Университет ИТМО

Факультет программной инженерии и компьютерной техники

Вычислительная математика.
Лабораторная работа №6.
Численное решение обыкновенных дифференциальных уравнений

Группа:	P32131
Студент:	Смирнов Виктор Игоревич
Вариант:	16

2023

Ключевые слова

Дифференциальные Уравнения, Численные Методы.

Содержание

1 Цель работы

Цель работы - решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

2 Используемые методы

2.1 Метод Эйлера

Пусть дана Задача Коши для уравнения первого порядка.

$$\frac{dy}{dx} = f(x, y),\ y(x_0) = y_0 \tag{1}$$

Хотим решить данной уравнение на интервале $[x_0, x_n]$, который разобъем на интервалы с шагом h.

Тогда рабочая формула форумула метода будет:

$$y_i = y_{i-1} + (x_i - x_{i-1}) f(x_{i-1}, y_{i-1}) \tag{2}$$

Факты:

1. Прост в реализации
2. Имеет погрешность $O(h)$

Также можно улучшить точность алгоритма, рассмотрев частный случай неявного метода Рунге — Кутты первого порядка:

$$y'_{n+1} = y_n + hf(x_n, y_n) - \text{Прогноз} \tag{3}$$

$$y_{n+1} = y_n + h\frac{f(x_n, y_n) + f(x_{n+1}, y'_{n+1})}{2} - \text{Коррекция} \tag{4}$$

```
def solve(input: Input, corrected: bool = True) -> Output:
    '''
                                                          -          .

    https://ru.wikipedia.org/wiki/           _
    https://ru.wikipedia.org/wiki/                ___
    '''

    f, (x_0, y_0), x_n, h, eps = input.validated

    p = 2 if corrected else 1  #

    def next(h: float) -> Tuple[float, float]:
        x_i = x[-1] + h
        y_i = y[-1] + (x_i - x[-1]) * f(x[-1], y[-1])
        return (x_i, y_i)

    def correct(x_i: float, y_i: float) -> Tuple[float, float]:
        y_i = y[-1] + (x_i - x[-1]) * (
            f(x[-1], y[-1]) + f(x_i, y_i)
        ) / 2
        return (x_i, y_i)

    x = [x_0]
    y = [y_0]
```

```python
    while abs(x[-1] - x_n) >= h / 2:
        x_i, y_i = next(h)
        x_i_half, y_i_half = next(h / 2)

        if corrected:
            x_i, y_i = correct(x_i, y_i)
            x_i_half, y_i_half = correct(x_i_half, y_i_half)

        if not runge_rule(y_i, y_i_half, p, eps):
            h /= 2
            continue

        x += [x_i]
        y += [y_i]

    return Output(
        list(map(lambda t: Point(*t), zip(x, y))),  # type: ignore
        Table(
            ['i', 'x', 'y'],
            list(zip(range(len(x)), x, y)),
        ),
    )
```

Листинг 1: Реализация метода Эйлера на Python

2.2 Метод Рунге — Кутты 4ого порядка

Задача аналогичная, метод другой.

Рабочая формула:

$$k_1 = f(x_n, y_n) \tag{5}$$

$$k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \tag{6}$$

$$k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \tag{7}$$

$$k_3 = f(x_n + h, y_n + hk_3) \tag{8}$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{9}$$

Факты:

1. Гораздо сложнее Эйлера
2. Зато гарантирует погрешность $O(h^4)$

```python
def solve(input: Input, runge=True) -> Output:
    '''
    https://ru.wikipedia.org/wiki/                    ___
    '''

    f, (x_0, y_0), x_n, h, eps = input.validated

    p = 4  #

    def k_1(x: float, y: float) -> float:
        return f(x, y)

    def k_2(x: float, y: float) -> float:
        return f(x + h / 2, y + h * k_1(x, y) / 2)

    def k_3(x: float, y: float) -> float:
        return f(x + h / 2, y + h * k_2(x, y) / 2)

    def k_4(x: float, y: float) -> float:
        return f(x + h, y + h * k_3(x, y))

    def next(h: float) -> Tuple[float, float]:
        return (
```

```
            x[-1] + h,
            y[-1] + h / 6
            * (k1[-1] + 2 * k2[-1] + 2 * k3[-1] + k4[-1])
        )

    x, y = [x_0], [y_0]
    k1, k2, k3, k4 = [], [], [], []
    while abs(x[-1] - x_n) >= h / 2:
        k1 += [k_1(x[-1], y[-1])]
        k2 += [k_2(x[-1], y[-1])]
        k3 += [k_3(x[-1], y[-1])]
        k4 += [k_4(x[-1], y[-1])]

        x_i, y_i = next(h)

        if runge:
            _, y_i_half = next(h / 2)
            if not runge_rule(y_i, y_i_half, p, eps):
                h /= 2
                continue

        y += [y_i]
        x += [x_i]

    return Output(
        list(map(lambda t: Point(*t), zip(x, y))),  # type: ignore
        Table(
            ['i', 'x', 'y', 'k1', 'k2', 'k3', 'k4'],
            list(zip(range(len(x)), x, y, k1, k2, k3, k4))
        )
    )
```

