Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04

Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №6

Вариант 11

Выполнил: Савельева Диана Александровна

P32082

Преподаватель: Машина Екатерина Алексеевна

Санкт-Петербург, 2023 г.

***1. Цель:*** решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

***2. Алгоритмы решения:***

* *Усовершенствованный метод Эйлера*

Рассчитать приближенное значение yi+1. Подставить в рабочую формулу метода.

* *Метод Рунге-Кутта 4 порядка*

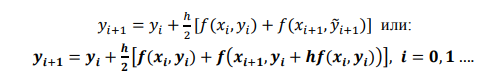
Рассчитать коэффициенты на каждом шаге. Подставить в рабочую формулу метода.

* *Метод Адамса*

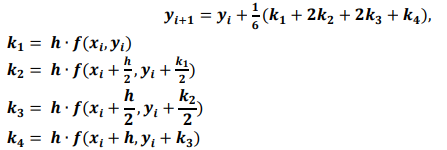
Рассчитать по формуле Рунге-Кутта 4 порядка первые 4 значения. Начиная с 5, рассчитывать в соответствии с рабочей формулой, значения.

***3. Рабочие формулы:***

* *Усовершенствованный метод Эйлера*

**

* *Метод Рунге-Кутта 4 порядка*

**

* *Метод Адамса*

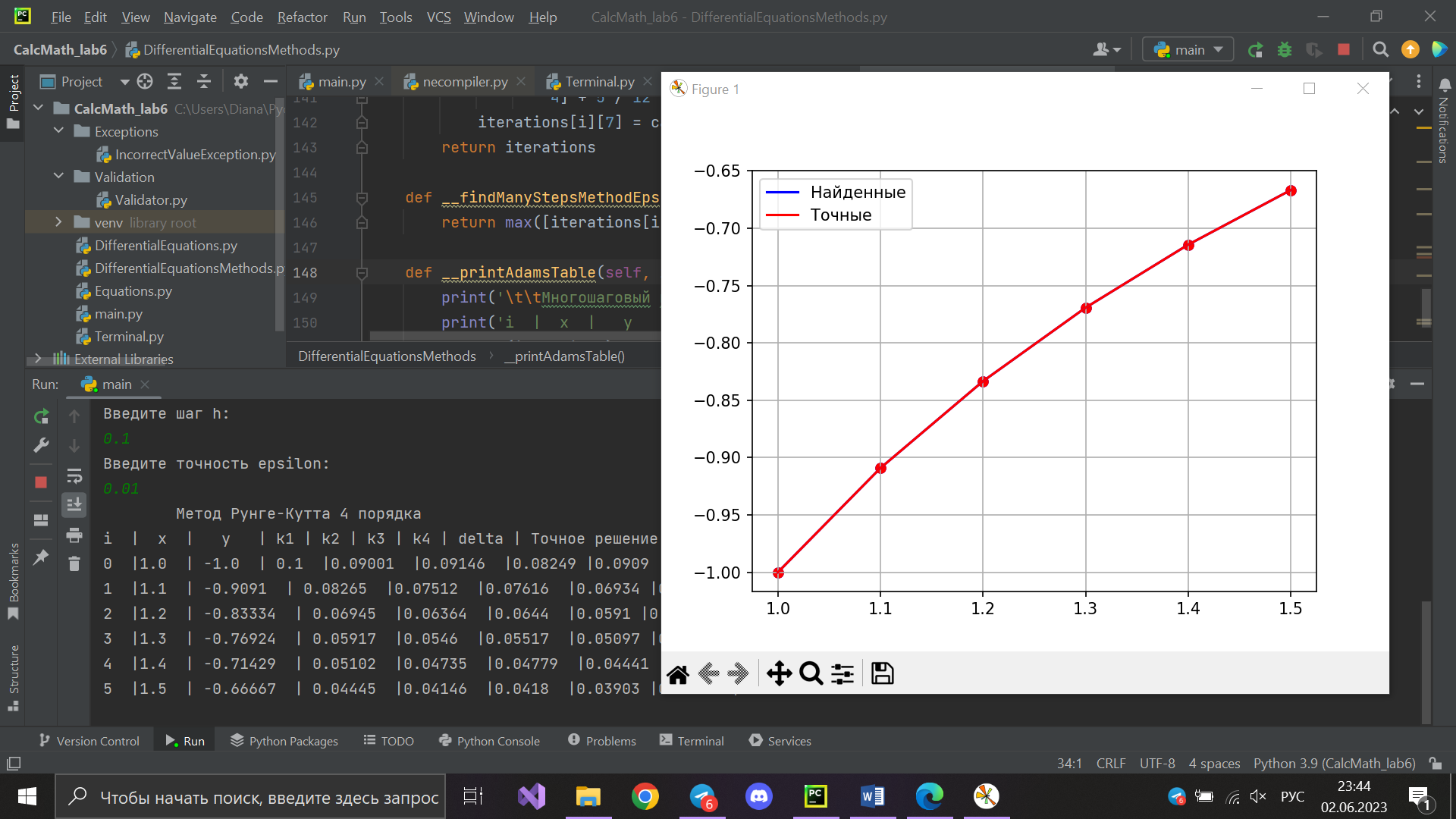
**

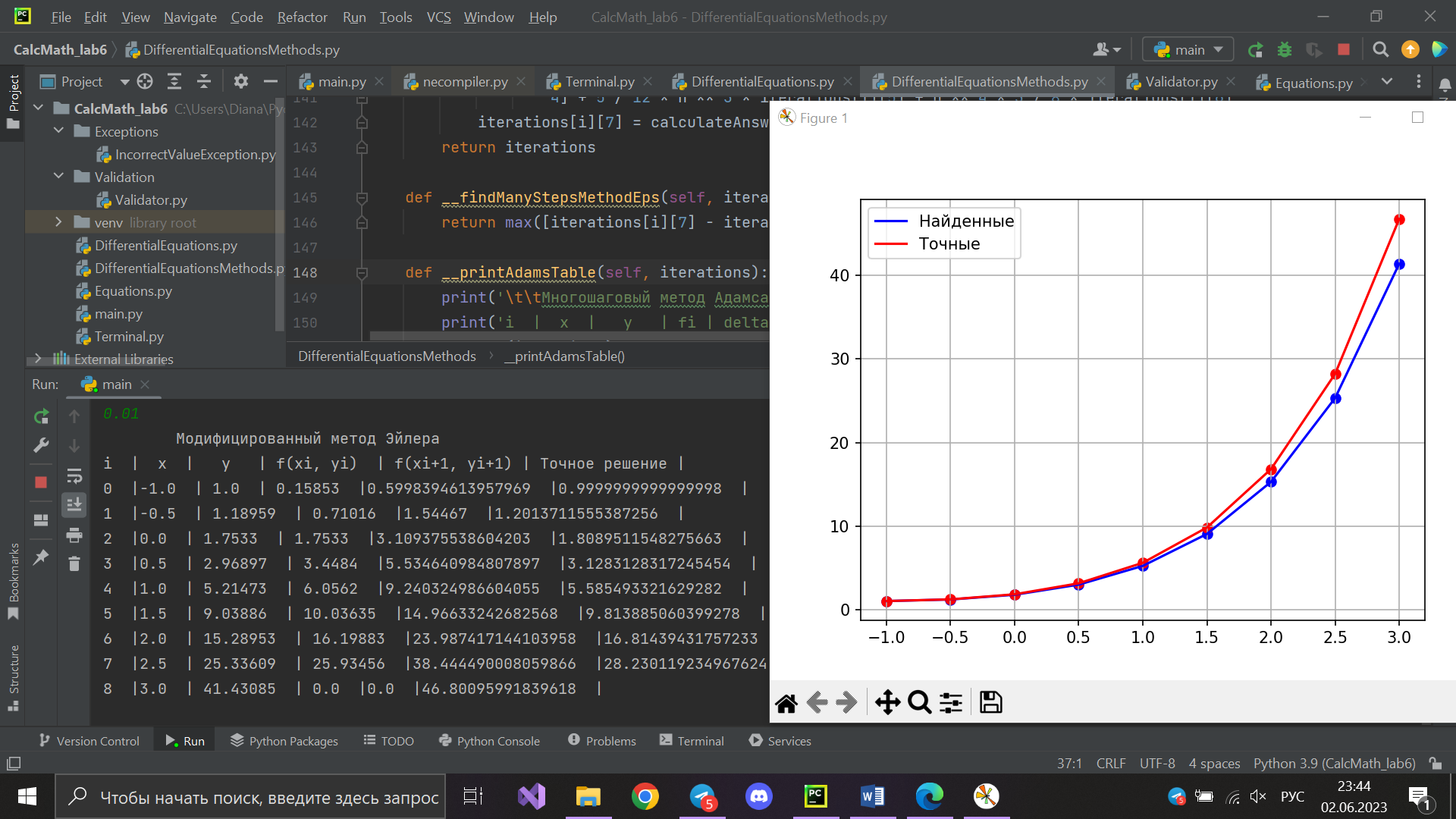
***4. Листинг:***

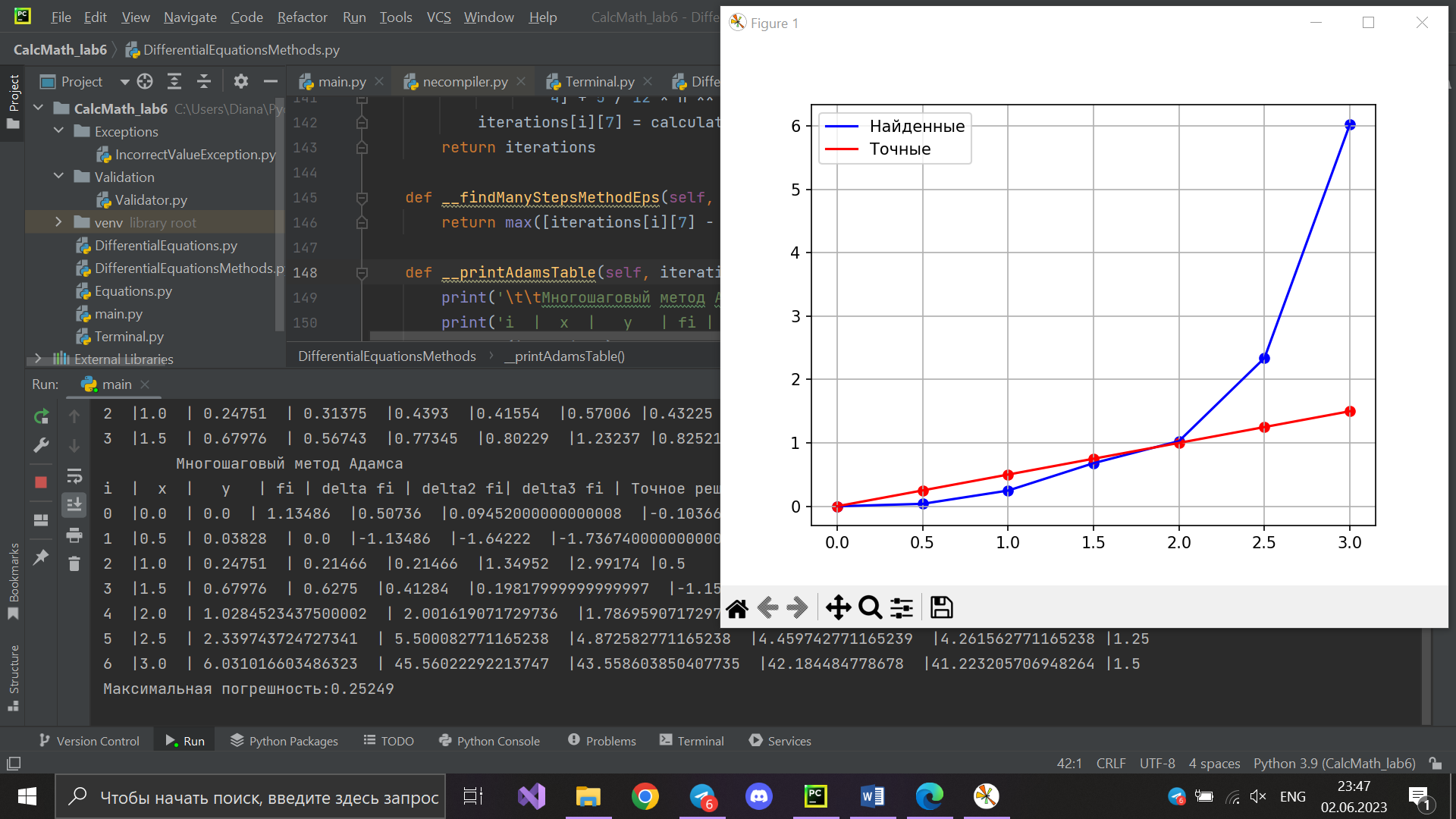
import math  
  
import numpy as np  
  
from DifferentialEquations import DifferentialEquations  
from Equations import calculateFunction, calculateAnswer  
from Exceptions.IncorrectValueException import IncorrectValueException  
  
EILER\_COLS = 6  
RUNGE\_COLS = 9  
ADAMS\_COLS = 8  
  
  
class DifferentialEquationsMethods(DifferentialEquations):  
  
 def methodModifiedEiler(self, function\_number):  
 results = self.\_calculateModifiedEilerMethod(function\_number)  
 self.\_\_printEilerModifiedTable(results)  
 return np.array(results)[:, 2], np.array(results)[:, 5]  
  
