Санкт-Петербургский национальный
исследовательский университет ИТМО

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

Отчёт по лабораторной работе №5

Выполнил: Чан Дык Зюи

Группа: P32202

Преподаватель: Ольга Вячеславовна Перл

Санкт Петербург, 2023

Цель работы:

Решения задачи Коши –решения дифференциального уравнения по заданным начальным условиям

Одношаговые методы: Метод Рунге-Кутты 4-го порядка

Описание метода, расчетные формулы:

Наиболее часто используемый метод Рунге-Кутты для нахождения решений дифференциальных уравнений — это метод RK4, т. е. четырехкомпонентный метод Рунге-Кутты. Метод Рунге-Кутты дает приблизительное значение y в данной точке x. Метод Рунге-Кутта RK4 может решать только ОДУ первого порядка.

Начальное условие y0=f(x0), а корень x вычисляется в диапазоне от x0 до xn.

$$y' = F(x, y),\ y_0 = f(x_0) \to y = f(x)$$

Метод Рунге-Кутты 4-го порядка
(1) $y' = F(x, y),\ y_0 = f(x_0) \to y = f(x)$
(2) $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5)$

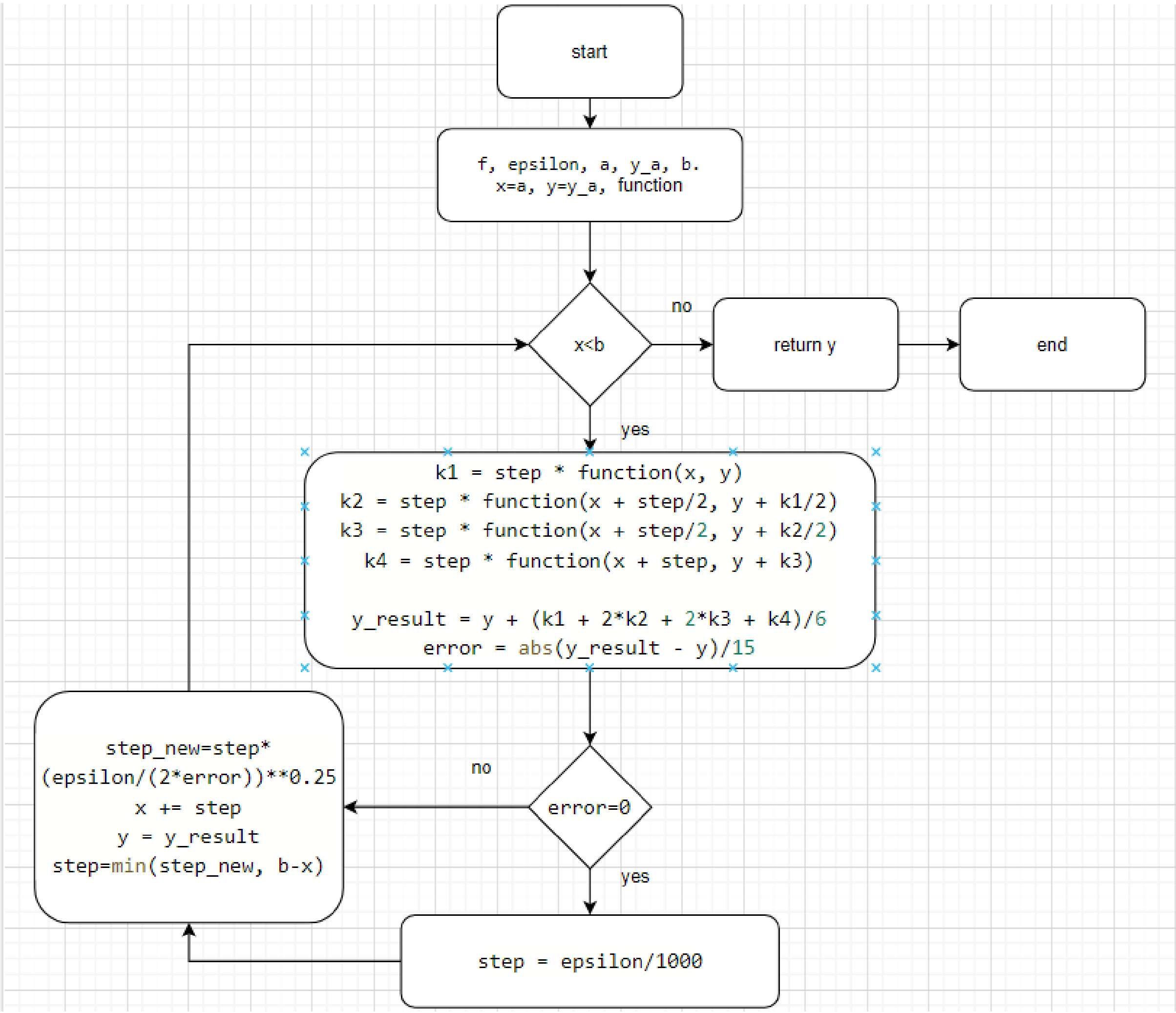
$k_1 = hF(x_n, y_n)$

$k_2 = hF\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$

$k_3 = hF\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$

$k_4 = hF(x_n + h, y_n + k_3)$

Блок-схема для метода Рунге-Кутты:



Код реализации решения на python:

```python
#!/bin/python3

import math
import os
import random
import re
import sys

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 'solveByRungeKutta' function below.
    #
    # The function is expected to return a DOUBLE.
```

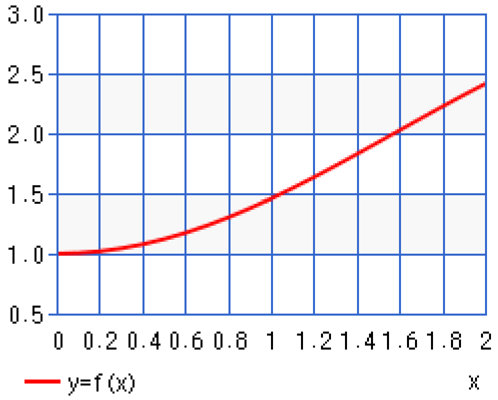
```
    # The function accepts following parameters:
    #  1. INTEGER f
    #  2. DOUBLE epsilon
    #  3. DOUBLE a
    #  4. DOUBLE y_a
    #  5. DOUBLE b
    #
    def solveByRungeKutta(f, epsilon, a, y_a, b):
        function = Result.get_function(f)

        x = a
        y = y_a

        step = 0.1


