Написать программу численного решения задачи Коши для уравнения:

$$\frac{d^5y}{dx^5} + 15\frac{d^4y}{dx^4} + 90\frac{d^3y}{dx^3} + 270\frac{d^2y}{dx^2} + 405\frac{dy}{dx} + 243y = 0, \quad x \in [0,5],$$

$$y|_{x=0} = 0, \ \left.\frac{dy}{dx}\right|_{x=0} = 3, \ \left.\frac{d^2y}{dx^2}\right|_{x=0} = -9, \ \left.\frac{d^3y}{dx^3}\right|_{x=0} = -8, \ \left.\frac{d^4y}{dx^4}\right|_{x=0} = 0.$$

1. Реализовать какую-либо численную схему без использования готовых решателей.
2. Построить график решения.
3. Обосновать достоверность полученных результатов.

Решение.

С самого начала стоит знать, что задача имеет точное аналитическое решение

$$y(x) = -\frac{1}{12}e^{-3x} \cdot x \cdot (129x^3 + 16x^2 - 54x - 36).$$

Известно, что дифференциальное уравнение порядка n может быть сведено к системе n уравнений первого порядка путем переобозначений производных $y_0 = y$, $y_1 = y'$, $y_2 = y''$, … . В нашем случае:

$$\begin{cases} y_0' = y_1 \\ y_1' = y_2 \\ y_2' = y_3 \\ y_3' = y_4 \\ y_4' = -(15y_4 + 90y_3 + 270y_2 + 405y_1 + 243y_0) \end{cases}$$

с соответствующим вектором начальных условий $\{y_0, y_1, y_2, y_3, y_4\}_0 = \{0, 3, -9, -8, 0\}$.

Каждое уравнение системы $y_j' = f_j(x, y_0, \dots, y_4)$ переписывается через разностною схему и на каждом новом шаге разбиения получаем сразу решение всей системы для всех функций $y_j, j = \overline{0,4}$. Нас, естественно, будет интересовать только функция $y_0 = y$, являющаяся решением исходной задачи.

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

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{h}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4).$$

На одном шаге вектор-функция вычисляется 4 раза:

$\boldsymbol{k}_1 = \boldsymbol{f}(x_n, \boldsymbol{y}_n)$,

$\boldsymbol{k}_2 = \boldsymbol{f}\left(x_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{k}_1\right)$,

$\boldsymbol{k}_3 = \boldsymbol{f}\left(x_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{k}_2\right)$,

$\boldsymbol{k}_4 = \boldsymbol{f}(x_n + h, \boldsymbol{y}_n + h\boldsymbol{k}_3)$.

У метода 4-й порядок точности: суммарная ошибка имеет порядок $O(h^4)$.

Программировать будем на Python 3.8. Нам потребуется пакеты для работы с массивами numpy и рисования графиков matplotlib:

```python
import numpy as np
import matplotlib.pyplot as plt
```

Реализуем функцию runge_kutta(x0, y0, xN, h), которая возвращает сетку решений $\{x_i, \boldsymbol{y}_i\}$. Входные параметры функции:

- x0 – начальная точка по x,
- y0 – вектор начальных значений $\{y_0, y_1, y_2, y_3, y_4\}_0$,
- xN – конечная точка по x,
- h – шаг сетки.

Пример вызова: начальные условия и интервал как в исходной задаче, шаг 0,01.

```python
X, Y = runge_kutta(x0=0, y0=[0, 3, -9, -8, 0], xN=5, h=0.01)
```

В результате X будет одномерный массив из x_i, а Y будет двумерный массив (в строке с номером i хранятся компоненты $\boldsymbol{y}_i = \{y_0, \dots, y_4\}_i$, которые соответствует i-й прогонке цикла по разбиению).

