Федеральное государственное автономное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский университет
ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ
ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА № 6

по дисциплине

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №14

Выполнила
Студентка группы P32151
Ярусова Анна Александровна
Преподаватель:
Машина Екатерина Алексеевна

Санкт-Петербург,
2023

Цель

Решить задачу Коши для обыкновенных дифференциальных уравнений численными методами

Задание

1. № варианта определяется как номер в списке группы согласно ИСУ;
2. В программе численные методы решения обыкновенных дифференциальных уравнений (ОДУ) должен быть реализован в виде отдельного класса /метода/функции;
3. Пользователь выбирает ОДУ вида y’ = f (x, y) (не менее трех уравнений), из тех, которые предлагает программа;
4. Предусмотреть ввод исходных данных с клавиатуры: начальные условия , интервал дифференцирования , шаг h, точность ;
5. Для исследования использовать одношаговые методы и многошаговые методы (метод Милна, модифицированный Эйлера, Рунге-Кутта 4 порядка);
6. Составить таблицу приближенных значений интеграла дифференциального уравнения, удовлетворяющего начальным условиям, для всех методов, реализуемых в программе;
7. Для оценки точности одношаговых методов использовать правило Рунге
8. Для оценки точности многошаговых методов использовать точное решение задачи
9. Построить графики точного решения и полученного приближенного решения (разными цветами)
10. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных.
11. Проанализировать результаты работы программы

Описание метода, расчетные формулы

<u>Модифицированный метод Эйлера</u>

$$\boldsymbol{y_{i+1} = y_i + \frac{h}{2}\left[f(x_i, y_i) + f\big(x_{i+1}, y_i + hf(x_i, y_i)\big)\right],\ i = 0, 1 \ldots.}$$

Метод Рунге-Кутта

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

$$k_1 = h \cdot f(x_i, y_i)$$

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

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

$$k_4 = h \cdot f(x_i + h, y_i + k_3)$$

Метод Милна

а) этап прогноза:

$$y_i^{\text{прогн}} = y_{i-4} + \frac{4h}{3}(2f_{i-3} - f_{i-2} + 2f_{i-1})$$

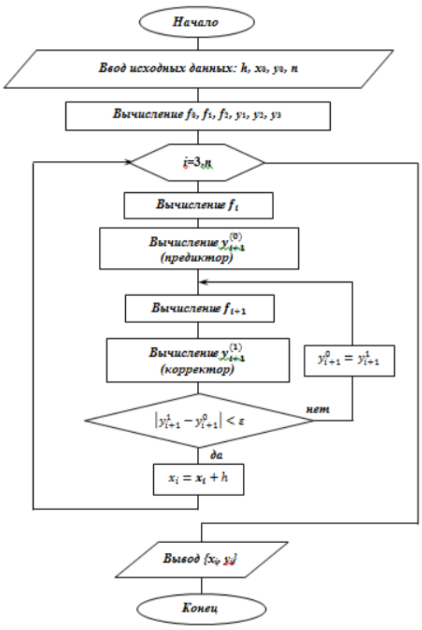
б) этап коррекции:

$$y_i^{\text{корр}} = y_{i-2} + \frac{h}{3}(f_{i-2} + 4f_{i-1} + f_i^{\text{прогн}})$$

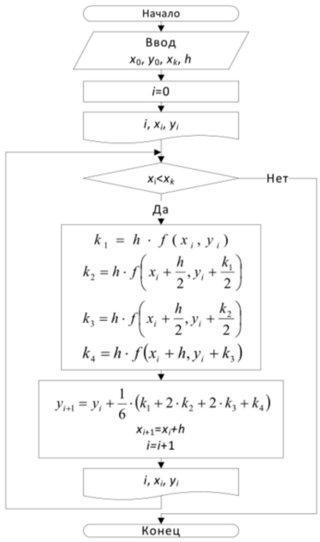
$$f_i^{\text{прогн}} = f(x_i, y_i^{\text{прогн}})$$

