Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»
Факультет Программной Инженерии и Компьютерной Техники

Вариант №9
Лабораторная работа №6
по теме
Численное решение обыкновенных дифференциальных уравнений
по дисциплине
Вычислительная математика

Выполнил Студент группы P3212
Кобелев Р.П.
к. т. н. Преподаватель:
Наумова Н.А.

г. Санкт-Петербург
2024г.

Содержание

1 Цель работы

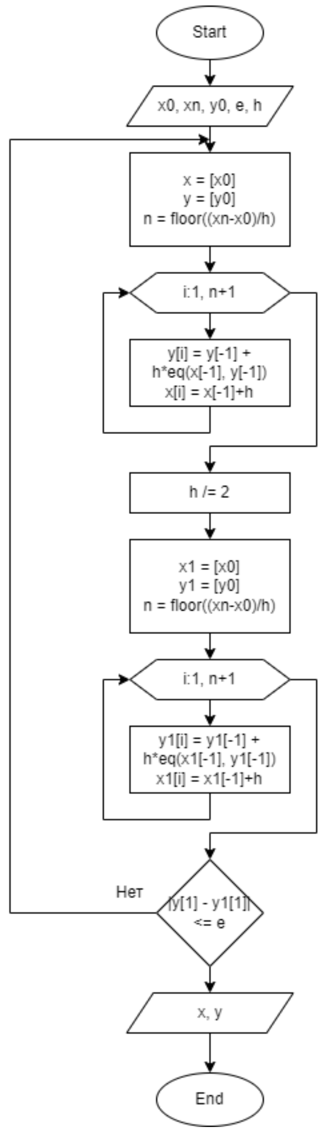
Решить задачу интерполяции, найти значения функ- ции при заданных значениях аргумента, отличных от узловых точек.

2 Задание

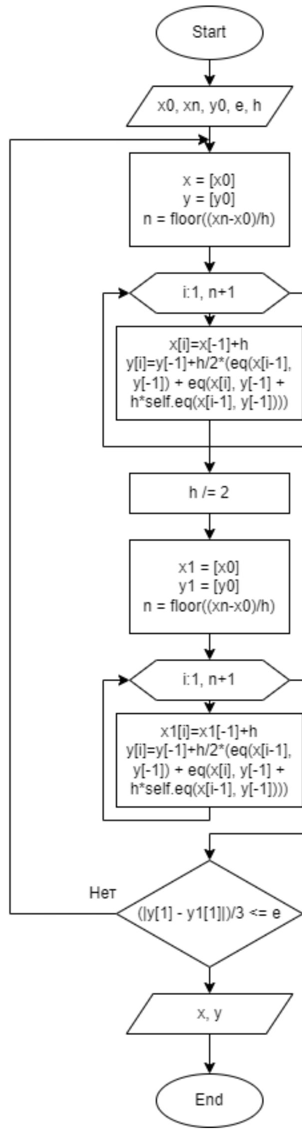
2.1 Порядок выполнения работы

1. В программе численные методы решения обыкновенных дифференциальных уравнений (ОДУ) должен быть реализован в виде отдельного класса /метода/функции;
2. Пользователь выбирает ОДУ вида $y' = f(x, y)$ (не менее трех уравнений), из тех, которые предлагает программа;
3. Предусмотреть ввод исходных данных с клавиатуры: начальные условия $y_0 = y(x_0)$, интервал дифференцирования $[x_0, x_n]$, шаг h, точность ε;
4. Для исследования использовать одношаговые методы и многошаговые методы (см. табл.1);
5. Составить таблицу приближенных значений интеграла дифференциального уравнения, удовлетворяющего начальным условиям, для всех методов, реализуемых в программе;
6. Для оценки точности одношаговых методов использовать правило Рунге;
7. Для оценки точности многошаговых методов использовать точное решение задачи: $\varepsilon = \max_{0 \leq i \leq n} |y_{i\text{точн}} - y_i|$
8. Построить графики точного решения и полученного приближенного решения (разными цветами);
9. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных.
10. Проанализировать результаты работы программы.