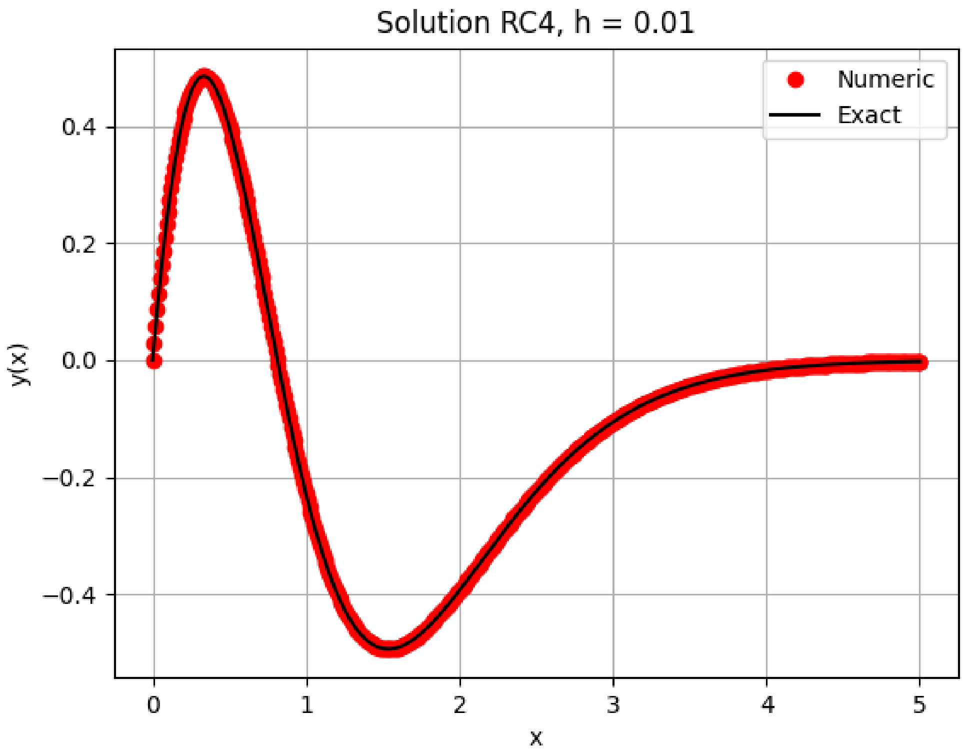
Программный код метода-функции runge_kutta(x0, y0, xN, h) представлен ниже.

```python
def runge_kutta(x0, y0, xN, h):
    """…Documentation…"""
    X = []
    Y = []
    X.append(x0)
    Y.append(y0)

    xi, yi = x0, y0
    # fill the grid by integration
    while xi < xN:

        h = min(h, xN-xi)

        k1 = rhs(xi, yi)
        k2 = rhs(xi + h/2, yi + k1/2)
        k3 = rhs(xi + h/2, yi + k2/2)
        k4 = rhs(xi + h, yi + k3)

        yi = yi + h * (k1  + 2*k2 + 2*k3 + k4) / 6
        xi = xi + h

        X.append(xi)
        Y.append(yi)

    return np.array(X), np.array(Y)
```

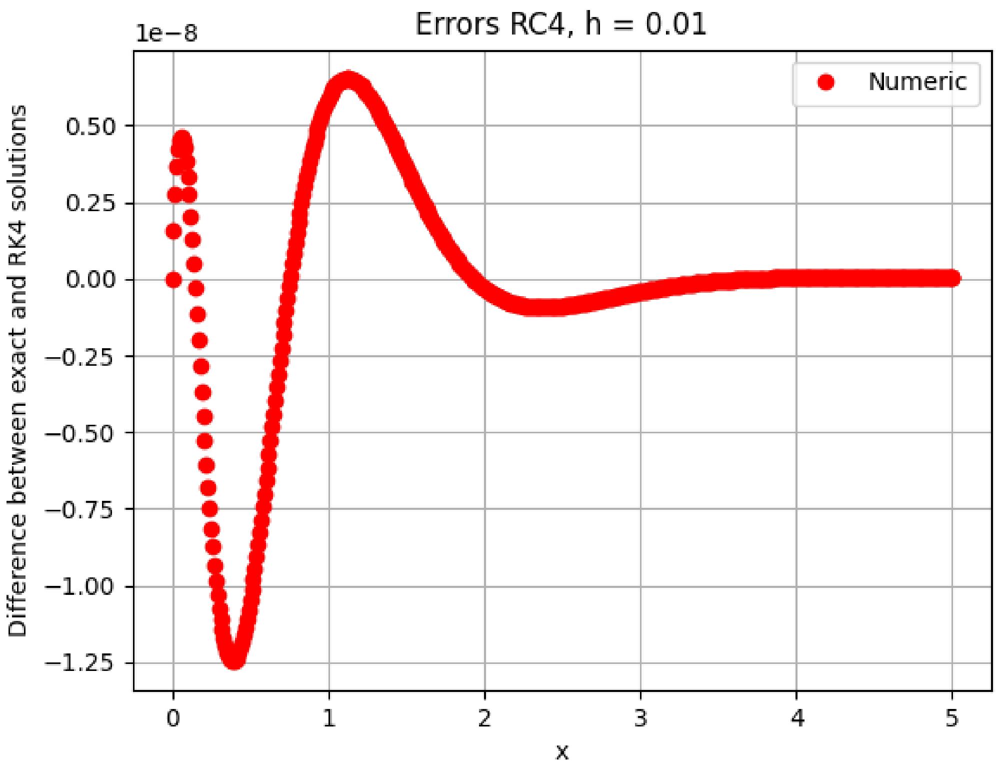
Внутри происходит вызов функции rhs(x, y) – аббревиатура от «right hand side», которая умеет считать правые части $f_j(x, y_0, \dots, y_4)$ исходной системы.

```python
def rhs(x, y):
    """Right hand side function"""
    f = np.array([y[1], y[2], y[3], y[4],
                  (-1)*(15*y[4] + 90*y[3] + 270*y[2] + 405*y[1] + 243*y[0])
                  ])
    return f
```

Главный результат работы программы (т.е. решение исходной задачи Коши) представлен на графике. Шаг выбран $h = 0{,}01$. Все точки численного решения нанесены на график в виде красных кружков. Визуально они прекрасно накладываются на точное аналитическое решение (сплошная черная линия).



