# Университет ИТМО

Факультет программной инженерии и компьютерной техники

Дисциплина «Вычислительная математика»

**Отчет**

По лабораторной работе №5

Улучшенный метод Эйлера.

Выполнил:

*Терновский И.Е*

Преподаватель:

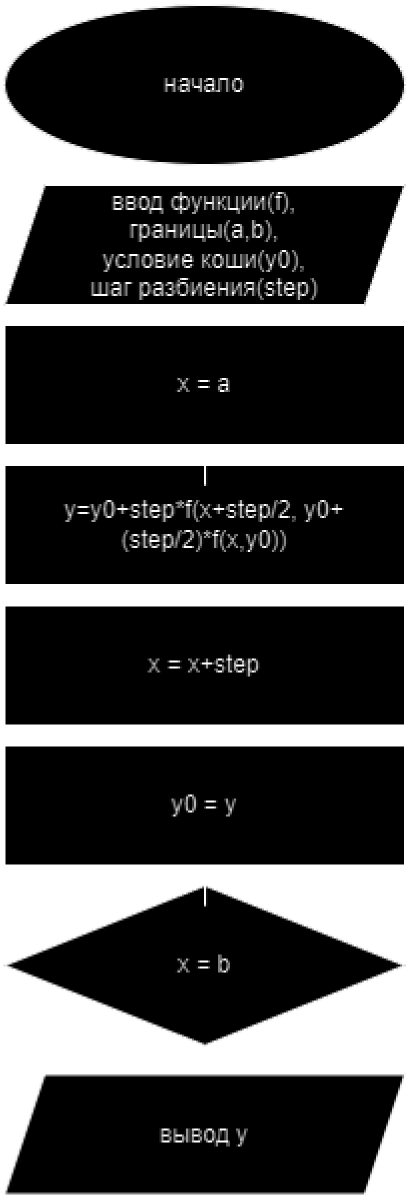
*Перл О.В*

Санкт-Петербург, 2024 г.

## Описание метода

Метод заключается в том, что мы приближаем решение нашего ОДУ некоторым набором ломанных линий, которые, кстати называются ломанными Эйлера. То есть мы разбиваем некоторый отрезок на некоторое количество разбиений и строим на каждом отрезке разбиения отрезок, который будет примерно повторять тот же отрезок на исходной функции. Сами же точки для построения ломанной Эйлера получаются из начальной точки (задачи Коши), а затем вычисляются рекуррентно.

## Блок-схема метода



P.S Как и в прошлой задаче, данная блок схема отражает “классический” метод, когда в функцию подается либо шаг, либо количество разбиений, в задании же метод немного модифицирован для нахождения решения с некоторой точностью

## Код численного метода:

```
import math

class Result:

    def first_function(x: float, y: float):
        return math.sin(x)

    def second_function(x: float, y: float):
        return (x * y) / 2

    def third_function(x: float, y: float):
        return y - (2 * x) / y

    def fourth_function(x: float, y: float):
        return x + y

    def default_function(x: float, y: float):
        return 0.0

    # How to use this function:
    # func = Result.get_function(4)
    # func(0.01)
    def get_function(n: int):
        if n == 1:
            return Result.first_function
        elif n == 2:
            return Result.second_function
        elif n == 3:
            return Result.third_function
        elif n == 4:
            return Result.fourth_function
        else:
            return Result.default_function

    #
    # Complete the 'solveByEulerImproved' function below.
    #
    # The function is expected to return a DOUBLE.
    # The function accepts following parameters:
    #  1. INTEGER f
    #  2. DOUBLE epsilon
    #  3. DOUBLE a
    #  4. DOUBLE y_a
    #  5. DOUBLE b
    #
    def solveByEulerImproved(self, f, epsilon, a, y_a, b):
        # Get the function to integrate
        func = Result.get_function(f)

        step_size = (b - a) / 1000  # initial step size
```

```
        current_x = a
        current_y = y_a
        while current_x < b:
            half_step = step_size / 2
            delta_y = step_size * func(current_x + half_step, current_y +
half_step * func(current_x, current_y))
            new_y = current_y + delta_y
            while abs(new_y - current_y) > epsilon:
                step_size = step_size / 2  # reduce step size
                half_step = step_size / 2
                delta_y = step_size * func(current_x + half_step, current_y +
half_step * func(current_x, current_y))
                new_y = current_y + delta_y
            current_y = new_y
            current_x += step_size
        return current_y


def main():
    print("Выберите функцию для интегрирования:")
    print("1: sin(x)")
    print("2: (x * y) / 2")
    print("3: y - (2 * x) / y")
    print("4: x + y")
    try:
        function_index = int(input("Введите номер функции: "))

        precision = float(input("Введите желаемую точность: "))
        start_x = float(input("Введите начальное значение x: "))
        start_y = float(input("Введите начальное значение y(x): "))
        end_x = float(input("Введите конечное значение x: "))
    except ValueError:
        print("Ошибка ввода. Попробуйте снова.")
        return

    result = Result().solveByEulerImproved(function_index, precision,
start_x, start_y, end_x)
    print(f"Результат: {result}")


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

Пример работы:

1)