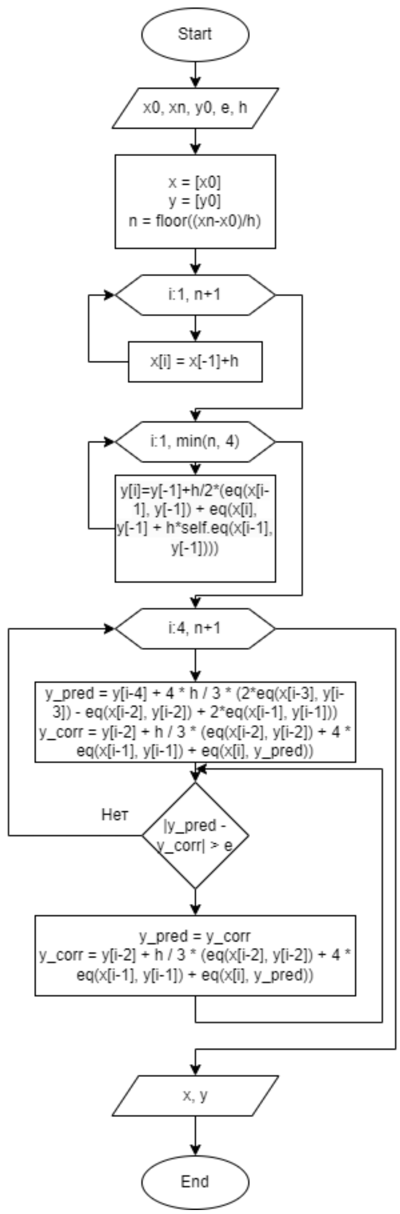
3 Программная реализация задачи



Эйлер



Усоверешнствованный Эйлер



Милн

3.1 Листинг программы

DifferentialEquations.py

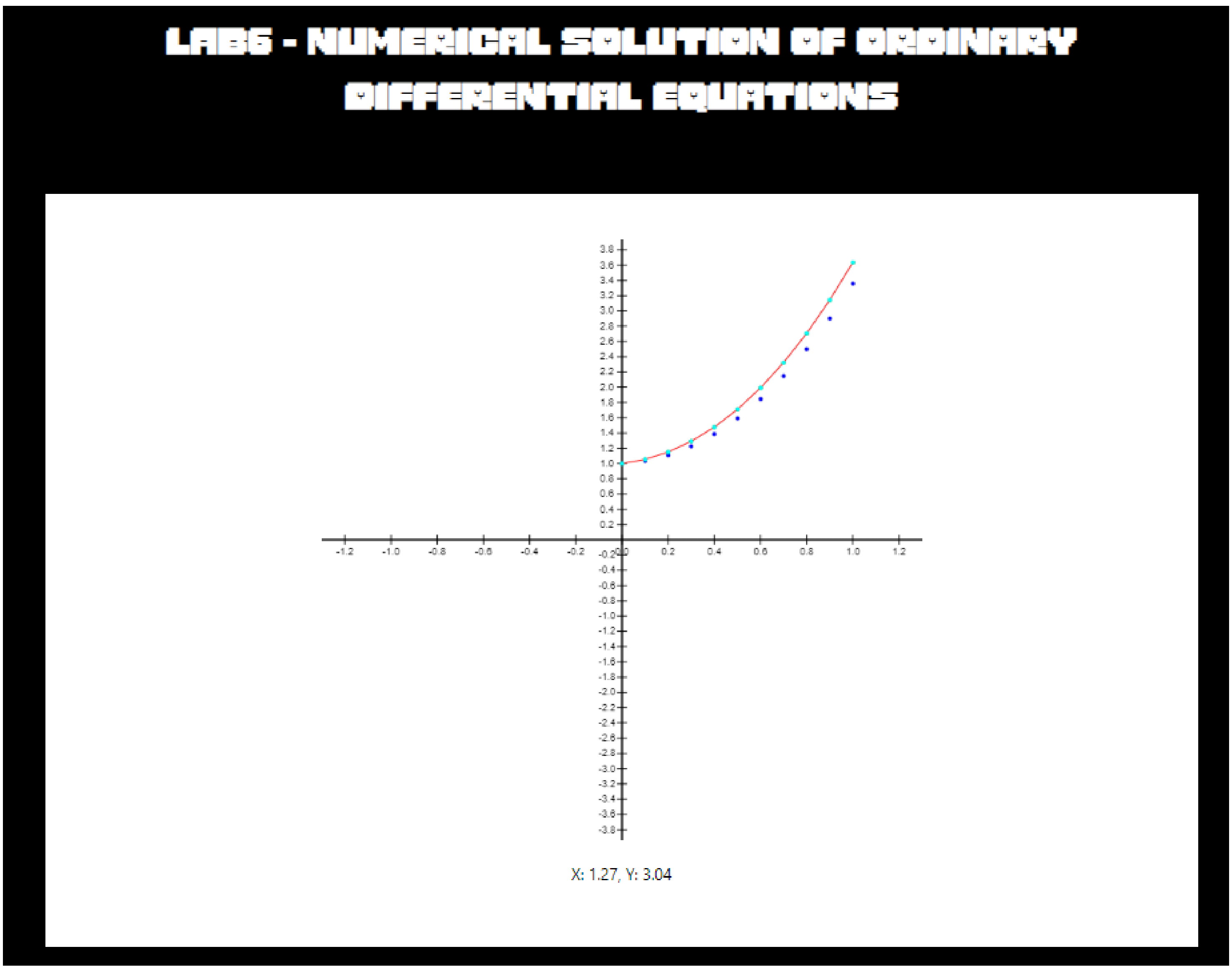
```
from dataclasses import dataclass
import numpy as np


def f1(x ,y):
  return 4*x + y/3
def fy1(x, c):
  return c*np.exp(x/3) - 12*x - 36
def c1(x, y):
  return (y + 12*x + 36)/np.exp(x/3)


def f2(x, y):
```

```python
    return x**2 + y
def fy2(x, c):
    return c * np.exp(x) - x**2 - 2 * x - 2
def c2(x, y):
    return (-y - x**2 - 2 * x - 2) / (-np.exp(x))

def f3(x ,y):
    return y * np.cos(x)
def fy3(x, c):
    return c * np.exp(np.sin(x))
def c3(x, y):
    return y /np.exp(np.sin(x))

@dataclass
class Differential:
    eq_num: int
    x0: int
    xn: int
    y0: int
    e: int
    h: int
    eq = None
    fy = None
    c = None
    def init(self):
        match self.eq_num:
            case 1:
                self.eq = f1
                self.fy = fy1
                self.c = c1(self.x0, self.y0)
            case 2:
                self.eq = f2
                self.fy = fy2
                self.c = c2(self.x0, self.y0)
            case _:
                self.eq = f3
                self.fy = fy3
                self.c = c3(self.x0, self.y0)


    def Direct(self):
        n = int((self.xn-self.x0)/self.h)
        x = [self.x0]
        y = [self.y0]
        for i in range(1, n+1):
            x.append(x[-1]+self.h)


        for i in range(1, n+1):
            y.append(self.fy(x[i], self.c))

        return [["{:.3f}".format(x[i]), "{:.3f}".format(y[i])] for i in range(len(x))]

