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

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №22

Студенты:
Самарина Арина Анатольевна
Суржицкий Арсений Арсентьевич
Группа P3266

Преподаватель:
Машина Екатерина Александровна

Санкт-Петербург, 2024

1. Цель работы

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

2. Описание методов, расчётные формулы

Метод Эйлера

Метод Э й л е р а (1707–1783) основан на разложении искомой функции $y(x)$в ряд Тейлора в окрестностях узлов $x = x_i$ $(i = 0, 1, \dots)$, в котором отбрасываются все члены, содержащие производные второго и более высоких порядков:

$$Y(x_i + h) = Y(x_i) + Y'(x_i) \cdot h + O(h^2)$$

Полагаем: $Y'(x_i) = f(x_i, Y(x_i)) = f(x_i, y_i) = \frac{y_{i+1} - y_i}{h}$

Введем последовательность равноотстоящих точек $x_0, x_1, \dots, x_n$ (узлов), выбрав малый шаг $h = x_{i+1} - x_i = const$. Тогда получаем **формулу Эйлера**:

$$\boldsymbol{y_{i+1} = y_i + hf(x_i, y_i)} \tag{7}$$

При i=0 находим значение сеточной функции y_1 при $x = x_1$:

$$y_1 = y_0 + hf(x_0, y_0)$$

Значение y_0 задано начальным условием:

$$\boldsymbol{y_0 = Y_0} \tag{8}$$

Аналогично могут быть найдены значения сеточной функции в других узлах:

$$y_2 = y_1 + hf(x_1, y_1)$$

.....

$$y_n = y_{n-1} + hf(x_{n-1}, y_{n-1})$$

Построенный алгоритм называется методом Эйлера. Разностная схема этого метода представлена соотношениями (7), (8). Они имеют вид рекуррентных формул, с помощью которых значение сеточной функции y_{i+1} в любом узле x_{i+1} вычисляется по ее значению y_i в предыдущем узле x_i. Поэтому метод Эйлера относится к ***одношаговым*** методам.