Найдем разницу между точным и численным решением, т.е. погрешность работы алгоритма. Из графика видно, что эта ошибка порядка $O(10^{-8})$, что соответствует ожиданиям для данного шага $h = 0{,}01$ и методики (Рунге-Кутты 4-го порядка).



Так как на практике аналитическое решение может отсутствовать, было бы полезно иметь еще какой-либо ориентир для сравнения. Без сильной модификации метода можно сравнивать схему Рунге-Кутты 4-го порядка со схемой Рунге-Кутты 1-го порядка (схемой Эйлера).

$\boldsymbol{y}_{n+1}^{\text{РК}} = \boldsymbol{y}_n + \frac{h}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4),$

$\boldsymbol{y}_{n+1}^{\text{Э}} = \boldsymbol{y}_n + h\boldsymbol{k}_1.$

Оценка погрешности:

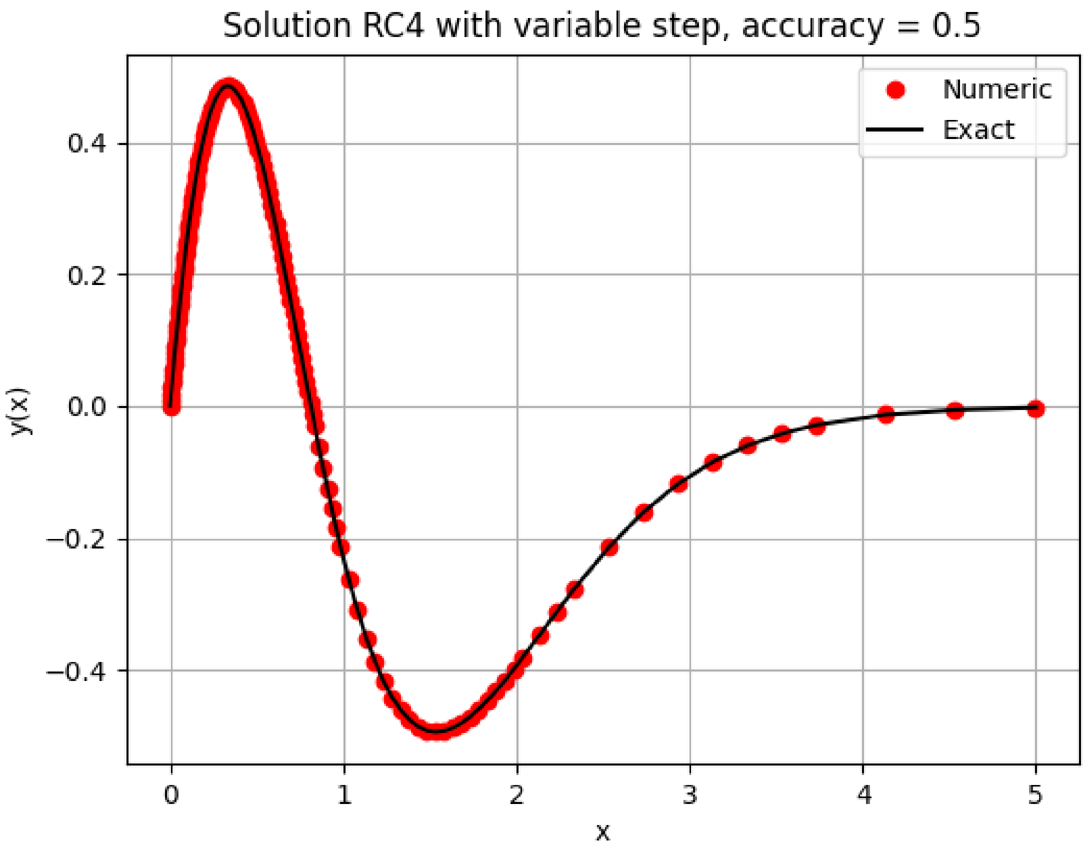
$\boldsymbol{E}_{n+1} = \boldsymbol{y}_{n+1}^{\text{РК}} - \boldsymbol{y}_{n+1}^{\text{Э}} = \frac{h}{6}(-5\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4).$

Если погрешность больше допуска $\|\boldsymbol{E}_{n+1}\| > t$, то инкремент нужно пересчитать с уменьшенным шагом $h/2$. Если погрешность достаточно маленькая $\|\boldsymbol{E}_{n+1}\| < t/2^4$, то следующий шаг берем вдвое большим $2h$. Модифицированная функция носит название runge_kutta_tolerance(x0, y0, xN, h, t) и имеет дополнительный параметр t, отвечающий за точность.

Пример работы модифицированного метода

```
X, Y = runge_kutta_tolerance(x0, y0, xN, h=0.1, t=0.5)
```

Допуск t=0.5 специально был взят широким, чтобы увидеть переменный шаг.



Функции runge_kutta (x0, y0, xN, h), runge_kutta_tolerance(x0, y0, xN, h, t), а также rhs(x, y) находятся в модуле models.py. В файле main.py происходит вызов этих функций и визуализация решения.

Репозиторий:

https://github.com/alexande-popov/Runge_Kutta.git