Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Факультет программной инженерии и компьютерной техники

Дисциплина: **Вычислительная математика**

Лабораторная работа №6

" «Численное дифференцирование»"

Вариант: 9

Выполнил: Кузнецов Максим Александрович

Группа: P3111

Преподаватель: Малышева Татьяна Алексеевна

Санкт-Петербург 2022 г

Цель лабораторной работы: решить задачу Коши численными методами. Написать программную реализацию данных методов по варианту.

Для исследования использовать:

- Одношаговые методы;
- Многошаговые методы.

Условия:

Программная реализация задачи:

1. Исходные данные: ОДУ вида $y' = f(x, y)$, начальные условия $y(x_0)$, интервал дифференцирования $[a, b]$, шаг h, точность ε.
2. Составить таблицу приближенных значений интеграла дифференциального уравнения, удовлетворяющего начальным условиям. Для оценки точности использовать правило Рунге.
3. Построить графики точного решения и полученного численного решения (разными цветами).

4. Анализ результатов работы: апробация и тестирование.

Описание методов:

Методы:

Одношаговые методы:

1. Метод Эйлера;
2. Усовершенствованный метод Эйлера;
3. Метод Рунге-Кутта 4- го порядка.

Многошаговые методы:

4. Адамса;
5. Милна.

№ варианта	Метод	№ варианта	Метод
1	1, 4	16	1, 5

2	2, 5	17	2, 4
3	3, 5	18	3, 4
4	2, 4	19	1, 4
5	3, 4	20	2, 5
6	1, 5	21	1, 4
7	2, 4	22	1, 5
8	3, 4	23	2, 4
9	2, 5	24	3, 4
10	1, 5	25	1, 5
11	2, 4	26	2, 4
12	3, 4	27	1, 4
13	2, 5	28	3, 5
14	3, 5	29	2, 5
15	1, 4	30	1, 4

В программе рассматриваются методы по 9-му варианту: Модифицированный метод Эйлера и метод Милна.

Модификации метода Эйлера.

Рассмотрим уравнение (2) в окрестностях узлов $x = x_i + h/2$ (i=0, 1…), являющихся серединами отрезков $[x_i, x_{i+1}]$. В левой части (2) заменим производную центральной разностью, а в правой части заменим значение функции $f(x_i + h/2, Y(x_i + h/2))$ средним арифметическим значений функции $f(x, Y)$ в точках (x_i, y_i) и (x_{i+1}, y_{i+1}). Тогда:

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

Отсюда:

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i, y_i) + f(x_{i+1}, y_{i+1})]. \qquad (9)$$

Полученная схема является неявной, поскольку искомое значение y_{i+1} входит в обе части соотношения (9) и его нельзя выразить явно. Для вычисления y_{i+1} можно применить один из итерационных методов. Если имеется хорошее начальное приближение y_i, то можно построить решение с использованием двух итераций следующим образом. Считая y_i начальным приближением, вычисляем первое приближение $\tilde{y}_{i+1}$ по формуле метода Эйлера (7):

$$\tilde{y}_{i+1} = y_i + hf(x_i, y_i)$$

Вычисленное значение $\tilde{y}_{i+1}$ подставляем вместо y_{i+1} в правую часть соотношения (9) и находим окончательное значение:

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

$$\boldsymbol{y_{i+1} = y_i + \frac{h}{2}\big[f(x_i, y_i) + f\big(x_{i+1}, y_i + hf(x_i, y_i)\big)\big],\ i = 0, 1 \ldots} \qquad (10)$$

Данные рекуррентные соотношения описывают новую разностную схему, являющуюся ***модифицированным методом Эйлера***, которая называется методом ***Эйлера с пересчетом***. Метод Эйлера с пересчетом имеет ***второй порядок точности*** $\delta_n = O(h^2)$.

Метод Милна

Метод Милна относится к многошаговым методам и представляет один из методов прогноза и коррекции.

Для получения формул Милна используется первая интерполяционная формула Ньютона с разностями до третьего порядка.

Решение в следующей точке находится в два этапа. На первом этапе осуществляется прогноз значения функции, а затем на втором этапе - коррекция полученного значения. Если полученное значение y после коррекции существенно отличается от спрогнозированного, то проводят еще один этап коррекции. Если опять имеет место существенное отличие от предыдущего значения (т.е. от предыдущей коррекции), то проводят еще одну коррекцию и т.д.