        while x < b:
            RungeKuttaCoeff1 = step * function(x, y)
            RungeKuttaCoeff2 = step * function(x + step/2, y + Ru
ngeKuttaCoeff1/2)
            RungeKuttaCoeff3 = step * function(x + step/2, y + Ru
ngeKuttaCoeff2/2)
            RungeKuttaCoeff4 = step * function(x + step, y + Rung
eKuttaCoeff3)

            # Calculate the new value of the function at the next
point x using the calculated Runge-Kutta coefficients
            y_result = y + (RungeKuttaCoeff1 + 2*RungeKuttaCoeff2
 + 2*RungeKuttaCoeff3 + RungeKuttaCoeff4)/6

            # Calculate the error between the new value and the
old value of the function
            error = abs(y_result - y)/15
            if error == 0:
                step = epsilon/1000
            step_new=step*(epsilon/(2*error))**0.25
            # Update the value of the independent variable x and
the value of the function y
            x += step
            y = y_result
            step=min(step_new, b-x)

        return y
# The value 15 in the error calculation comes from the fact that
the Runge-Kutta method is a fourth-order method, which means that
the error is proportional to h^5, where h is the step size.
Dividing by 15 gives an estimate of the error that is
proportional to h^4.
```

```python
# The value 0.25 in the step size calculation comes from the fact that the Runge-Kutta method is a fourth-order method, which means that the error is proportional to h^5, where h is the step size. Solving for h in the error formula and substituting into the step size formula gives step_new=step*(epsilon/(2*error))**0.25.
if __name__ == '__main__':
    f = int(input().strip())
    epsilon = float(input().strip())
    a = float(input().strip())
    y_a = float(input().strip())
    b = float(input().strip())
    result = Result.solveByRungeKutta(f, epsilon, a, y_a, b)
    print(str(result) + '\n')
```