Блок-схема

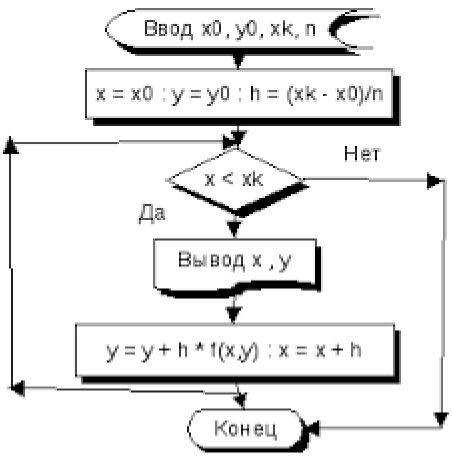
Милн



Рунге-Кутта



<u>Модифицированный Эйлер:</u>



Листинг численного метода

<u>Милн:</u>

```
def prediction(equation, x, y, h, i):
    return y[i - 4] + 4 * h / 3 * (2 * equation(x[i - 3], y[i - 3]) - equation(x[i - 2], y[i - 2]) + 2 * equation(x[i - 1], y[i - 1]))


def correction(equation, x, y, h, i, y_pred):
    return y[i - 2] + h / 3 * (equation(x[i - 2], y[i - 2]) + 4 * equation(x[i - 1], y[i - 1]) + equation(x[i], y_pred))


def miln_method(equation, x0, xn, y0, h, accuracy):
    print("Miln method")
```

```
    n = ceil((xn - x0) / h) + 1
    if n < 4:
        print("Error: h is too big, not enough points for Miln
method")
        return None, None
    x = [x0 + i * h for i in range(n)]
    runge_kutta_y, runge_kutta_x =
runge_kutta_method(equation, x[0], x[3], y0, h, accuracy,
True)
    y = runge_kutta_y[:4]
    print("Prediction || Correction || abs(Prediction -
Correction) || accuracy")
    for i in range(4, n):
        y_pred = prediction(equation, x, y, h, i)
        y_corr = correction(equation, x, y, h, i, y_pred)
        print(f"{round(y_pred, 6)} || {round(y_corr, 6)} ||
{round(abs(y_pred - y_corr), 6)} || {accuracy}")
        while abs(y_corr - y_pred) > accuracy:
            y_pred = y_corr
            y_corr = correction(equation, x, y, h, i, y_pred)
            print(f"{round(y_pred, 6)} || {round(y_corr, 6)}
|| {round(abs(y_pred - y_corr), 6)} || {accuracy}")
        y.append(y_corr)
    print("--------------------------------------")
    return y, x
```

Модифицированный Эйлер:

```
def euler(equation, x_prev, x, y_prev, h):
    return y_prev + h / 2 * (equation(x_prev, y_prev) +
equation(x, y_prev + h * equation(x_prev, y_prev)))


def modify_euler_method(equation, x0, xn, y0, h, accuracy):
    x = np.arange(x0, xn + h, h)
    y = np.zeros(len(x))
    y[0] = y0
    for i in range(1, len(x)):
        y[i] = euler(equation, x[i - 1], x[i], y[i - 1], h)
    return y, x
```

Рунге-Кутта:

```
def runge_kutta(equation, x, y, h):
```

```
    k1 = h * equation(x, y)
    k2 = h * equation(x + h / 2, y + k1 / 2)
    k3 = h * equation(x + h / 2, y + k2 / 2)
    k4 = h * equation(x + h, y + k3)
    return y + (k1 + 2 * k2 + 2 * k3 + k4) / 6


def runge_kutta_method(equation, x0, xn, y0, h, accuracy,
for_miln=False):
    print("Runge-Kutta method")
    y_prev = y0
    y_curr = 0
    y_curr_2 = accuracy + 1
    print("h || y_curr || y_curr_2 || abs(y_curr - y_curr_2)
|| accuracy")
    while abs(y_curr - y_curr_2) > accuracy:
        y_curr = runge_kutta(equation, x0, y_prev, h)
        y_curr_2 = runge_kutta(equation, x0, y_prev, h / 2)
        y_curr_2 = runge_kutta(equation, x0 + h / 2, y_curr_2,
h / 2)
        print(f"{round(h, 6)} || {round(y_curr, 6)} ||
{round(y_curr_2, 6)} || {round(abs(y_curr - y_curr_2), 6)} ||
{accuracy}")
        h /= 2
    h *= 2
    x = np.arange(x0, xn + h, h)
    y = [0 for i in range(len(x))]
    y[0] = y0
    for i in range(1, len(x)):
        y[i] = runge_kutta(equation, x[i - 1], y[i - 1], h)
    print("------------------------------------")
    return y, x
```