Листинг 2: Реализация метода Рунге-Кутты на Python

2.3 Метод Милна

Данный метод уже является многошаговым, то есть требует несколько точек для старта. Используется принцип прогноза и коррекции.

$$y_i^{predict} = y_{i-4} + \frac{4h}{3}(2f_{i-3} - f_{i-2} + 2f_{i-1}), f_i = f(x_i, y_i) \tag{10}$$

$$f_i^{predict} = f(x_i, y_i^{predict}) \tag{11}$$

$$y_i^{correct} = y_{i-2} + \frac{h}{3}(f_{i-2} + 4f_{i-1} + f_i^{predict}) \tag{12}$$

```
def solve(input: Input, one_step: OneStepMethod) -> Output:
    '''

                                                                       .


                 .

                                          ,
     (                ,
                                 -           ).
    '''

    f, (x_0, y_0), x_n, h, eps = input.validated

    p = 4 #

    # Compute first 3 points
    (points, _) = one_step(Input(f, Point(x_0, y_0), x_0 + 3 * h, h, eps))
```

```python

    def next(h: float) -> Tuple[float, float, float, float]:
        x_i = x[-1] + h
        y_i_predict = y[-4] + 4 / 3 * h * (
            + 2 * f(x[-3], y[-3])
            - f(x[-2], y[-2])
            + 2 * f(x[-1], y[-1])
        )
        f_i_predict = f(x_i, y_i_predict)
        y_i_correct = y[-2] + h / 3 * (
            + f(x[-2], y[-2])
            + 4 * f(x[-1], y[-1])
            + f_i_predict
        )
        return (x_i, y_i_predict, f_i_predict, y_i_correct)

    x = list(map(lambda p: p.x, points))
    y = y_predict = list(map(lambda p: p.y, points))
    y_correct = list(map(lambda p: p.y, points))
    f_predict = [f(x, y) for x, y in zip(x, y)]
    while abs(x[-1] - x_n) >= h / 2:
        x_i, y_i_predict, f_i_predict, y_i_correct = next(h)
        _, _, _, y_i_correct_half = next(h / 2)

        if not runge_rule(y_i_correct, y_i_correct_half, p, eps):
            h /= 2
            continue

        y_predict += [y_i_predict]
        y_correct += [y_i_correct]
        f_predict += [f_i_predict]
        x += [x_i]

    return Output(
        list(map(lambda t: Point(*t), zip(x, y))),  # type: ignore
        Table(
            ['i', 'x', 'y_c', 'y_p', 'f_p'],
            list(zip(
                range(len(x)),
                x,
                y,
                y_predict,
                f_predict,
            )),
        ),
    )
```

Листинг 3: Реализация метода Милне на Python

2.4 Правило Рунге

Контроль точности осуществляется правилом Рунге.

$$R = \frac{|y_i^h - y_i^{h/2}|}{2^p - 1} \leq \varepsilon \tag{13}$$

3 Примеры работы программы

```
=== Welcome to differential eqsolver ===

Here you can solve differential equations using:
- Basic Euler
- Corrected Euler
- Runge-Kutta 4
- Milne

Choice a one of first-order equations:
[0]: dy/dx = y + (1 + x) * y ^ 2
[1]: dy/dx = -2 * y
[2]: dy/dx = 2 * x * exp(x ** 2) / (exp(x ** 2) + 1)
```

```
Enter a number of equation i in [0..2]: 0
Taken: dy/dx = y + (1 + x) * y ^ 2
Enter boundaties:
Enter x_0: 1
Enter y_0: -1
Enter x_n: 1.5
Enter h: 0.1
Enter eps: 0.000001

Input parameters:
> x_0 = 1.0
> y_0 = -1.0
> x_n = 1.5
> h   = 0.1
> eps = 1e-06

Enjoy results!

=== Report of Basic Euler ===

Result of Basic Euler is -0.6666664538724102
Total iterations: 327681
Stop at: 1.5000000000291038

=== Report of Corrected Euler ===

Result of Corrected Euler is -0.6666666666710818
Total iterations: 163841
Stop at: 1.499999999992724

=== Report of Runge-Kutta 4 ===

Result of Runge-Kutta 4 is -0.6666666666664894
Total iterations: 20481
Stop at: 1.500000000001819

=== Report of Milne ===

Result of Milne is -0.6666666666664923
Total iterations: 20481
Stop at: 1.500000000001819
```