 def \_calculateModifiedEilerMethod(self, function\_number):  
 x\_arr = self.getArrayX()  
 n = len(x\_arr)  
 h = self.getH()  
 x0 = self.getX0()  
 y0 = self.getY0()  
 iterations = [[0.0 for x in range(EILER\_COLS)] for i in range(n)]  
 for i in range(n):  
 iterations[i][0] = i  
 iterations[i][1] = round(float(x\_arr[i]), 5)  
 if i == 0:  
 iterations[i][2] = y0  
 function = round(calculateFunction(x\_arr[i], iterations[i][2], function\_number), 5)  
 if i != n - 1:  
 iterations[i + 1][2] = round(iterations[i][2] + h / 2 \* (  
 function + calculateFunction(x\_arr[i + 1], iterations[i][2] + h \* function,  
 function\_number)),  
 5) # yi+1  
 if np.isnan(iterations[i + 1][2]) or np.isinf(iterations[i + 1][2]):  
 raise IncorrectValueException('Невозможно вычислить.')  
 iterations[i][3] = function # f(xi,yi)  
 function\_2 = calculateFunction(x\_arr[i + 1], iterations[i][2] + h \* function, function\_number)  
 iterations[i][4] = function\_2 # f(xi+1,yi+hf(xi,yi))  
 iterations[i][5] = calculateAnswer(x0, y0, x\_arr[i], function\_number)  
 return iterations  
  
 def \_\_printEilerModifiedTable(self, iterations):  
 print('\t\tМодифицированный метод Эйлера')  
 print('i | x | y | f(xi, yi) | f(xi+1, yi+1) | Точное решение |')  
 n = len(iterations)  
 for i in range(n):  
 print(f'{iterations[i][0]} |{iterations[i][1]} | {iterations[i][2]} | {iterations[i][3]} |'  
 f'{iterations[i][4]} |{iterations[i][5]} |')  
  
 def methodRungeCutta4(self, function\_number):  
 results = self.\_calculateRungeCutta4(function\_number)  
 self.\_\_printRungeCuttaTable(results, len(results))  
 return np.array(results)[:, 2], np.array(results)[:, 8]  
  