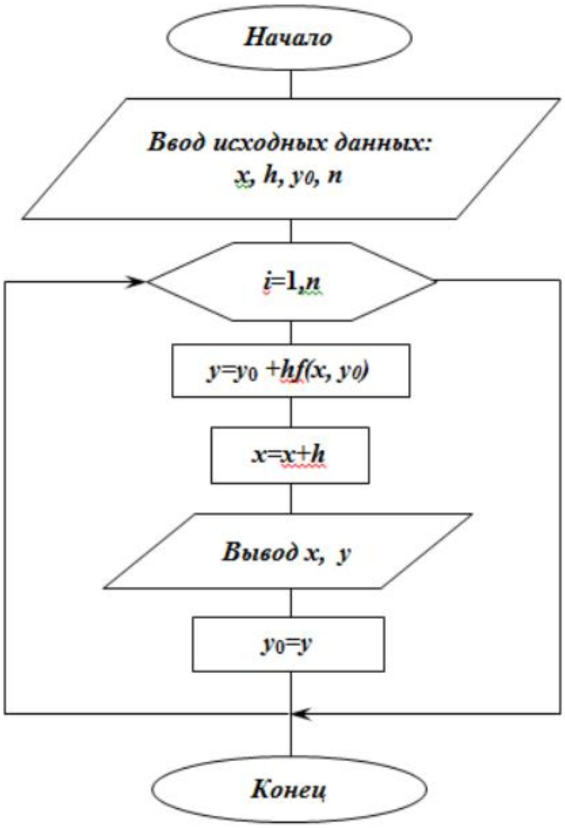

Рис.1 Блок-схема метода Эйлера

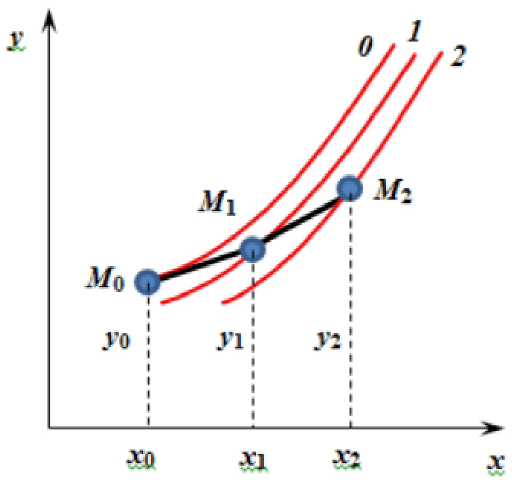

Рис. 2. Геометрическая интерпретация метода Эйлера

На рис.2 изображены первые два шага, т. е. проиллюстрировано вычисление сеточной функции в точках x_1, x_2. Интегральные кривые 0,1,2 описывают точные решения уравнения (5). При этом кривая 0 соответствует точному решению задачи Коши (5), (6), так как она проходит через начальную точку $M_0(x_0, y_0)$. Точки M_1, M_2 получены в результате численного решения задачи Коши методом Эйлера. Их отклонения от кривой 0 характеризуют погрешность метода. При выполнении каждого шага мы фактически попадаем на другую интегральную кривую. Отрезок M_0M_1 – отрезок касательной к кривой 0 в точке M_0, ее наклон характеризуется значением производной $Y'(x_0) = f(x_0, y_0)$. Погрешность появляется потому, что приращение значения функции при переходе от x_0 к x_1 заменяется приращением ординаты касательной к кривой 0 в точке M_0.

Касательная M_1M_2 уже проводится к другой интегральной кривой 1. Таким образом, погрешность метода Эйлера приводит к тому, что на каждом шаге приближенное решение переходит на другую интегральную кривую.

Метод Эйлера имеет ***первый порядок точности*** $\delta_n = O(h)$

Модификации метода Эйлера.

Рассмотрим уравнение (2) в окрестностях узлов $x = x_i + h/2$ (i=0, 1…), являющихся серединами отрезков $[x_i, x_{i+1}]$. В левой части (2) заменим производную центральной разностью, а в правой части заменим значение функции $f(x_i + h/2, Y(x_i + h/2))$ средним арифметическим значений функции $f(x, Y)$ в точках (x_i, y_i) и (x_{i+1}, y_{i+1}). Тогда:

$$\frac{y_{i+1} - y_i}{h} = \frac{1}{2}[f(x_i, y_i) + f(x_{i+1}, y_{i+1})].$$

Отсюда:

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i, y_i) + f(x_{i+1}, y_{i+1})]. \qquad (9)$$

Полученная схема является неявной, поскольку искомое значение y_{i+1} входит в обе части соотношения (9) и его нельзя выразить явно. Для вычисления y_{i+1} можно применить один из итерационных методов. Если имеется хорошее начальное приближение y_i, то можно построить решение с использованием двух итераций следующим образом. Считая y_i начальным приближением, вычисляем первое приближение $\tilde{y}_{i+1}$ по формуле метода Эйлера (7):

$$\tilde{y}_{i+1} = y_i + hf(x_i, y_i)$$

Вычисленное значение $\tilde{y}_{i+1}$ подставляем вместо y_{i+1} в правую часть соотношения (9) и находим окончательное значение:

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i, y_i) + f(x_{i+1}, \tilde{y}_{i+1})] \text{ или:}$$

$$\boldsymbol{y_{i+1} = y_i + \frac{h}{2}\big[f(x_i, y_i) + f\big(x_{i+1}, y_i + hf(x_i, y_i)\big)\big],\ i = 0, 1 \ldots} \qquad (10)$$

Данные рекуррентные соотношения описывают новую разностную схему, являющуюся ***модифицированным методом Эйлера***, которая называется методом ***Эйлера с пересчетом***. Метод Эйлера с пересчетом имеет ***второй порядок точности*** $\delta_n = O(h^2)$.

