МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №6

по дисциплине

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №8

Выполнила:
Конаныхина Антонина
P3215
Преподаватель:
Малышева Татьяна
Алексеевна

Санкт-Петербург, 2022

Цель работы:

Решить задачу Коши численными методами.
Для исследования использовать:

- Одношаговые методы;
- Многошаговые методы.

Задание:

Программная реализация задачи:
1. Исходные данные: ОДУ вида $y' = f(x, y)$, начальные условия $y(x_0)$, интервал дифференцирования $[a, b]$, шаг h, точность ε.
2. Составить таблицу приближенных значений интеграла дифференциального уравнения, удовлетворяющего начальным условиям. Для оценки точности использовать правило Рунге.
3. Построить графики точного решения и полученного численного решения (разными цветами).

Рабочие формулы используемых методов

Метод Рунге-Кутта:

$$\boldsymbol{y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)}$$

где

$$\boldsymbol{k_1 = h \cdot f(x_i, y_i)}$$
$$\boldsymbol{k_2 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_1}{2})}$$
$$\boldsymbol{k_3 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_2}{2})}$$
$$\boldsymbol{k_4 = h \cdot f(x_i + h, y_i + k_3)}$$

Метод Адамса:

$$\boldsymbol{y_{i+1} = y_i + hf_i + \frac{h^2}{2}\Delta f_i + \frac{5h^3}{12}\Delta^2 f_i + \frac{3h^4}{8}\Delta^3 f_i}$$

где

$$\Delta f_i = f_i - f_{i-1}$$
$$\Delta^2 f_i = f_i - 2f_{i-1} + f_{i-2}$$
$$\Delta^3 f_i = f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}$$