    def Euler(self):
        h = self.h
        quitloop = True
```

```python
        while quitloop == True:
            x = [self.x0]
            y = [self.y0]
            n = int((self.xn-self.x0)/h)
            for i in range(1, n+1):
                y.append(y[-1] + h*self.eq(x[-1], y[-1]))
                x.append(x[-1]+h)

            h /=2
            x1 = [self.x0]
            y1 = [self.y0]
            n = int((self.xn-self.x0)/h)
            for i in range(1, n+1):
                y1.append(y1[-1] + h*self.eq(x1[-1], y1[-1]))
                x1.append(x1[-1]+h)
            if abs(y[1] - y1[1]) <= self.e:
                quitloop = False


        return [["{:.3f}".format(x[i]), "{:.3f}".format(y[i])] for i in range(len(x))]

    def ExtendedEuler(self):
        h = self.h
        quitloop = True
        while quitloop == True:
            x = [self.x0]
            y = [self.y0]
            n = int((self.xn-self.x0)/h)
            for i in range(1, n+1):
                x.append(x[-1]+h)
                y.append(y[-1]+h/2*(self.eq(x[-2], y[-1]) + self.eq(x[-1], y[-1] + h*self.eq(x[-2], y[-1]))))

            h /=2
            x1 = [self.x0]
            y1 = [self.y0]
            n = int((self.xn-self.x0)/h)
            for i in range(1, n+1):
                x1.append(x1[-1]+h)
                y1.append(y1[-1]+h/2*(self.eq(x1[-2], y1[-1]) + self.eq(x1[-1], y1[-1] + h*self.eq(x1[-2], y1[-1]))))

            if abs(y[1] - y1[1])/3 <= self.e:
                quitloop = False


        return [["{:.3f}".format(x[i]), "{:.3f}".format(y[i])] for i in range(len(x))]

    def Milne(self):