Пример работы

<u>Пример 1:</u>

Choose the equation:

1. y' = 2x (default)

2. y' = e^(-3x)

3. y' = cos(x)

1

Enter interval:
Enter left border x0 (default 0):
-10
Enter right border xn (default 1):
10
Enter y(x0) (default 0):
-15
Enter h (default 0.1):
1
Enter accuracy (default 0.01):
0.1
Runge-Kutta method
h || y_curr || y_curr_2 || abs(y_curr - y_curr_2) || accuracy
1.0 || -34.0 || -34.0 || 0.0 || 0.01

Miln method
Runge-Kutta method
h || y_curr || y_curr_2 || abs(y_curr - y_curr_2) || accuracy
1.0 || -34.0 || -34.0 || 0.0 || 0.01

Prediction || Correction || abs(Prediction - Correction) || accuracy
-79.0 || -79.0 || 0.0 || 0.01
-90.0 || -90.0 || 0.0 || 0.01
-99.0 || -99.0 || 0.0 || 0.01
-106.0 || -106.0 || 0.0 || 0.01
-111.0 || -111.0 || 0.0 || 0.01
-114.0 || -114.0 || 0.0 || 0.01
-115.0 || -115.0 || 0.0 || 0.01
-114.0 || -114.0 || 0.0 || 0.01
-111.0 || -111.0 || 0.0 || 0.01
-106.0 || -106.0 || 0.0 || 0.01
-99.0 || -99.0 || 0.0 || 0.01
-90.0 || -90.0 || 0.0 || 0.01
-79.0 || -79.0 || 0.0 || 0.01
-66.0 || -66.0 || 0.0 || 0.01
-51.0 || -51.0 || 0.0 || 0.01
-34.0 || -34.0 || 0.0 || 0.01
-15.0 || -15.0 || 0.0 || 0.01

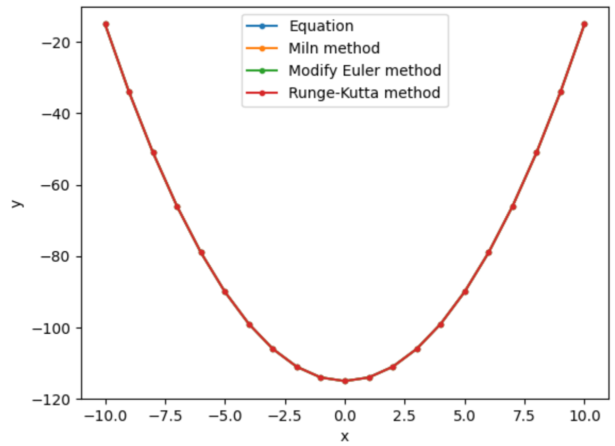
Results:
Miln method:
x = -10.0 y = -15.0
x = -9.0 y = -34.0
x = -8.0 y = -51.0
x = -7.0 y = -66.0

x = -6.0 y = -79.0
x = -5.0 y = -90.0
x = -4.0 y = -99.0
x = -3.0 y = -106.0
x = -2.0 y = -111.0
x = -1.0 y = -114.0
x = 0.0 y = -115.0
x = 1.0 y = -114.0
x = 2.0 y = -111.0
x = 3.0 y = -106.0
x = 4.0 y = -99.0
x = 5.0 y = -90.0
x = 6.0 y = -79.0
x = 7.0 y = -66.0
x = 8.0 y = -51.0
x = 9.0 y = -34.0
x = 10.0 y = -15.0

Modify Euler method:
x = -10.0 y = -15.0
x = -9.0 y = -34.0
x = -8.0 y = -51.0
x = -7.0 y = -66.0
x = -6.0 y = -79.0
x = -5.0 y = -90.0
x = -4.0 y = -99.0
x = -3.0 y = -106.0
x = -2.0 y = -111.0
x = -1.0 y = -114.0
x = 0.0 y = -115.0
x = 1.0 y = -114.0
x = 2.0 y = -111.0
x = 3.0 y = -106.0
x = 4.0 y = -99.0
x = 5.0 y = -90.0
x = 6.0 y = -79.0
x = 7.0 y = -66.0
x = 8.0 y = -51.0
x = 9.0 y = -34.0
x = 10.0 y = -15.0

