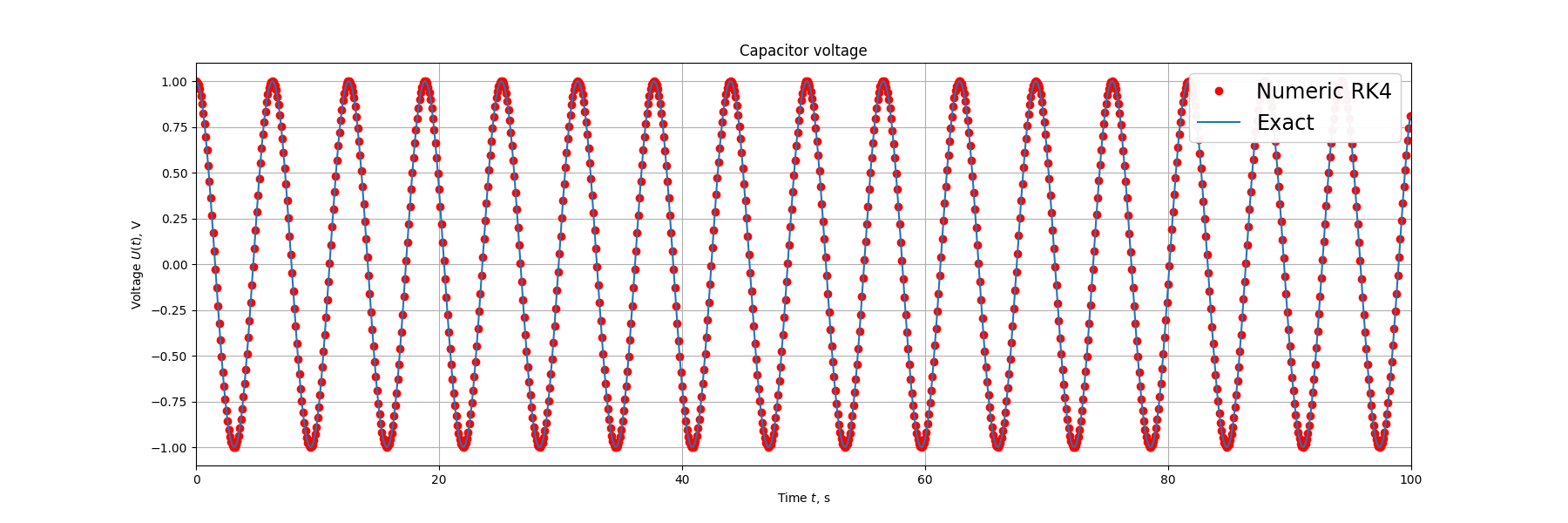
Массив функции правых частей системы :

def F(t, U):  
 """Right hand side function"""  
  
 f = np.zeros(U.shape)  
 f[0] = U[1]  
 f[1] = (-1) \* omega \*\* 2 \* U[0]  
  
 return f

Результаты работы программы (т.е. решение исходной задачи Коши) представлен на графиках.

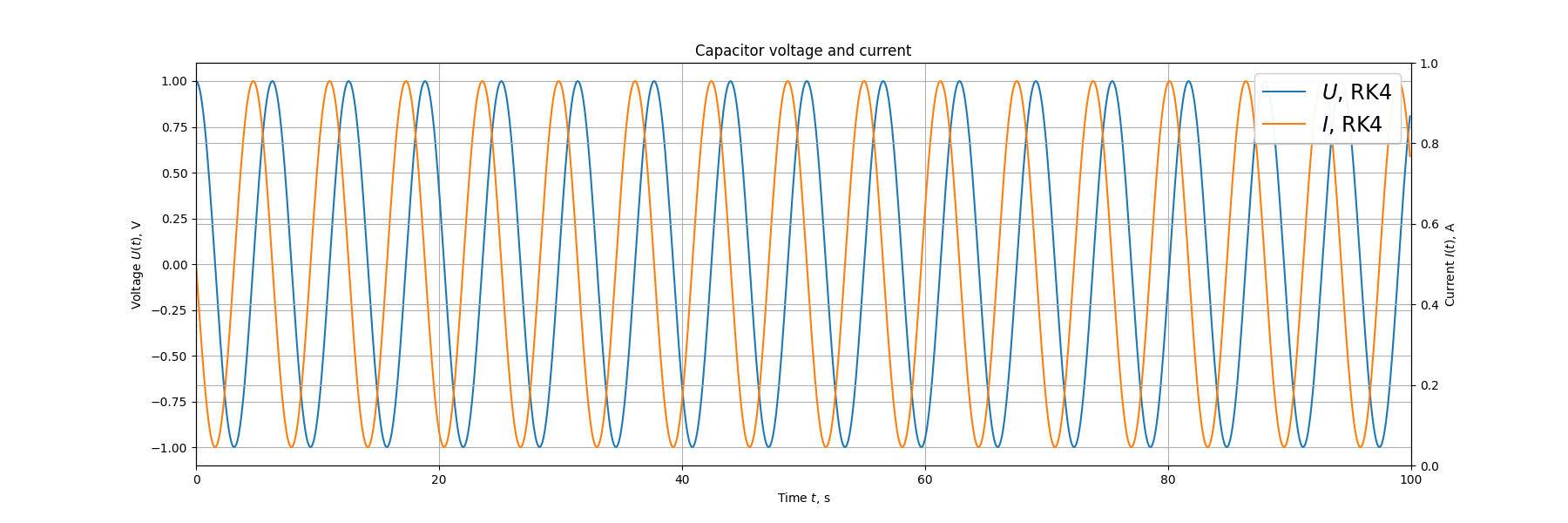
### 1. Напряжение на конденсаторе.