Вычислительные формулы:

а) этап прогноза:

$$y_i^{\text{прогн}} = y_{i-4} + \frac{4h}{3}(2f_{i-3} - f_{i-2} + 2f_{i-1})$$

б) этап коррекции:

$$y_i^{\text{корр}} = y_{i-2} + \frac{h}{3}(f_{i-2} + 4f_{i-1} + f_i^{\text{прогн}})$$
$$f_i^{\text{прогн}} = f(x_i, y_i^{\text{прогн}})$$

Для начала счета требуется задать решения в трех первых точках, которые можно получить одношаговыми методами (например, методом Рунге-Кутта).

Листинг программы:

```
import numpy as np
from matplotlib import pyplot as plt
import math
import sys
import sympy
X, Y = sympy.symbols("x,y")

def show(x, y, desc=""):
  plt.plot(x,y,'o')
  plt.xlabel("Value of x")
  plt.ylabel("Value of y")
  plt.title(desc)
  plt.show()


# Методом мод. Эйлера
def euler_modified(eq, a, b, x0, y0, h):
    n = int(math.ceil((b - a) / h))
    equation = str(eq)
    result: list[(float, float)] = [(x0, y0)]

    for i in range(n+1):
        xi = float(result[i][0])
        yi = float(result[i][1])
        xi1 = float(xi + h)
        f1 = get_val(equation, xi, yi)
```

```python
        f2 = get_val(equation, xi1, yi + h * f1)
        yi1 = float(yi + h / 2 * (f1 + f2))
        result.append((xi1, yi1))
    return result


# Методом Мильна
def milne(eq, a, b, x0, y0, h, eps):
    n: int = math.ceil((b - a) / h)
    result, _, _ = runge_rule(eq, a, b, x0, y0, h, eps)
    plotX = []
    plotYMilne = []
    for i in range(3, n+1):
        xi = result[i][0]
        xi1 = xi + h
        y3 = float(result[i - 3][1])
        f2 = float(get_val(eq, result[i - 2][0], result[i - 2][1]))
        f1 = get_val(eq, result[i - 1][0], result[i - 1][1])
        fi = get_val(eq, result[i][0], result[i][1])
        y_cur = float(y3 + 4 * h / 3 * (2 * f2 - f1 + 2 * fi))
        yi1 = float(result[i - 1][1])
        fi1r = float(get_val(eq, xi1, y_cur))
        y_prev = float(yi1 + h / 3 * (f1 + 4 * fi + fi1r))
        while abs(y_prev - y_cur) > eps:
            y_cur = y_prev
            fi1r = get_val(eq, xi1, y_cur)
            y_prev = yi1 + h / 3 * (f1 + 4 * fi + fi1r)
        result.append((xi1, y_prev))
    return result

def runge_rule(eq, a, b, x0, y0, h, eps):
    eps_cur = eps
    h_cur = h
    res = euler_modified(eq, a, b, x0, y0, h)
    r = 10000
    while r >= eps_cur:
        prev_res = res
        h /= 2
        res = euler_modified(eq, a, b, x0, y0, h)
        r = abs(res[0][1] - prev_res[0][1]) / 3
        for i in range(len(res[::2])):
            r = max(r, abs(res[::2][i][1] - prev_res[i][1]) / 3)

    result = [i for i in res[::int(h_cur / h)]]
    h, h_cur = h_cur, h
    return result, r, h_cur

def estimate(y_arr, y_arr2, p):
    print("Оценка погрешности:")
```