Runge-Kutta method:
x = -10.0 y = -15.0
x = -9.0 y = -34.0
x = -8.0 y = -51.0
x = -7.0 y = -66.0

x = -6.0 y = -79.0
x = -5.0 y = -90.0
x = -4.0 y = -99.0
x = -3.0 y = -106.0
x = -2.0 y = -111.0
x = -1.0 y = -114.0
x = 0.0 y = -115.0
x = 1.0 y = -114.0
x = 2.0 y = -111.0
x = 3.0 y = -106.0
x = 4.0 y = -99.0
x = 5.0 y = -90.0
x = 6.0 y = -79.0
x = 7.0 y = -66.0
x = 8.0 y = -51.0
x = 9.0 y = -34.0
x = 10.0 y = -15.0



Пример 2:

Choose the equation:
1. y' = 2x (default)
2. y' = e^(-3x)
3. y' = cos(x)
3
Enter interval:
Enter left border x0 (default 0):
0
Enter right border xn (default 1):
1
Enter y(x0) (default 0):
0.1
Enter h (default 0.1):

0.1
Enter accuracy (default 0.01):
0.01
Runge-Kutta method
h || y_curr || y_curr_2 || abs(y_curr - y_curr_2) || accuracy
0.1 || 0.199833 || 0.199833 || 0.0 || 0.01

Miln method
Runge-Kutta method
h || y_curr || y_curr_2 || abs(y_curr - y_curr_2) || accuracy
0.1 || 0.199833 || 0.199833 || 0.0 || 0.01

Prediction || Correction || abs(Prediction - Correction) || accuracy
0.489415 || 0.489418 || 3e-06 || 0.01
0.579423 || 0.579426 || 3e-06 || 0.01
0.66464 || 0.664643 || 3e-06 || 0.01
0.744215 || 0.744218 || 3e-06 || 0.01
0.817354 || 0.817356 || 3e-06 || 0.01
0.883325 || 0.883327 || 3e-06 || 0.01
0.941469 || 0.941471 || 2e-06 || 0.01

Results:
Miln method:
x = 0.0 y = 0.1
x = 0.1 y = 0.199833
x = 0.2 y = 0.298669
x = 0.3 y = 0.39552
x = 0.4 y = 0.489418
x = 0.5 y = 0.579426
x = 0.6 y = 0.664643
x = 0.7 y = 0.744218
x = 0.8 y = 0.817356
x = 0.9 y = 0.883327
x = 1.0 y = 0.941471

Modify Euler method:
x = 0.0 y = 0.1
x = 0.1 y = 0.19975
x = 0.2 y = 0.298504
x = 0.3 y = 0.395274
x = 0.4 y = 0.489094
x = 0.5 y = 0.579026
x = 0.6 y = 0.664172
x = 0.7 y = 0.743681
x = 0.8 y = 0.816758
x = 0.9 y = 0.882674

x = 1.0 y = 0.94077

Runge-Kutta method:

x = 0.0 y = 0.1

x = 0.1 y = 0.199833

x = 0.2 y = 0.298669

x = 0.3 y = 0.39552

x = 0.4 y = 0.489418

x = 0.5 y = 0.579426

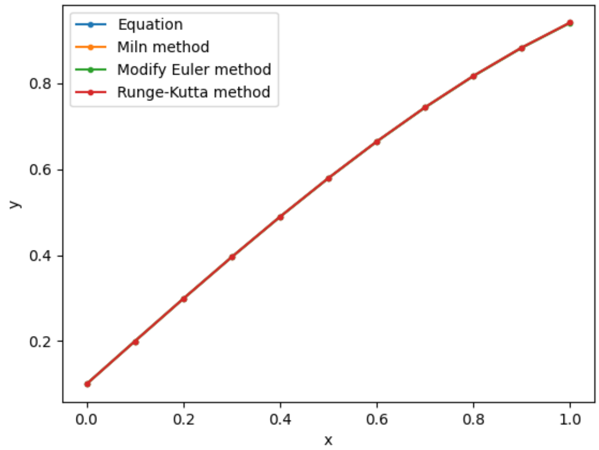
x = 0.6 y = 0.664642

x = 0.7 y = 0.744218

x = 0.8 y = 0.817356

x = 0.9 y = 0.883327

x = 1.0 y = 0.941471



Вывод

В ходе выполнения лабораторной работы я узнала о решении задачи Коши для обыкновенных дифференциальных уравнений численными методами, написала для этого код на языке Python.