Это функция из системы или нулевой столбец ответного массива U[:,0]. Шаг по времени выбран c. Все точки численного решения нанесены на график в виде красных кружков. Визуально они прекрасно накладываются на точное аналитическое решение (сплошная синяя линия).

**

### 2. Ток в цепи.

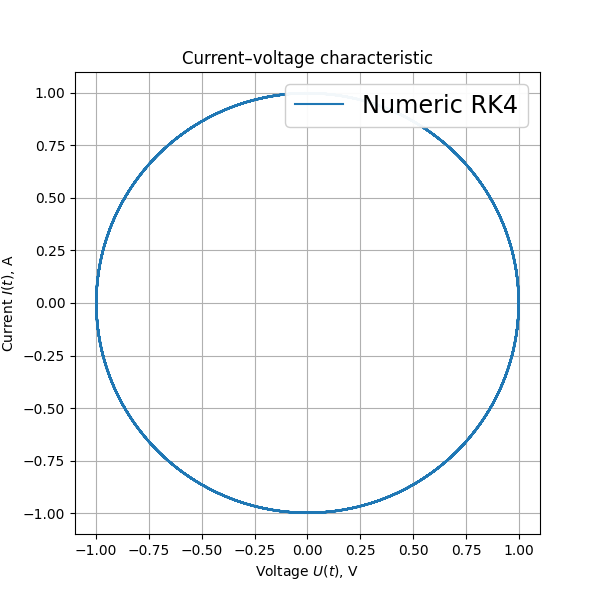
Это функция из системы или первый столбец ответного массива U[:,1] с размерным множителем . Зависимость силы тока в цепи представлена совместно с напряжением на конденсаторе, чтобы наглядно продемонстрировать смещение фаз между током и напряжением в четверть периода. Шаг по времени выбран c. Желтая линия показывает силу тока, полученную численно методом Рунге-Кутты 4 порядка; синяя линия показывает напряжение на конденсаторе, полученное в этом же численном решении.



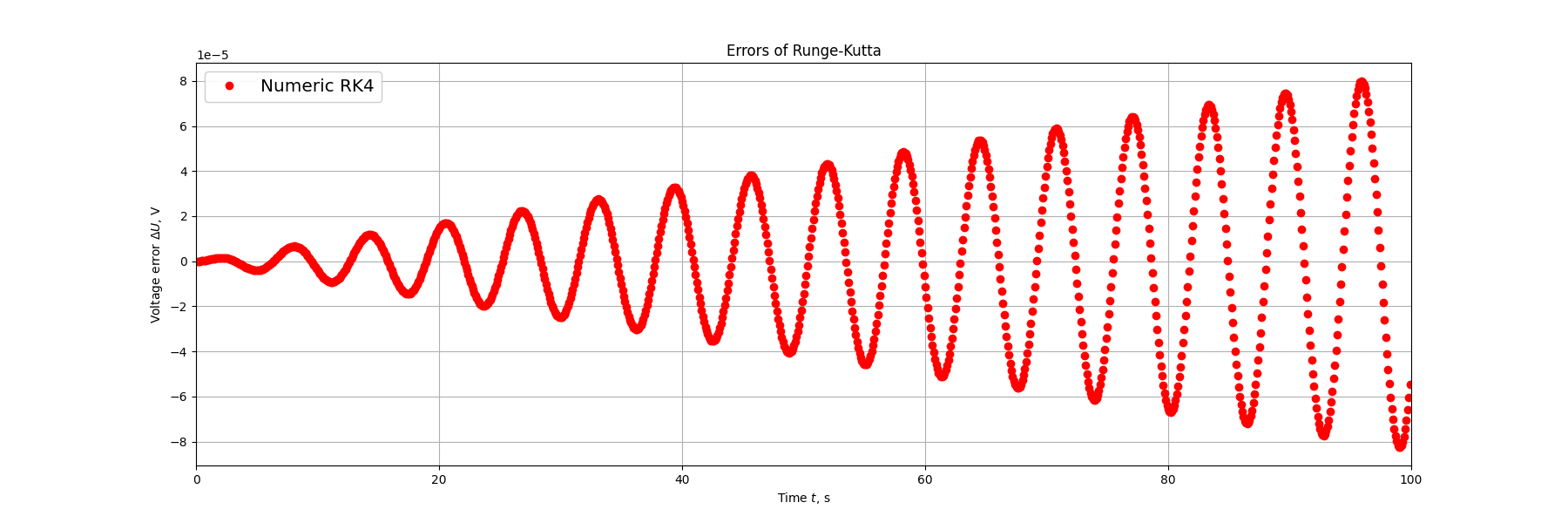
### 3. Ошибки.

Так как известно точное аналитическое решение, было бы интересно с ним свериться.

Если численное решение стабильно, то на совместном графике будет замкнутая кривая (эллипс), т.к. , . На графике численное решение, шаг c.



Тем не менее, если посмотреть на разницу численного решения и идеального, то видно, что ошибка накапливается и через интервалов станет сравнима с величиной самого решения. На графике ошибки численного решения для , шаг c.



В рамках данного интервала *t* ∈ [0; 100] ошибка согласуется с ожиданием . Если требуется повысить точность, то самым простым способом будет уменьшение шага разбиения, но это будет сильно влиять на сложность вычислений.

Функция runge\_kutta\_4(f, T, U0) находится в модуле models.py. Также в models реализована трехшаговая схема Адамса-Бешфорса adams\_ bashforth\_3(f, T, U0), но результаты здесь не привожу.

В файле main.py происходит инициализация задачи, вызов решателей и визуализация решения.