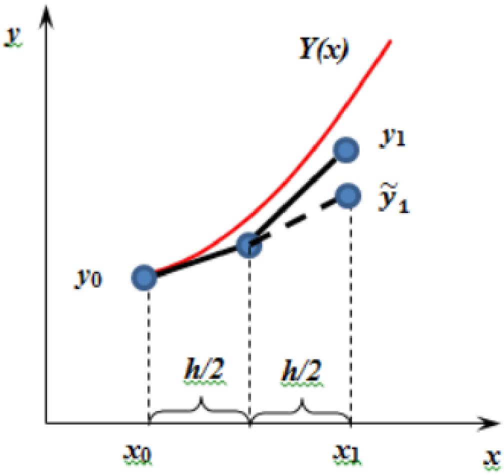

Рис.3 Геометрическая интерпретация метода Эйлера с пересчетом

На рис.3 изображен первый шаг вычислений методом Эйлера с пересчетом. Касательная к кривой $Y(x)$ в точке x_0, y_0 проводится с угловым коэффициентом $y' = f(x_0, y_0)$ С ее помощью найдено значение $\tilde{y}_1$, которое используется затем для определения наклона касательной $f(x_1, \tilde{y}_1)$ в точке x_1, y_1. Отрезок с таким наклоном заменяет первоначальный отрезок касательной от точки $x_0 + h/2$ до точки x_1. В результате получается уточненное значение искомой функции y_1 в этой точке.

С помощью метода Эйлера с пересчетом можно проводить контроль точности решения путем сравнения значений $\tilde{y}_{i+1}$ и y_{i+1} и выбора на основании этого соответствующей величины шага h в каждом узле. Например, если величина $|y_{i+1} - \tilde{y}_{i+1}| > \varepsilon \cdot |y_{i+1}|$, значение h следует уменьшить. Используя эти оценки, можно построить алгоритм метода Эйлера с пересчетом с автоматическим выбором шага.

Метод Милна

Метод Милна относится к многошаговым методам и представляет один из методов прогноза и коррекции.

Для получения формул Милна используется первая интерполяционная формула Ньютона с разностями до третьего порядка.

Решение в следующей точке находится в два этапа. На первом этапе осуществляется прогноз значения функции, а затем на втором этапе - коррекция полученного значения. Если полученное значение y после коррекции существенно отличается от спрогнозированного, то проводят еще один этап коррекции. Если опять имеет место существенное отличие от предыдущего значения (т.е. от предыдущей коррекции), то проводят еще одну коррекцию и т.д.

Вычислительные формулы:

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:

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

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

Для начала счета требуется задать решения в трех первых точках, которые можно получить одношаговыми методами (например, методом Рунге-Кутта).

Метод требует несколько меньшего количества вычислений (достаточно только два раза вычислить $f(x, y)$, остальные берутся с предыдущих этапов).

Суммарная погрешность этого метода есть величина $\delta_n = O(h^4)$.

3. Листинг программы

Main.py

```
from odu import odu


def main():
    odu()


if __name__ == "__main__":
    main()
```

