Федеральное государственное автономное

образовательное учреждение высшего образования

Национальный Исследовательский Университет ИТМО

Лабораторная работа 6

«Численное решение обыкновенных дифференциальных уравнений»

Дисциплина: Вычислительная математика

Вариант 13

Выполнил: Терехин Никита Денисович

Факультет: Программной инженерии и компьютерной техники

Группа: P3208

Преподаватель: Машина Екатерина Алексеевна

г. Санкт-Петербург, 2024 год

Оглавление

Цель работы

Решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

Текст задания

1. В программе численные методы решения обыкновенных дифференциальных уравнений (ОДУ) должен быть реализован в виде отдельного класса / метода / функции;
2. Пользователь выбирает ОДУ вида (не менее трех уравнений), из тех, которые предлагает программа;
3. Предусмотреть ввод исходных данных с клавиатуры: начальные условия , интервал дифференцирования , шаг h, точность;
4. Для исследования использовать одношаговые методы и многошаговые методы;
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: $\varepsilon = \max\limits_{0 \le i \le n} |y_{i\ \text{точн}} - y_i|$;
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.

Рабочие формулы методов

Формула Эйлера:

$$y_{i+1} = y_i + h \cdot f(x_i, y_i)$$

Модификация формулы Эйлера:

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

$$\widehat{y}_{i+1} = y_i + h \cdot f(x_i, y_i)$$

Формула Рунге-Кутта 4 порядка:

$$y_{i+1} = y_i + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$
$$k_1 = h \cdot f(x_i, y_i)$$
$$k_2 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_1}{2})$$
$$k_3 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_2}{2})$$
$$k_4 = h \cdot f(x_i + h, y_i + k_3)$$

Листинг программы

```
# main.py

import math

from matplotlib import pyplot as plt
from tabulate import tabulate

from P3208.Terekhin_367558.lab2.functions import
DifferentialEquation, EQUATIONS
from P3208.Terekhin_367558.lab2.main import request_from_list
from P3208.Terekhin_367558.lab2.readers import ConsoleReader
from P3208.Terekhin_367558.lab6.differential import DIFFERENTIALS

if __name__ == '__main__':
   equation: DifferentialEquation = request_from_list(EQUATIONS)
   reader: ConsoleReader = ConsoleReader('From console')
   init_y = reader.read_argument('Enter initial y: ')
   h = reader.read_argument('Enter step size h: ')
   a, b, eps = reader.read_tuple('Enter differential interval
using two numbers: ')

   a, b = min(a, b), max(a, b)

   if equation.c is None:
       equation.c = equation.const(a, init_y)

   x_range = [i / 100 for i in range(math.floor(a * 100),
math.ceil(b * 100))]
   y_range = [equation.solution(k, equation.c) for k in x_range]

   colors = ['red', 'green', 'orange', 'purple', 'cyan',
'magenta', 'pink', 'yellow', 'brown']
   color_index: int = 0

   plt.plot(x_range, y_range)
```

```
    for diff in DIFFERENTIALS:
        diff.set_data(a, b, h, init_y)
        y = diff.solve(equation, eps)
        y_accurate = [equation.solution(k, equation.c) for k in
diff.x]
        ans = []
        for i in range(len(y_accurate)):
            ans.append(list(map(lambda k: round(k, 4), [diff.x[i],
y[i], y_accurate[i]])))

        print(tabulate([diff.description]))
        print(tabulate(ans, tablefmt='pretty', headers=['x', 'y',
'accurate_answer']))

        plt.plot(diff.x, diff.y, color=colors[color_index %
len(colors)])
        color_index += 1
    plt.show()
```

```
# differential.py

from abc import abstractmethod
from typing import Final

from P3208.Terekhin_367558.lab2.functions import Describable,
DifferentialEquation


class Differential(Describable):
    def __init__(self, description: str):
        super().__init__(description)
        self.x: list[float] = []
        self.y: list[float] = []
        self.y_new: list[float] = []
        self.h: float = 0

    def set_data(self, a: float, b: float, h: float, init_y:
float):
        self.x = []
        c: float = a
        while c < b:
            self.x.append(c)
            c += h / 2
            if c >= b:
                break
            c += h / 2
        self.x.append(b)
```

```
        self.y = [init_y] + [0] * (len(self.x) - 1)
        self.y_new = [init_y] + [0] * (len(self.x) - 1)
        self.h = h

    @abstractmethod
    def solve(self, equation: DifferentialEquation, eps: float):
        pass


class EulerDifferential(Differential):
    def __init__(self):
        super().__init__('Euler Method')

    def solve(self, equation: DifferentialEquation, eps: float) ->
list[float]:
        self.set_data(self.x[0], self.x[-1], self.h, self.y[0])
        n: int = len(self.x)
        for i in range(1, n):
            self.y[i] = self.y[i - 1] + self.h *
equation.function(self.x[i - 1], self.y[i - 1])
            self.y_new[i] = self.y[i] + self.h *
equation.function((self.x[i] + self.x[i - 1]) / 2, self.y[i])
            if i == (n - 1) and abs(self.y[i] - self.y_new[i]) >=
eps:
                self.h /= 2
                return self.solve(equation, eps)
        return self.y


class ModifiedEulerDifferential(Differential):
    def __init__(self):
        super().__init__('Modified Euler Method')

    def solve(self, equation: DifferentialEquation, eps: float) ->
list[float]:
        self.set_data(self.x[0], self.x[-1], self.h, self.y[0])
        n: int = len(self.x)
        for i in range(1, n):
            _y = self.y[i - 1] + self.h *
equation.function(self.x[i - 1], self.y[i - 1])
            self.y[i] = self.y[i - 1] + self.h / 2 *
(equation.function(self.x[i - 1], self.y[i - 1])
                                                     +
equation.function(self.x[i], _y))
            _y_new = self.y[i] + self.h *
equation.function((self.x[i] + self.x[i - 1]) / 2, self.y[i])
            self.y_new[i] = self.y[i] + self.h / 2 *
(equation.function((self.x[i] + self.x[i - 1]) / 2, self.y[i])
```

```
                                                                +
equation.function(self.x[i], _y_new))
            if i == (n - 1) and (abs(self.y[i] - self.y_new[i]) /
3) >= eps:
                self.h /= 2
                return self.solve(equation, eps)
        return self.y


class MilneDifferential(Differential):
    def __init__(self):
        super().__init__('Milne\'s Method')

    def solve(self, equation, eps):
        for i in range(1, 5):
            _y = self.y[i - 1] + self.h *
equation.function(self.x[i - 1], self.y[i - 1])
            self.y[i] = self.y[i - 1] + self.h / 2 *
(equation.function(self.x[i - 1], self.y[i - 1])
                                                                +
equation.function(self.x[i], _y))
        n = len(self.x)
        for i in range(4, n):
            self.y[i] = self.y[i - 4] + (4 * self.h / 3) * (2 *
equation.function(self.x[i - 3], self.y[i - 3])
                                                                      -
equation.function(self.x[i - 2], self.y[i - 2])
                                                                      + 2 *
equation.function(self.x[i - 1], self.y[i - 1]))
            step = 0
            while abs(self.y[i] - equation.solution(self.x[i],
equation.c)) >= eps and step < 1000:
                self.y[i] = self.y[i - 2] + (self.h / 3) *
(equation.function(self.x[i - 2], self.y[i - 2])
                                                                      - 4 *
equation.function(self.x[i - 1], self.y[i - 1])
                                                                      +
equation.function(self.x[i], self.y[i]))
                step += 1
        return self.y


DIFFERENTIALS: Final[list[Differential]] = [
    EulerDifferential(),
    ModifiedEulerDifferential(),
    MilneDifferential()
]
```