        x = [self.x0]
        y = [self.y0]
        n = int((self.xn-self.x0)/self.h)
        for i in range(1, n+1):
            x.append(x[-1]+self.h)

        for i in range(1, min(n, 4)):
```

```
      y.append(y[-1]+self.h/2*(self.eq(x[i-1], y[-1]) + self.eq(x[i], y[-1] + self.h*self.eq(x[i-1], y[-1]))))

    for i in range(4, n+1):
      y_pred = y[i-4] + 4 * self.h / 3 * (2*self.eq(x[i-3], y[i-3]) - self.eq(x[i-2], y[i-2]) + 2*self.eq(x[i-1], y[i-1]))
      y_corr = y[i-2] + self.h / 3 * (self.eq(x[i-2], y[i-2]) + 4 * self.eq(x[i-1], y[i-1]) + self.eq(x[i], y_pred))
      while abs(y_pred - y_corr) > self.e:
        y_pred = y_corr
        y_corr = y[i-2] + self.h / 3 * (self.eq(x[i-2], y[i-2]) + 4 * self.eq(x[i-1], y[i-1]) + self.eq(x[i], y_pred))
      y.append(y_corr)

    return [["{:.3f}".format(x[i]), "{:.3f}".format(y[i])] for i in range(len(x))]
```

3.2 Пример работы программы



$$1)4 \cdot x + \frac{y}{3} \qquad 2)x^2 + y \qquad 3)y \cdot \cos x$$

1

0

1

1

0.1

0.1

SUBMIT

Метод Эйлера

X	Y
0.000	1.000
0.100	1.033
0.200	1.108
0.300	1.225
0.400	1.386
0.500	1.592
0.600	1.845
0.700	2.146
0.800	2.498
0.900	2.901
1.000	3.358

Усовершенствованный метод Эйлера

X	Y
0.000	1.000
0.100	1.054
0.200	1.150
0.300	1.291
0.400	1.476
0.500	1.709
0.600	1.990
0.700	2.322
0.800	2.705
0.900	3.142
1.000	3.635

Метод Милна

X	Y
0.000	1.000
0.100	1.054
0.200	1.150
0.300	1.291
0.400	1.476
0.500	1.709
0.600	1.990
0.700	2.322
0.800	2.705
0.900	3.142
1.000	3.635

4 Github

Ссылка на GitHub

5 Вывод

В этой работе я ознакомился с различными методами решения обычных дифференциальных уравнений