Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

КТиУ, кафедра Информатики и Прикладной Математики

Лабораторная работа №4

по дисциплине

«Вычислительная математика»

«Решение ОДУ(задача Коши) методом Милна»

Выполнил:

Студент группы P3217

Григорьев Георгий

Санкт-Петербург

2018 г.

Задается ОбДифУр вида y’ + f(x,y) = 0 , пользователь задает начальные условия (x0, y0), конец отрезка и точность.
Программа сама вычисляет шаг в зависимости от точности для нахождения массива значений x и y.
Используя интерполирование 3-й работы строим график.
У кого 3-я работа была аппроксимация, строит график по полученным данным, задав очень маленькую точность.

//Не забудьте вставить примеры в отчет!
+ пару примеров ОДУ - с заранее известным решением вида y = f(x), где изначально y’ зависело не только от x.

Заданные обыкновенные дифференциальные уравнения:

$$1.\ y'(t) = -y(t) + \sqrt{y}(t)$$
$$2.\ y'(t) = 0.01y(1-y)$$
$$3.\ y'(t) = y^3(t)$$
$$4.\ y'(t) = 1/\log t$$
$$5.\ y'(t) = sin^2(t)$$

Рассмотрим решение уравнения на интервале $[t_i, t_{i+1}]$. Будем считать, что решение в точках $t_0, t_1, t_2, \ldots, t_i$ уже найдено, и значения в этих точках будем использовать для нахождения значения $x(t_{i+1})$.

Проинтегрируем уравнение (9.1) на интервале $[t_i, t_{i+1}]$ и получим соотношение [2]

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(t, x(t))\, dt$$

Подставив в (9.20) вместо функции $f(t, x(t))$ интерполяционный полином Ньютона, построенный по точкам $(t_{k-3}, x_{k-3}), (t_{k-2}, x_{k-2}), (t_{k-1}, x_{k-1}), (t_k, x_k)$ получаем первое приближение — прогноз Милна $\tilde{x_{k+1}}$ для значения функции в точке t_{k+1}[2]

$$\tilde{x}_{k+1} = x_{k-3} + \frac{4h}{3}(2f(t_{k-2}, x_{k-2}) - f(t_{k-1}, x_{k-1}) + 2f(t_k, x_k)) \quad (9.23)$$

Следующий полином Ньютона для функции $f(t, x(t))$ построим по точкам $(t_{k-2}, x_{k-2}), (t_{k-1}, x_{k-1}), (t_k, x_k)$ и новой точке $(t_{k+1}, \tilde{x}_{k+1})$, после чего подставляем его в (9.20) и получаем второе приближение — корректор Милна [2]

$$x_{k+1} = x_{k-1} + \frac{h}{3}(f(t_{k-1}, x_{k-1}) + 4f(t_k, x_k) + f(t_{k+1}, \tilde{x}_{k+1})) \quad (9.24)$$

В методе Милна для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы (9.23), (9.24), а первые четыре точки можно получить методом Рунге–Кутта.

Существует модифицированный метод Милна. В нём сначала вычисляется первое приближение по формуле (9.23), затем вычисляется управляющий параметр [2]

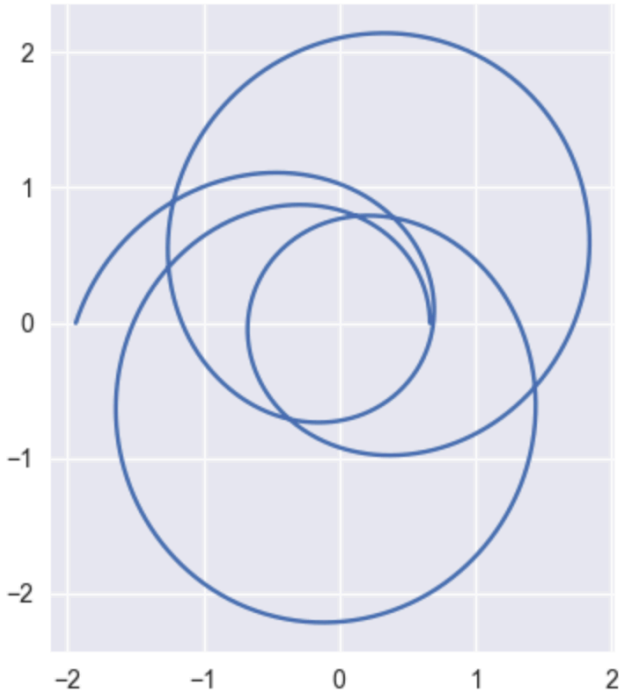
$$m_{k+1} = \tilde{x}_{k+1} + \frac{28}{29}(x_k - \tilde{x}_k) \quad (9.25)$$

После чего вычисляется значение второго приближения — корректор Милна по формуле

$$x_{k+1} = x_{k-1} + \frac{h}{3}(f(t_{k-1}, x_{k-1}) + 4f(t_k, x_k) + f(t_{k+1}, m_{k+1})) \quad (9.26)$$

В модифицированном методе Милна первые четыре точки можно получить методом Рунге-Кутта, а для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы (9.23), (9.25), (9.26).

Решение для 1. (точность 2000 шагов, начальная точка $\pi/2$, конечная 7π)



Листинг программы:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import scipy.integrate as integrate


def f1(t, phi):
    u, udot = t
    return [udot, -u + sqrt(u)]


def f2(t, y):
```

```
    return 0.01 * y * (1 - y)


def f3(t, y):
    return y**3


def f4(t, y):
    return 1 / np.log(y)


def f5(t, y):
    return np.sin(y) ** 2


if __name__ == '__main__':
    n = input("Choose precision (n) (Enter for 2000): ")
    if not n:
        n = 2000
    else:
        n = int(n)
    i = int(input("Choose function:\n 1. -u + sqrt(u)\n 2. 0.01*y*(1-y)
\n 3. y**3 \n 4. 1/log(y) \n 5. sin(y) ** 2\n"))
    tends = [7 * np.pi, 3000, 100, 100, 3]
    tend = input(f"Choose end of interpolation range (Enter for {tends[i
- 1]}): ")
    if not tend:
        tend = tends[i - 1]
    else:
        tend = float(tend)
    tinits = [[1.49907, 0], 0.5, -1, 10, 0.1]
    tinit = input(f"Choose initial point (Enter for {tinits[i - 1]}): ")
    if not tinit:
        tinit = tinits[i - 1]
    else:
        float(tinit)

    if i == 1:
        sqrt = np.sqrt
        cos = np.cos
        sin = np.sin

        phi = np.linspace(0, tend, n)
        t = integrate.odeint(f1, tinit, phi)
        u, udot = t.T
        fig, ax = plt.subplots()
        ax.plot(1 / u * cos(phi), 1 / u * sin(phi))
        ax.set_aspect('equal')
        plt.grid(True)
        plt.show()
    elif i == 2:
        trange = np.linspace(0.1, tend, n)
```

```
        f = f2
    elif i == 3:
        trange = np.linspace(0.1, tend, n)
        f = f3
    elif i == 4:
        trange = np.linspace(2, tend, n)
        f = f4
    elif i == 5:
        trange = np.linspace(-3, tend, n)
        f = f5

    t = integrate.odeint(f, tinit, trange)
    plt.plot(trange, t)
    plt.show()
```

Вывод: Метод Милна может использоваться для универсального решения дифференциальных уравнений, т.к. шаг подбирается адаптивно и вычисления сбалансированы. Точность по сравнению с другими методами отличается незначительно.