Результаты работы программы

```
1. y' = 0,7 + 2y + 1,1x^2
2. y'= 4x + y/3
3. y' = y + (1 + x)y^2
4. y' = 2xy / (x^2 + 1)
Choose equation:
char
No such equation. Try again
9
No such equation. Try again
3
Enter initial y:
char
Should be a float number. Try again:
-1
Enter step size h:
0.1
Enter differential interval using two numbers:
3
not enough values to unpack (expected 2, got 1)
Try again:
1 1.5
Input precision:
4
Precision is a positive float less than 1
Try again:
0.01
-  -  -  -  -    -  -  -  -  -  -
E  u  l  e  r    M  e  t  h  o  d
-  -  -  -  -    -  -  -  -  -  -
+--------+---------+-----------------+
|   x    |    y    | accurate_answer |
+--------+---------+-----------------+
|  1.0   |  -1.0   |      -1.0       |
| 1.0125 | -0.9875 |     -0.9877     |
| 1.025  | -0.9753 |     -0.9756     |
| 1.0375 | -0.9634 |     -0.9639     |
|  1.05  | -0.9518 |     -0.9524     |
| 1.0625 | -0.9405 |     -0.9412     |
| 1.075  | -0.9295 |     -0.9302     |
| 1.0875 | -0.9187 |     -0.9195     |
|  1.1   | -0.9081 |     -0.9091     |
| 1.1125 | -0.8978 |     -0.8989     |
| 1.125  | -0.8878 |     -0.8889     |
| 1.1375 | -0.8779 |     -0.8791     |
|  1.15  | -0.8683 |     -0.8696     |
| 1.1625 | -0.8589 |     -0.8602     |
```