 def \_calculateRungeCutta4(self, function\_number):  
 x\_arr = self.getArrayX()  
 n = len(x\_arr)  
 h = self.getH()  
 y0 = self.getY0()  
 x0 = self.getX0()  
 iterations = [[0.0 for x in range(RUNGE\_COLS)] for i in range(n)]  
 for i in range(n):  
 iterations[i][0] = i  
 iterations[i][1] = round(float(x\_arr[i]), 5)  
 if i == 0:  
 iterations[i][2] = y0  
 iterations[i][3] = round(h \* calculateFunction(x\_arr[i], iterations[i][2], function\_number), 5)  
 iterations[i][4] = round(  
 h \* calculateFunction(x\_arr[i] + h / 2, iterations[i][2] + iterations[i][3] / 2, function\_number),  
 5)  
 iterations[i][5] = round(  
 h \* calculateFunction(x\_arr[i] + h / 2, iterations[i][2] + iterations[i][4] / 2, function\_number),  
 5)  
 iterations[i][6] = round(  
 h \* calculateFunction(x\_arr[i] + h, iterations[i][2] + iterations[i][5], function\_number),  
 5)  
 fun = round(1 / 6 \* (iterations[i][3] + 2 \* iterations[i][4] + 2 \* iterations[i][5] + iterations[i][6]), 5)  
 if i != n - 1:  
 iterations[i + 1][2] = round(iterations[i][2] + fun, 5) # yi+1  
 if np.isnan(iterations[i + 1][2]) or np.isinf(iterations[i + 1][2]):  
 raise IncorrectValueException('Невозможно вычислить.')  
 iterations[i][7] = fun  
 iterations[i][8] = calculateAnswer(x0, y0, x\_arr[i], function\_number)  
 return iterations  
  
 def \_\_printRungeCuttaTable(self, iterations, num):  
 print('\t\tМетод Рунге-Кутта 4 порядка')  
 print('i | x | y | k1 | k2 | k3 | k4 | delta | Точное решение |')  
 for i in range(num):  
 print(f'{iterations[i][0]} |{iterations[i][1]} | {iterations[i][2]} | {iterations[i][3]} |'  
 f'{iterations[i][4]} |{iterations[i][5]} |{iterations[i][6]} |{iterations[i][7]} |{iterations[i][8]}')  
  
 def methodAdams(self, function\_number):  
 results = self.\_calculateRungeCutta4(function\_number)  
 self.\_\_printRungeCuttaTable(results, 4)  
 iterations = self.\_\_calculateAdams(function\_number, results[:4])  
 self.\_\_printAdamsTable(iterations)  
 print(f'Максимальная погрешность:{self.\_\_findManyStepsMethodEps(iterations)}')  
 return np.array(iterations)[:, 2], np.array(iterations)[:, 7]  
  
 def \_\_calculateAdams(self, function\_number, results):  
 x\_arr = self.getArrayX()  
 n = len(x\_arr)  
 h = self.getH()  
 y0 = self.getY0()  
 x0 = self.getX0()  
 iterations = [[0.0 for x in range(ADAMS\_COLS)] for i in range(n)]  
 for i in range(4):  
 iterations[i][0] = i  
 iterations[i][1] = round(float(x\_arr[i]), 5)  
 iterations[i][2] = results[i][2]  
 fi = round(results[i - 1][3] / h, 5)  
 iterations[i][3] = fi  
 delta\_fi = fi - round(results[i - 2][3] / h, 5)  
 iterations[i][4] = delta\_fi  
 delta\_fi\_2 = fi - round(2 \* results[i - 2][3] / h, 5) + round(results[i - 3][3] / h, 5)  
 iterations[i][5] = delta\_fi\_2  
 delta\_fi\_3 = fi - round(3 \* results[i - 2][3] / h, 5) + round(3 \* results[i - 3][3] / h, 5) - round(  
 results[i - 4][3] / h, 5)  
 iterations[i][6] = delta\_fi\_3  
 iterations[i][7] = results[i][8]  
 for i in range(3, n):  
 if i != 3:  
 iterations[i][0] = i  
 iterations[i][1] = round(float(x\_arr[i]), 5)  
 fi = calculateFunction(x\_arr[i], iterations[i][2], function\_number)  
 iterations[i][3] = fi  
 delta\_fi = fi - iterations[i - 2][3]  
 iterations[i][4] = delta\_fi  
 delta\_fi\_2 = fi - 2 \* iterations[i - 2][3] + iterations[i - 3][3]  
 iterations[i][5] = delta\_fi\_2  
 delta\_fi\_3 = fi - 3 \* iterations[i - 2][3] + 3 \* iterations[i - 3][3] - iterations[i - 4][3]  
 iterations[i][6] = delta\_fi\_3  
 if i != n - 1:  
 iterations[i + 1][2] = iterations[i][2] + h \* iterations[i][3] + h \*\* 2 / 2 \* iterations[i][  
 4] + 5 / 12 \* h \*\* 3 \* iterations[i][5] + h \*\* 4 \* 3 / 8 \* iterations[i][6]  
 iterations[i][7] = calculateAnswer(x0, y0, x\_arr[i], function\_number)  
 return iterations  
  