Листинг 4: Результаты работы программы 0

```
$ bash ci/run.sh < res/1.txt
=== Welcome to differential eqsolver ===

Here you can solve differential equations using:
- Basic Euler
- Corrected Euler
- Runge-Kutta 4
- Milne

Choice a one of first-order equations:
[0]: dy/dx = y + (1 + x) * y ^ 2
[1]: dy/dx = -2 * y
[2]: dy/dx = 2 * x * exp(x ** 2) / (exp(x ** 2) + 1)
Enter a number of equation i in [0..2]: Taken: dy/dx = -2 * y
Enter boundaties:
Enter x_0: Enter y_0: Enter x_n: Enter h: Enter eps:
Input parameters:
> x_0 = 0.0
> y_0 = 2.0
> x_n = 4.0
> h   = 0.1
> eps = 0.1

Enjoy results!

=== Report of Basic Euler ===

Result of Basic Euler is 0.0004369490010567903
Total iterations: 81
```

```
Stop at: 3.999999999999994

=== Report of Corrected Euler ===

Result of Corrected Euler is 0.000713812404891529
Total iterations: 41
Stop at: 4.000000000000002

=== Report of Runge-Kutta 4 ===

Result of Runge-Kutta 4 is 0.0006710098386727152
Total iterations: 41
Stop at: 4.000000000000002
```

Листинг 5: Результаты работы программы 1

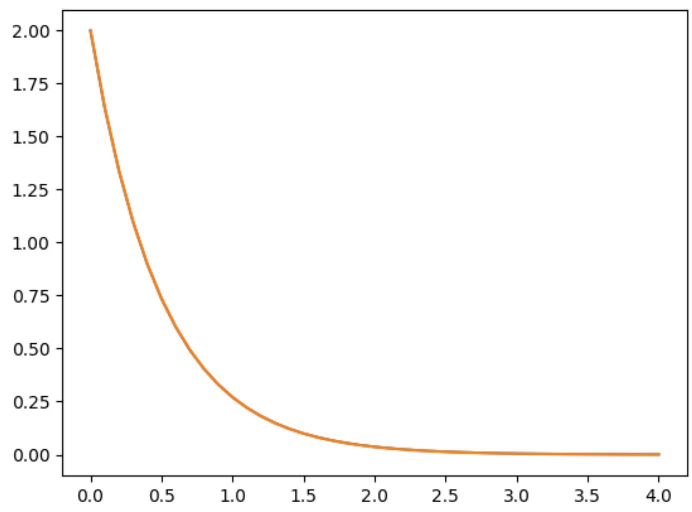

Рис. 1: График полученного графика 1

```
$ bash ci/run.sh < res/2.txt
=== Welcome to differential eqsolver ===

Here you can solve differential equations using:
- Basic Euler
- Corrected Euler
- Runge-Kutta 4
- Milne

Choice a one of first-order equations:
[0]: dy/dx = y + (1 + x) * y ^ 2
[1]: dy/dx = -2 * y
[2]: dy/dx = 2 * x * exp(x ** 2) / (exp(x ** 2) + 1)
Enter a number of equation i in [0..2]: Taken: dy/dx = 2 * x * exp(x ** 2) / (exp(x **
    2) + 1)
Enter boundaties:
Enter x_0: Enter y_0: Enter x_n: Enter h: Enter eps:
Input parameters:
> x_0 = 0.0
> y_0 = 0.69314718056
> x_n = 4.0
> h   = 0.1
> eps = 0.001

Enjoy results!

=== Report of Basic Euler ===

Result of Basic Euler is 15.99181773714573
```

```
Total iterations: 11239
Stop at: 3.999999999998924

=== Report of Corrected Euler ===

Result of Corrected Euler is 16.00001130507996
Total iterations: 3524
Stop at: 4.000000000000414

=== Report of Runge-Kutta 4 ===

Result of Runge-Kutta 4 is 16.00000010821849
Total iterations: 737
Stop at: 3.9999999999999227
```

Листинг 6: Результаты работы программы 2

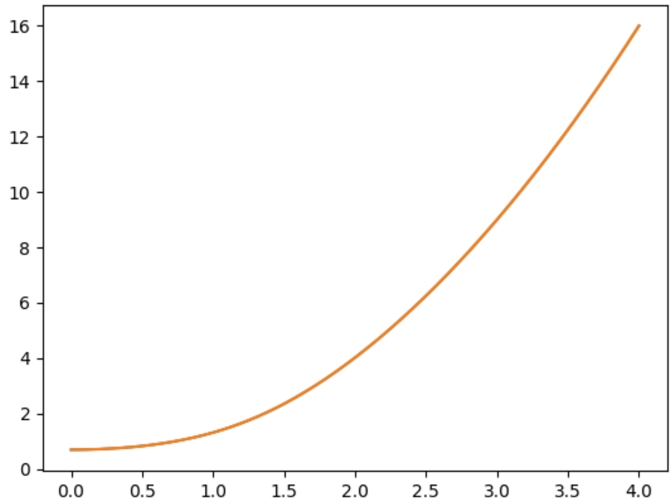

Рис. 2: График полученного графика 2

4 Вывод

Выполнив данную лабораторную я познакомился с базовыми методами решения дифференциальных уравнений. Точность методов контролировалась правилом Рунге. Метод Эйлера гораздо менее эффективнее, чем метод Рунге-Кутта, что и ожидалось, ведь точность первого метода $O(n)$, а второго аж $O(n^4)$. Графики при малых шагах на глаз совпали с дейтсвительными ответами.

Список литературы

[1] Лекции Татьяны Алексеевны Малышевой