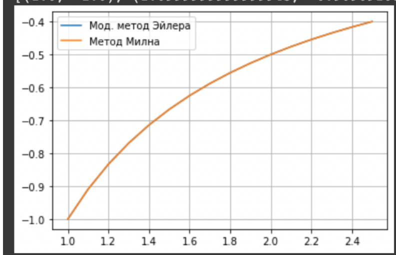
```
    print(runge_rule(y_arr, y_arr2, p))

# Преобразуем величины в sympy объект
def get_val(equation: str, x: float, y: float):
    return float(sympy.sympify(equation).evalf(subs={X: x, Y: y}))

def resize(array, new_size, new_value=0):
    """Resize to biggest or lesser size."""
    element_size = len(array[0]) #Quantity of new elements equals to
quantity of first element
    if new_size > len(array):
        new_size = new_size - 1
        while len(array)<=new_size:
            n = tuple(new_value for i in range(element_size))
            array.append(n)
    else:
        array = array[:new_size]
    return array
def run():
  print("Ур-ие:")
  while True:
    try:
      equation: str = input("y(x)' = ")
      get_val(equation, 1, 1)
      break
    except (TypeError):
      print("Введите снова!", file=sys.stderr)
  eq = equation
  print("Интервал дифференцирования (a и b):")
  while True:
    try:
      a, b = map(float, input().strip().split(" "))
      break
    except ValueError:
      print("Попробуйте еще раз!", file=sys.stderr)
  if a > b:
    a, b = b, a
  x0, a1, b1 = a, a, b
  while True:
    try:
      print("Введите y0:")
      print(f"y({x0}) = ")
      a = float(input().strip())
      break
    except ValueError:
      print("Попробуйте еще раз!", file=sys.stderr)
  y0 = a
  while True:
```

```python
  try:
    print("Введите h:")
    print(f"h = ")
    a = float(input().strip())
    break
  except ValueError:
    print("Попробуйте еще раз!", file=sys.stderr)
h = a
while True:
  try:
    print("Введите eps:")
    print(f"eps = ")
    a = float(input().strip())
    break
  except ValueError:
    print("Попробуйте еще раз!", file=sys.stderr)
eps = a
n: int = math.ceil((b1 - a1) / h)
print(n)
euler_result, _, _ = runge_rule(eq, a1, b1, x0, y0, h, eps)
milne_result = milne(eq, a1, b1, x0, y0, h, eps)
euler_result = euler_result[:n+1]
milne_result = milne_result[:n+1]
print(euler_result)
print(milne_result)
plotX = []
plotYEuler = []
plotYMilne = []
for index, tuple in enumerate(euler_result):
  plotX.append(tuple[0])
  plotYEuler.append(tuple[1])
for index, tuple in enumerate(milne_result):
  plotYMilne.append(tuple[1])
plt.figure()
plt.plot(plotX, plotYEuler, label="Мод. метод Эйлера")
plt.plot(plotX, plotYMilne, label="Метод Милна")
plt.grid(True)
plt.legend()
plt.show()
# out1 = 0
# out2 = 0
# step = h
# while 1:
#   euler_result = euler_modified(eq, a, b, x0, y0, step)
#   euler_result_2 = euler_modified(eq, a, b, x0, y0, step/2)
#   if(runge_rule(euler_result[0], euler_result_2[0], 3)<=eps):
#     out1 = euler_result_2
#     estimate(euler_result[0], euler_result_2[0], 3)
```

```
  #     break
  #   step = step/2
  # euler_results = out1
  # print(euler_results)
  # while 1:
  #   milne_result = milne(eq, a, b, x0, y0, step, eps)
  #   milne_result_2 = milne(eq, a, b, x0, y0, step/2, eps)
  #   if(runge_rule(milne_result[0], milne_result_2[0], 3)<=eps):
  #     out2 = milne_result_2
  #     estimate(milne_result[0], milne_result_2[0], 3)
  #     break
  #   step = step/2
  # milne_results = out2
  # print(milne_results)
  # while 1:
  #   euler_result = euler_modified(eq, a, b, x0, y0, h)
  #   euler_result_2 = euler_modified(eq, a, b, x0, y0, h/2)
  #   if(runge_rule(euler_result[0][1], euler_result_2[0][1], 3)<=eps):
  #     break
  #   step = step/2
  # show(x_euler, y_euler_2, py_euler, "Approximation Solution with
Modified Euler's Method",5)
  # estimate(y_euler, y_euler_2, 3)
  # print("Метод Мильна")
  # step = 1
  # while 1:
  #   if(runge_rule(y_milne, y_milne_2, 3)<=0.001):
  #     break
  #   step = step/2
  # show(x_milne, y_milne, py_milne, "Approximation Solution with
Milne's Method",5)
  # estimate(y_milne, y_milne_2, 3)
```

Пример работы:

```
Ур-ие:
y(x)' = y+(1+x)*y^2
Интервал дифференцирования (a и b):
1. 2.5
Введите y0:
y(1.0) =
-1
Введите h:
h =
0.1
Введите eps:
eps =
0.000001
15
[(1.0, -1.0), (1.0999999999999943, -0.9090910881323658), (1.1999999999999886, -0.8333336009569484), (1.299999999999983, -0.7692310747861445), (1.3999999999999773, -0.71428602
[(1.0, -1.0), (1.0999999999999943, -0.9090910881323658), (1.1999999999999886, -0.8333336009569484), (1.299999999999983, -0.7692310747861445), (1.3999999999999773, -0.71428602
```



<u>Примечание по коду:</u>

После ввода оценки точности eps на экран выводится количество точек, к которым будут даны значения (другими словами – длина исходного интервала, выраженная в количестве точек). Затем, в первой строке выводится пара значений Xi и Yi для метода мод. Эйлера, и во второй то же самое, только для метода Милна. В конце – график.

Вывод:

В результате выполнения лабораторной работы я:

- Познакомился с различными способами численного решения задачи Коши с помощью методов мод. Эйлера и Милна
- Написал программную реализацию данных методов на языке Python