 def \_\_findManyStepsMethodEps(self, iterations):  
 return max([iterations[i][7] - iterations[i][2] for i in range(len(iterations))])  
  
 def \_\_printAdamsTable(self, iterations):  
 print('\t\tМногошаговый метод Адамса')  
 print('i | x | y | fi | delta fi | delta2 fi| delta3 fi | Точное решение |')  
 n = len(iterations)  
 for i in range(n):  
 print(f'{iterations[i][0]} |{iterations[i][1]} | {iterations[i][2]} | {iterations[i][3]} |'  
 f'{iterations[i][4]} |{iterations[i][5]} |{iterations[i][6]} |{iterations[i][7]} ')

import numpy as np  
  
  
class DifferentialEquations:  
  
 def \_\_init\_\_(self, x0, y0, xn, h, eps):  
 self.\_\_x0 = x0  
 self.\_\_y0 = y0  
 self.\_\_xn = xn  
 self.\_\_h = h  
 self.\_\_eps = eps  
 self.\_\_x\_array = list(np.arange(x0, xn + 0.00001, h))  
  
 def getY0(self):  
 return self.\_\_y0  
  
 def getX0(self):  
 return self.\_\_x0  
  
 def getXN(self):  
 return self.\_\_xn  
  
 def getH(self):  
 return self.\_\_h  
  
 def getEps(self):  
 return self.\_\_eps  
  
 def getArrayX(self):  
 return self.\_\_x\_array  
  
 def ruleRunge(self, I\_h, I\_h2, k, eps):  
 delta = abs(I\_h2 - I\_h) / (2 \*\* k - 1)  
 print(  
 f'Погрешность вычислений между значениями {I\_h2} и {I\_h} составляет {delta} {"<" if delta < eps else ">"} {eps}.')  
 if delta <= eps:  
 return True  
 else:  
 return False