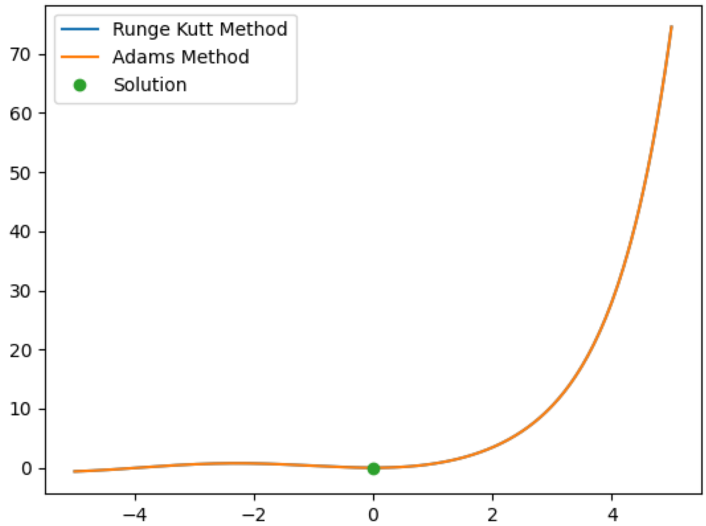
Листинг программы

```
def rungeKuttNext(func, x, y, h):
    k1 = h * func(x, y)
    k2 = h * func(x + h / 2, y + k1 / 2)
    k3 = h * func(x + h / 2, y + k2 / 2)
    k4 = h * func(x + h, y + k3)
    return y + (1 / 6) * (k1 + 2 * k2 + 2 * k3 + k4)


def rungeKutt(func, x_0, y_0, h, n):
    y_prev = y_0
    y_current = 0
    answer_x = [x_0]
    answer_y = [y_0]
    for i in range(n):
        x_prev = x_0 + i * h
        y_current = rungeKuttNext(func, x_prev, y_prev, h)
        answer_y.append(y_current)
        answer_x.append(x_0 + (i + 1)*h)
        y_prev = y_current
    return list(zip(answer_x, answer_y))


def rungeKuttMethod(func, a, b, x_0, y_0, h, e):
    R = e * 100
    real_h = h

    ans = rungeKutt(func, x_0, y_0, -real_h, int((x_0 - a) / real_h))
    ans = ans + rungeKutt(func, x_0, y_0, real_h, int((b - x_0) / real_h))
    y_old = ans[-1][1]
    count = 0
    while R > e:
        real_h /= 2
        ans = rungeKutt(func, x_0, y_0, -real_h, int((x_0 - a) / real_h))
        ans = ans + rungeKutt(func, x_0, y_0, real_h, int((b - x_0) /
real_h))
        y_now = ans[-1][1]
        R = abs((y_old - y_now))/(2**4 - 1)
        y_old = y_now

        count += 1
        if count > 10:
            break

    def sorter(e):
        return e[0]

    ans.sort(key=sorter)

    return ans


def adams(func, x_0, y_0, h, n):
    answer_x = []
    answer_y = []
    answer_x.append(x_0)
    answer_y.append(y_0)
    y_1 = rungeKuttNext(func, x_0, y_0, h)
    func_prev_1 = func(x_0 + h, y_1)
    answer_x.append(x_0 + h)
```

```
    answer_y.append(y_1)
    y_2 = rungeKuttNext(func, x_0 + h, y_1, h)
    func_prev_2 = func(x_0 + 2 * h, y_2)
    answer_x.append(x_0 + 2*h)
    answer_y.append(y_2)
    y_3 = rungeKuttNext(func, x_0 + 2 * h, y_2, h)
    func_prev_3 = func(x_0 + 3 * h, y_3)
    answer_x.append(x_0 + 3*h)
    answer_y.append(y_3)
    y_prev = rungeKuttNext(func, x_0 + 3 * h, y_3, h)
    func_prev_4 = func(x_0 + 4 * h, y_prev)
    answer_x.append(x_0 + 4*h)
    answer_y.append(y_prev)
    for i in range(4, n):
        delta_f_i = func_prev_4 - func_prev_3
        delta2_f_i = func_prev_4 - 2 * func_prev_3 + func_prev_2
        delta3_f_i = func_prev_4 - 3 * func_prev_3 + 3 * func_prev_2 -
func_prev_1
        x_next = x_0 + i * h
        x_prev = x_0 + (i - 1) * h
        y_next = y_prev + h * func(x_prev, y_prev) + ((h ** 2) / 2) *
delta_f_i + ((5 * (h ** 3)) / 12) * delta2_f_i + \
                 ((3 * (h ** 4)) / 8) * delta3_f_i

        answer_x.append(x_next)
        answer_y.append(y_next)
        func_prev_1 = func_prev_2
        func_prev_2 = func_prev_3
        func_prev_3 = func_prev_4
        func_prev_4 = func(x_next, y_next)
        y_prev = y_next
    return list(zip(answer_x, answer_y))

def adamsMethod(func, a, b, x_0, y_0, h, e):
    R = e * 100
    real_h = h

    ans = adams(func, x_0, y_0, -real_h, int((x_0 - a) / real_h) + 1)
    ans = ans + adams(func, x_0, y_0, real_h, int((b - x_0) / real_h) + 1)
    y_old = ans[-1][1]
    count = 0
    while R > e:
        real_h /= 2
        ans = adams(func, x_0, y_0, -real_h, int((x_0 - a) / real_h) + 1)
        ans = ans + adams(func, x_0, y_0, real_h, int((b - x_0) / real_h) +
1)
        y_now = ans[-1][1]
        R = abs((y_old - y_now)) / (2 ** 4 - 1)
        y_old = y_now

        count+=1
        if count > 10:
            break

    def sorter(e):
        return e[0]

    ans.sort(key=sorter)

    return ans
```

Результаты выполнения программы:

```
1 : y' =  sin(x) + y
2 : y' =  cos(x + y)
3 : y' =  x^3 - 2y
4 : y' =  (x - y)^2
Выберите функцию: 1
Введите x0: 0
Введите y0: 0
Введите правую границу (a): -5
Введите левую границу (b): 5
Введите h: 0.1
Введите e: 0.001
```



Вывод:

В результате выполнения данной лабораторной работы я познакомилась с различными методами решения задачи Коши и реализовала их на языке программирования Python. Оба метода имеют одинаковую точность и на небольших диапазонах имеют одинаковые значения. На больших значениях начинают показывать разное значение. Опыт показывает, что метод Рунге-Кутта точнее.