input_output.py

```
from functions import *
import matplotlib.pyplot as plt
import numpy as np


def try_to_convert_to_int(number):
    try:
```

```
        number_float = float(number)
        if number_float.is_integer():
            return int(number_float)
        else:
            return number_float
    except ValueError:
        return float(number)


def input_data():
    function, exact_y = choose_quation()
    x0 = choose_x()
    h = choose_h()
    n = choose_n()
    y0 = choose_y()
    eps = choose_eps()
    return function, exact_y, x0, eps, h, n, y0


def choose_x():
    while True:
        try:
            interval = input("Enter the x0 value:
").replace(",", ".")
            return try_to_convert_to_int(interval)

        except ValueError:
            print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue


def choose_y():
    while True:
        try:
            interval = input("Enter the y0 value:
").replace(",", ".")
            return try_to_convert_to_int(interval)

        except ValueError:
            print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue


def choose_h():
    while True:
```

```
        try:
            interval = input("Enter the h value(step range): 
").replace(",", ".")

            interval = try_to_convert_to_int(interval)
            if interval <= 0:
                print(f"h must be greater then 0")
                continue
            return interval

        except ValueError:
            print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue


def choose_n():
    while True:
        try:
            interval = int(
                input("Enter the N value(count of dots): 
").replace(",", ".")
            )

            if interval < 2:
                print(f"N must be greater then 1")
                continue
            return interval

        except ValueError:
            print("h must be int")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue


def choose_eps():
    while True:
        try:
            interval = input("Enter the epsilon value: 
").replace(",", ".")
            interval = try_to_convert_to_int(interval)
            if interval <= 0 or interval >= 1:
                print(f"epsilon must be in (0, 1)")
                continue
            return interval

        except ValueError:
```

```
            print("Incorrect number entered")
            continue

        except UnboundLocalError:
            print("Incorrect number entered")
            continue


def choose_quation():
    while True:
        try:
            print("Choose the quation:")
            input_func = int(
                input("1) y + (1+x)\n2) x+y\n3) sin(x) -
y\n").replace(",", ".")
            )
            if input_func == 1:
                f = f1
                exact_y = y1
                break
            elif input_func == 2:
                f = f2
                exact_y = y2
                break
            elif input_func == 3:
                f = f3
                exact_y = y3
                break
            else:
                print("Please, chose one of the available
options.")
        except ValueError:
            print("Please, chose one of the available options.")
    return f, exact_y


def draw_plot(ax, xs, ys, func, x0, y0, name, dx=0.01):
    ax.set_title(name)
    xss, yss = [], []
    a = xs[0]
    b = xs[-1]
    a -= dx
    b += dx
    x = a
    while x <= b:
        xss.append(x)
        yss.append(func(x, x0, y0))
        x += dx
    # Рисуем график точного решения
    ax.plot(xss, yss, "g", label="Exact Solution")

    # Рисуем точки численного решения
    ax.scatter(xs, ys, c="r", label="Numerical Solution")
```

```
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.legend()


def draw_dense_plot(ax, xs_dense, ys_dense):
    ax.plot(xs_dense, ys_dense, "b--", label="Numerical Function
(Dense)")
```

functions.py

```
from numpy import exp, sin, cos


def f1(x, y):
    try:
        return y + (1 + x) * y
    except OverflowError:
        print("Result too large")
        exit()


def y1(x, x0, y0):
    return y0 * exp((x**2 - x0**2) / 2 + (x - x0))


def f2(x, y):
    return x + y


def y2(x, x0, y0):
    return exp(x - x0) * (y0 + x0 + 1) - x - 1


def f3(x, y):
    return sin(x) - y


def y3(x, x0, y0):
    return (
        (2 * exp(x0) * y0 - exp(x0) * sin(x0) + exp(x0) *
cos(x0)) / (2 * exp(x))
        + (sin(x)) / 2
        - (cos(x)) / 2
    )
```