Примеры и результаты работы программы:

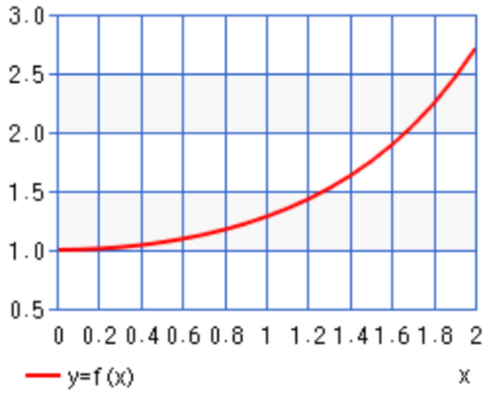
Пример 1: f=sin(x), epsilon = 0.001, x=0 , f(x) =1, f(0.5)=?

```
1
0.001
0
1
0.5
1.1224174399374478
```



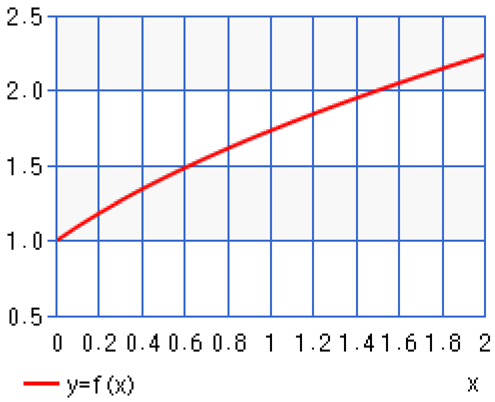
Пример 2: f= (x * y)/2, epsilon = 0.001, x=0 , f(x) =1, f(0.5)=?

```
2
0.001
0
1
0.5
1.0644944585186666
```



Пример 3: f= y - (2 * x)/y, epsilon = 0.001, x=0 , f(x) =1, f(0.5)=?

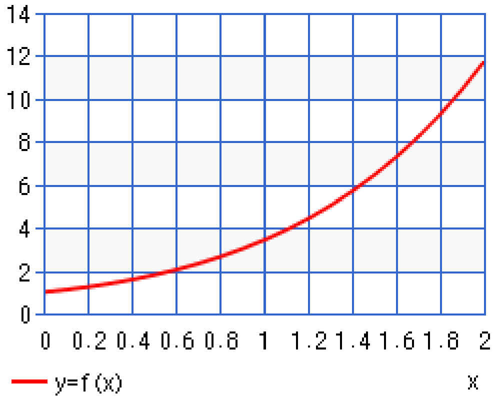
```
3
0.001
0
1
0.5
1.4142143040925295
```



Пример 4**:** f= x + y, epsilon = 0.001, x=0 , f(x) =1,

f(0.5)=?

```
4
0.001
0
1
0.5
1.7974422786887825
```



Вывод:
Преимущества:

- Метод Рунге-Кутты — очень точный численный метод решения дифференциальных уравнений.

- Это универсальный метод, который можно использовать для решения широкого круга дифференциальных уравнений, в том числе жестких или со сложными граничными условиями.

- Метод относительно прост в реализации и может быть адаптирован для решения задач различной сложности.

Минусы:

- Метод Рунге-Кутты может быть дорогостоящим в вычислительном отношении, особенно для методов более высокого порядка или для задач со многими временными шагами.

- Метод также может быть чувствителен к выбору размера шага и других параметров, что может повлиять на точность и стабильность решения.

- В некоторых случаях метод может быть не самым эффективным или точным методом решения конкретного дифференциального уравнения, и другие методы могут оказаться более подходящими.