```
Выберите функцию для интегрирования:
1: sin(x)
2: (x * y) / 2
3: y - (2 * x) / y
4: x + y
Введите номер функции: 1
Введите желаемую точность: 0.001
Введите начальное значение x: -5
Введите начальное значение y(x): 2
Введите конечное значение x: 5
Результат: 1.999999999999531
```

```
Выберите функцию для интегрирования:
1: sin(x)
2: (x * y) / 2
3: y - (2 * x) / y
4: x + y
Введите номер функции: 4
Введите желаемую точность: 0.001
Введите начальное значение x: 0
Введите начальное значение y(x): 0
Введите конечное значение x: 1
Результат: 0.7191409604997422
```

2)

```
Выберите функцию для интегрирования:
1: sin(x)
2: (x * y) / 2
3: y - (2 * x) / y
4: x + y
Введите номер функции: 1
Введите желаемую точность: asd
Ошибка ввода. Попробуйте снова.
```

3)

```
Выберите функцию для интегрирования:
1: sin(x)
2: (x * y) / 2
3: y - (2 * x) / y
4: x + y
Введите номер функции: 2
Введите желаемую точность: 0.001
Введите начальное значение x: 0
Введите начальное значение y(x): 1
Введите конечное значение x: 5
Результат: 518.0134636381541
```

4)

```
Выберите функцию для интегрирования:
1: sin(x)
2: (x * y) / 2
3: y - (2 * x) / y
4: x + y
Введите номер функции: 3
Введите желаемую точность: 0.001
Введите начальное значение x: 0
Введите начальное значение y(x): 1
Введите конечное значение x: 5
Результат: 3.3166872985852627
```

5)

# Вывод:

Улучшенный метод Эйлера является достаточно простым методом решения ОДУ, при этом классический метод Эйлера является одним из первых численных методов для решения ОДУ, и даже сам Коши использовал его для вычисления ОДУ. Метод достаточно прост в реализации, но сложнее чем классический метод Эйлера, так же он медленнее классического метода, так как в улучшенном методе вычисляются дополнительные точки. Из ограничений для применения, метод может быть нестабильным и может “взорваться” для жестких ОДУ. В сравнении с другими численными методами решения ОДУ метод Эйлера является одним из самых простых в реализации и одним из самых быстрых, так как вычислений в данном методе не так много. Очевидно, что с простотой вычислений приходит и неточность данных вычислений, поэтому даже улучшенный метод Эйлера является достаточно неточным по сравнению с методами Адамса и методом Рунге-Кутты. В то же время оба этих метода используют больше точек и соответственно требуют большего количества вычислений.
Алгоритмическая сложность метода: O(n), где n-число разбиений.
Численная ошибка в данном методе накапливается из неточности вычислений компьютера, а так же из неточности самого метода, а именно из за того что, метод аппроксимирует функцию прямыми линиями, которые очень часто не совсем подходят для описания сложной функции.