odu.py

```
from input_output import *
import warnings
```

```
def odu():
    f, exact_y, x0, eps, h, n, y0 = input_data()
    xs = [x0 + i * h for i in range(n)]

    print()
    methods = [
        ("Euler", euler_method),
        ("4th order Runge-Kutta",
fourth_order_runge_kutta_method),
        ("Milne", milne_method),
    ]

    fig, axes = plt.subplots(len(methods), 1, figsize=(10, 15))

    with warnings.catch_warnings():
        warnings.filterwarnings("error",
category=RuntimeWarning)
        try:
            for ax, (name, method) in zip(axes, methods):

                print(name + ":")

                ys = method(f, xs, y0, eps)

                xs_dense = np.arange(x0, xs[-1], h)
                ys_dense = method(f, xs_dense, y0, eps)
                print("y:\t[", *map(lambda x: round(x, 5), ys),
"]")
                print(
                    "y_exact:\t[", *map(lambda x:
round(exact_y(x, x0, y0), 5), xs), "]"
                )

                if method is milne_method:
                    inaccuracy = max(
                        [abs(exact_y(x, x0, y0) - y) for x, y in
zip(xs, ys)]
                    )
                    print(f"Error (max|y_exact - y_i|):
{inaccuracy}")
                else:
                    xs2 = []
                    for x1, x2 in zip(xs, xs[1:]):
                        xs2.extend([x1, (x1 + x2) / 2, x2])
                    ys2 = method(f, xs2, y0, eps)

                    p = 4 if method is
fourth_order_runge_kutta_method else 1
                    inaccuracy = max(
                        [abs(y1 - y2) / (2**p - 1) for y1, y2 in
zip(ys, ys2)]
```

```
                    )
                    print(f"Error (by Runge rule):
{inaccuracy}\n")

                draw_plot(ax, xs, ys, exact_y, x0, y0, name)
                draw_dense_plot(ax, xs_dense, ys_dense)

            plt.tight_layout()
            plt.show()
        except RuntimeWarning as e:
            print("Too large nums in solution, can't solve it.")
            exit()


import warnings


def euler_method(f, xs, y0, eps):
    warnings.filterwarnings("error", category=RuntimeWarning)
    try:
        ys = [y0]
        h = xs[1] - xs[0]
        for i in range(1, len(xs)):
            y_next = ys[i - 1] + h * f(xs[i - 1], ys[i - 1])
            ys.append(y_next)
        return ys
    except RuntimeWarning as e:
        print("Too large nums in solution:")
        exit()


def fourth_order_runge_kutta_method(f, xs, y0, eps):
    warnings.filterwarnings("error", category=RuntimeWarning)
    try:
        ys = [y0]
        h = xs[1] - xs[0]
        for i in range(1, len(xs)):
            k1 = h * f(xs[i - 1], ys[i - 1])
            k2 = h * f(xs[i - 1] + h / 2, ys[i - 1] + k1 / 2)
            k3 = h * f(xs[i - 1] + h / 2, ys[i - 1] + k2 / 2)
            k4 = h * f(xs[i - 1] + h, ys[i - 1] + k3)
            ys.append(ys[i - 1] + 1 / 6 * (k1 + 2 * k2 + 2 * k3
+ k4))
        return ys
    except RuntimeWarning as e:
        print("Too large nums in solution:")
        exit()


def milne_method(f, xs, y0, eps):
    with warnings.catch_warnings():
        warnings.filterwarnings("error",
category=RuntimeWarning)
```

```
        try:
            ys = fourth_order_runge_kutta_method(f, xs[:4], y0,
eps)
            h = xs[1] - xs[0]
            for i in range(4, len(xs)):
                pre_y = ys[i - 4] + 4 * h / 3 * (
                    2 * f(xs[i - 3], ys[i - 3])
                    - f(xs[i - 2], ys[i - 2])
                    + 2 * f(xs[i - 1], ys[i - 1])
                )
                cor_y = get_cor_y(xs, ys, f, pre_y, i, h)
                while abs(pre_y - cor_y) > eps:
                    pre_y = cor_y
                    cor_y = get_cor_y(xs, ys, f, pre_y, i, h)
                ys.append(cor_y)
            return ys
        except RuntimeWarning as e:
            print("Too large nums in solution:")
            exit()


def get_cor_y(xs, ys, f, pre_y, i, h):
    warnings.filterwarnings("error", category=RuntimeWarning)
    try:
        return ys[i - 2] + h / 3 * (
            f(xs[i - 2], ys[i - 2]) + 4 * f(xs[i - 1], ys[i -
1]) + f(xs[i], pre_y)
        )
    except RuntimeWarning as e:
        print("Too large nums in solution:")
        exit()
```

4. Примеры и результаты работы программы

5. Вывод

В ходе лабораторной работы были изучены и реализованы численные методы решения задачи Коши для ОДУ: метод Эйлера, усовершенствованный метод Эйлера и метод Милна. Программа показала корректность и эффективность этих методов. Усовершенствованный метод Эйлера и метод Милна обеспечили более высокую точность по сравнению с методом Эйлера. Работа подчеркнула важность выбора подходящего метода для достижения необходимой точности и эффективности.