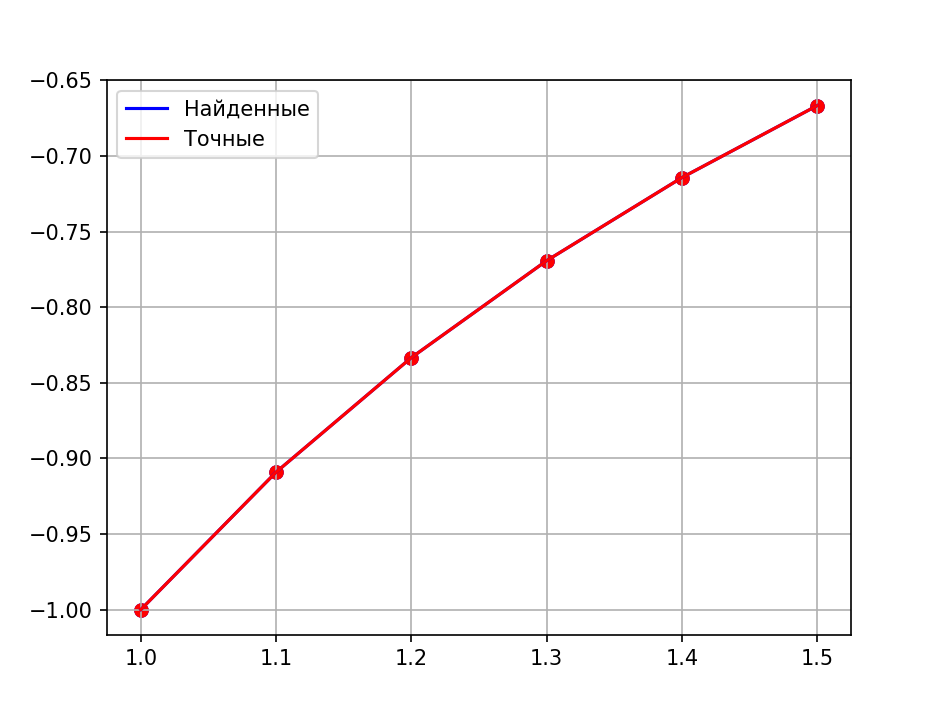
***5. Скриншоты результатов:***

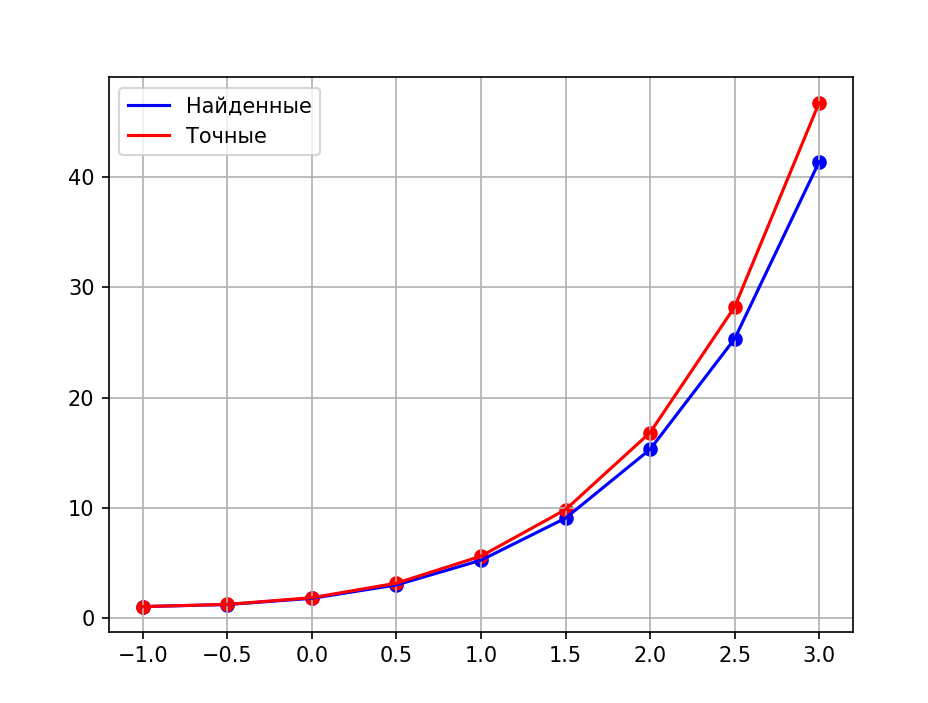


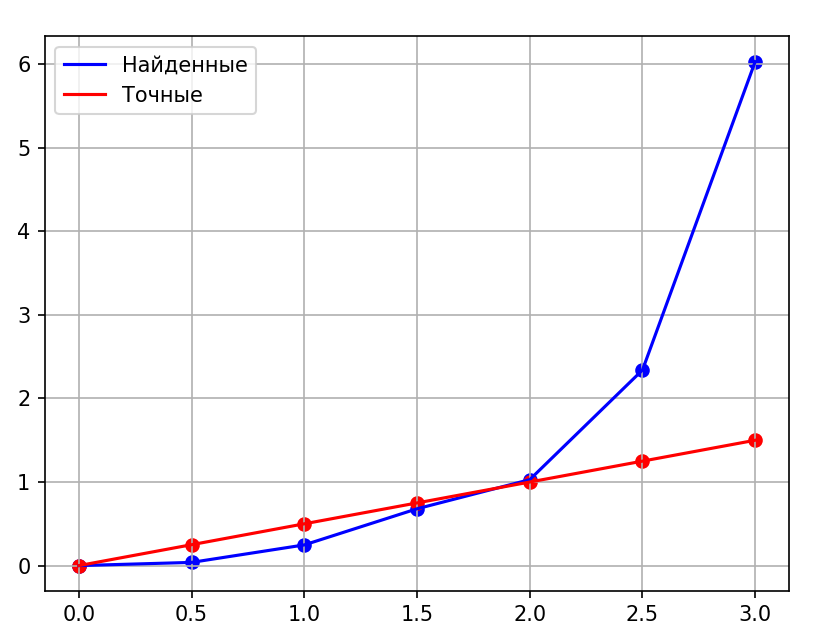




***6. Графики:***

******

******

******

***7. Выводы:*** я научилась решать задачу Коши для обыкновенных дифференциальных уравнений численными методами.