Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

**Отчет по лабораторной работе №6**

**по дисциплине «Вычислительная математика»**

**Вариант 11**

**Выполнили**:

Студенты группы P3265

Кручинина Дарья Сергеевна

Москвитина Полина

**Преподаватель:**

Машина Екатерина Алексеевна

**Цели работы**

Цель лабораторной работы: решить задачу Коши для обыкновенных

дифференциальных уравнений численными методами.

**Порядок выполнения работы**

2. В программе численные методы решения обыкновенных

дифференциальных уравнений (ОДУ) должен быть реализован в

виде отдельного класса /метода/функции;

3. Пользователь выбирает ОДУ вида (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 (см. табл.1);

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.

**Для 11 варианта в программе использовали методы: Усовершенствованный метод Эйлера, Метод Рунге-Кутта 4- го порядка и Адамса**

**Листинг программы**

from matplotlib import pyplot as plt  
from math import exp

def modified\_euler\_method(f, xs, y0):  
 ys = [y0]  
 h = xs[1] - xs[0]  
 for i in range(1, len(xs)):  
 k1 = f(xs[i - 1], ys[i - 1])  
 k2 = f(xs[i], ys[i - 1] + h \* k1)  
 ys.append(ys[i - 1] + h \* 0.5 \* (k1 + k2))  
 return ys

def fourth\_order\_runge\_kutta\_method(f, xs, y0):  
 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

def adams\_method(f, xs, y0):  
 ys = fourth\_order\_runge\_kutta\_method(f, xs[:4], y0)  
 h = xs[1] - xs[0]  
  
 for i in range(4, len(xs)):  
 df = f(xs[i - 1], ys[i - 1]) - f(xs[i - 2], ys[i - 2])  
 d2f = f(xs[i - 1], ys[i - 1]) - 2 \* f(xs[i - 2], ys[i - 2]) + f(xs[i - 3], ys[i - 3])  
 d3f = f(xs[i - 1], ys[i - 1]) - 3 \* f(xs[i - 2], ys[i - 2]) + 3 \* f(xs[i - 3], ys[i - 3]) - f(xs[i - 4], ys[i - 4])  
 y = ys[i - 1] + h \* f(xs[i - 1], ys[i - 1]) + h \*\* 2 / 2 \* df + 5 \* h \*\* 3 / 12 \* d2f + 3 \* h \*\* 4 / 8 \* d3f  
 ys.append(y)  
 return ys

def draw\_plot(a, b, func, dx=0.01):  
 xs, ys = [], []  
 a -= dx  
 b += dx  
 x = a  
 while x <= b:  
 xs.append(x)  
 ys.append(func(x))  
 x += dx  
 plt.plot(xs, ys, 'g')

def main(f, xs, y0, exact\_y):  
 methods = [modified\_euler\_method,  
 fourth\_order\_runge\_kutta\_method,  
 adams\_method]  
 for method in methods:  
 print(method.\_\_name\_\_)  
  
 ys = method(f, xs, y0)  
 if method in (adams\_method,):  
 inaccuracy = max([abs(exact\_y(x) - y)  
 for x, y in zip(xs, ys)])  
 else:  
 xs2 = []  
 for x1, x2 in zip(xs, xs[1:]):  
 xs2.extend([x1, (x1 + x2) / 2, x2])  
 ys2 = method(f, xs2, y0)  
 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("ys:", \*map(lambda x: round(x, 5), ys))  
 print(f"inaccuracy = {inaccuracy}")  
  
 plt.title(method.\_\_name\_\_)  
  
 draw\_plot(xs[0], xs[-1], exact\_y)  
 for i in range(len(xs)):  
 plt.scatter(xs[i], ys[i], c='r')  
 plt.xlabel("X")  
 plt.ylabel("Y")  
 plt.show()

def read\_number(s: str):  
 while True:  
 try:  
 return float(input(s))  
 except Exception:  
 continue

if \_\_name\_\_ == '\_\_main\_\_':  
 print("1. y' = x \*\* 2")  
 print("2. y' = 2 \* x \*\* 2 + 2")  
 print("3. y' = x")  
 mode = read\_number("Выберите функцию: ")  
 n = read\_number("Введите n: ")  
 x0 = read\_number("Введите x0: ")  
 xn = read\_number("Введите xn: ")  
 h = (xn - x0) / n  
 xs = [x0 + h \* i for i in range(int(n))]  
 try:  
 if mode == 1:  
 f = lambda x, y: x \*\* 2  
 y0 = 1  
 exact\_y = lambda x: x \*\* 3 / 3 + 1  
 elif mode == 2:  
 f = lambda x, y: 2 \* x \*\* 2 + 2  
 y0 = 6  
 exact\_y = lambda x: 2 \* x \*\* 3 / 3 + 2 \* x + 6  
 elif mode == 3:  
 f = lambda x, y: x  
 y0 = 1  
 exact\_y = lambda x: x \*\* 2 / 2 + 1  
 except (ZeroDivisionError, ArithmeticError) as e:  
 print("Функция не определена")  
  
main(f, xs, y0, exact\_y)

**Вывод программы**