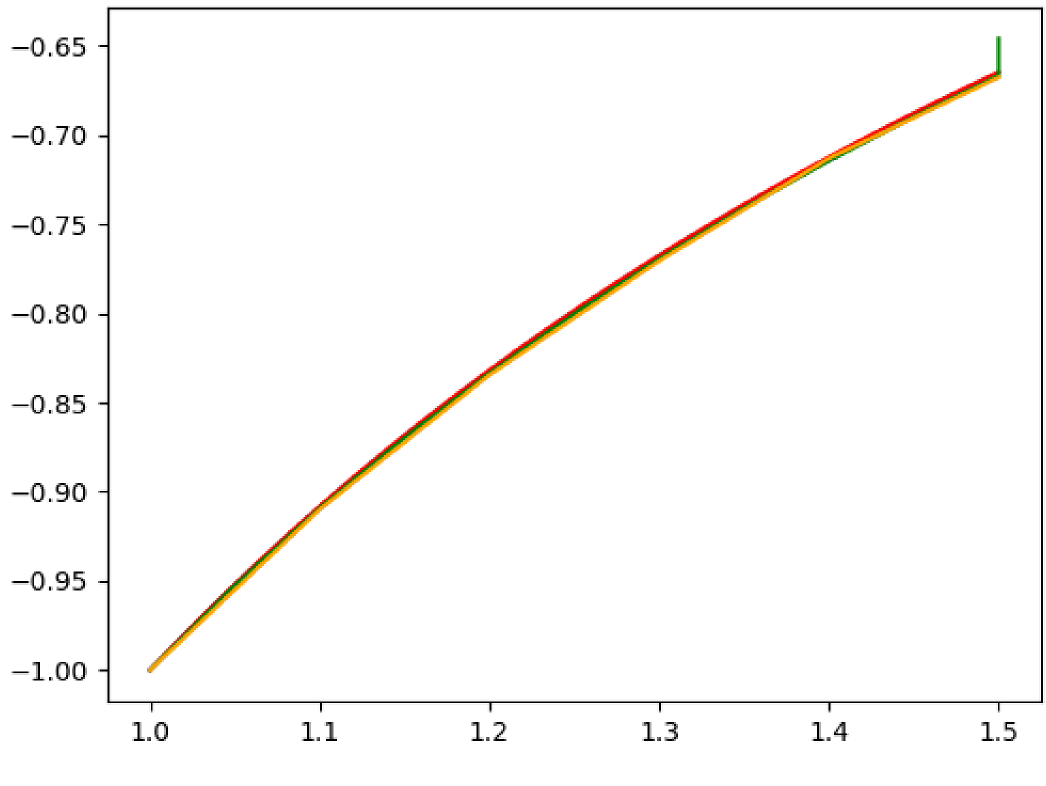
1.175	-0.8497	-0.8511
1.1875	-0.8407	-0.8421
1.2	-0.8319	-0.8333
1.2125	-0.8232	-0.8247
1.225	-0.8148	-0.8163
1.2375	-0.8065	-0.8081
1.25	-0.7984	-0.8
1.2625	-0.7905	-0.7921
1.275	-0.7827	-0.7843
1.2875	-0.775	-0.7767
1.3	-0.7675	-0.7692
1.3125	-0.7602	-0.7619
1.325	-0.753	-0.7547
1.3375	-0.7459	-0.7477
1.35	-0.739	-0.7407
1.3625	-0.7322	-0.7339
1.375	-0.7255	-0.7273
1.3875	-0.719	-0.7207
1.4	-0.7125	-0.7143
1.4125	-0.7062	-0.708
1.425	-0.7	-0.7018
1.4375	-0.6939	-0.6957
1.45	-0.6879	-0.6897
1.4625	-0.682	-0.6838
1.475	-0.6762	-0.678
1.4875	-0.6705	-0.6723
1.5	-0.6649	-0.6667

M o d i f i e d E u l e r M e t h o d

x	y	accurate_answer
1.0	-1.0	-1.0
1.05	-0.9525	-0.9524
1.1	-0.9093	-0.9091
1.15	-0.8698	-0.8696
1.2	-0.8336	-0.8333
1.25	-0.8003	-0.8
1.3	-0.7696	-0.7692
1.35	-0.7411	-0.7407
1.4	-0.7146	-0.7143
1.45	-0.69	-0.6897
1.5	-0.667	-0.6667
1.5	-0.646	-0.6667

```
M i l n e ' s   M e t h o d
- - - - - - -   - - - - - -
+-----+---------+-----------------+
|  x  |    y    | accurate_answer |
+-----+---------+-----------------+
| 1.0 |  -1.0   |       -1.0      |
| 1.1 |  -0.91  |     -0.9091     |
| 1.2 | -0.8346 |     -0.8333     |
| 1.3 | -0.7707 |     -0.7692     |
| 1.4 | -0.7132 |     -0.7143     |
| 1.5 | -0.6679 |     -0.6667     |
+-----+---------+-----------------+

Process finished with exit code 0
```



Выводы

В результате выполнения работы я научился решать задачу Коши для обыкновенных дифференциальных уравнений численными методами.
В программе были реализованы методы Эйлера, Эйлера с модификациями, а также многошаговый метод Милна

В конечном итоге получилось, что самым точным является модифицированный метод Эйлера, в то время как метод Милна на некоторых входных данных ведет себя неоднозначно