

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: Р3111

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-Кутты 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 $x = x_i + h/2$ ($i=0, 1, \dots$), 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:

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

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:

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

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

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-Кутты).

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])
        x1 = float(xi + h)
        f1 = get_val(equation, xi, yi)
```

```

        f2 = get_val(equation, xil, yi + h * f1)
        yil = float(yi + h / 2 * (f1 + f2))
        result.append((xil, yil))
    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]
        xil = 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))
        yil = float(result[i - 1][1])
        filr = float(get_val(eq, xil, y_cur))
        y_prev = float(yil + h / 3 * (f1 + 4 * fi + filr))
        while abs(y_prev - y_cur) > eps:
            y_cur = y_prev
            filr = get_val(eq, xil, y_cur)
            y_prev = yil + h / 3 * (f1 + 4 * fi + filr)
        result.append((xil, 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[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:

```

```

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 ерс:
ерс =
0.000001
15
[(1.0, -1.0), (1.0999999999999943, -0.9090910881323658), (1.1999999999999986, -0.8333336009569484), (1.2999999999999983, -0.7692310747861445), (1.3999999999999973, -0.71428602), (1.4999999999999963, -0.6666666666666667), (1.5999999999999953, -0.6250000000000001), (1.6999999999999943, -0.5882352941176471), (1.7999999999999933, -0.5555555555555556), (1.8999999999999923, -0.5263157894736842), (1.9999999999999913, -0.5000000000000001), (2.0999999999999903, -0.4761904761904762), (2.1999999999999893, -0.4545454545454545), (2.2999999999999883, -0.4347826086956522), (2.3999999999999873, -0.4166666666666667), (2.4999999999999863, -0.4000000000000001), (2.5999999999999853, -0.3846153846153846), (2.6999999999999843, -0.3703703703703703), (2.7999999999999833, -0.3571428571428571), (2.8999999999999823, -0.3450000000000000), (2.9999999999999813, -0.3338957044973766), (3.0999999999999803, -0.3238095238095238), (3.1999999999999793, -0.3146875000000000), (3.2999999999999783, -0.3064935064935065), (3.3999999999999773, -0.2991428571428571), (3.4999999999999763, -0.2926315789473684), (3.5999999999999753, -0.2868421052631579), (3.6999999999999743, -0.2817647058823529), (3.7999999999999733, -0.2773913043478261), (3.8999999999999723, -0.2737142857142857), (3.9999999999999713, -0.2707142857142857), (4.0999999999999703, -0.2683703703703704), (4.1999999999999693, -0.2666666666666667), (4.2999999999999683, -0.2655555555555556), (4.3999999999999673, -0.2649056603773585), (4.4999999999999663, -0.2646977777777778), (4.5999999999999653, -0.2648957264957265), (4.6999999999999643, -0.2654761904761905), (4.7999999999999633, -0.2664102564102564), (4.8999999999999623, -0.2676693989070434), (4.9999999999999613, -0.2692310747861445), (5.0999999999999603, -0.2710952380952381), (5.1999999999999593, -0.2732608695652174), (5.2999999999999583, -0.2757142857142857), (5.3999999999999573, -0.2784615384615385), (5.4999999999999563, -0.2815000000000000), (5.5999999999999553, -0.2848333333333333), (5.6999999999999543, -0.2884615384615385), (5.7999999999999533, -0.2923809523809524), (5.8999999999999523, -0.2965925925925926), (5.9999999999999513, -0.3011037735849056), (6.0999999999999503, -0.3059142857142857), (6.1999999999999493, -0.3110256410256410), (6.2999999999999483, -0.3164380952380952), (6.3999999999999473, -0.3221519230769231), (6.4999999999999463, -0.3281671755725191), (6.5999999999999453, -0.3344838709677419), (6.6999999999999443, -0.3411020408163265), (6.7999999999999433, -0.3480217391304348), (6.8999999999999423, -0.3552430952380952), (6.9999999999999413, -0.3627661904761905), (7.0999999999999403, -0.3705921052631579), (7.1999999999999393, -0.3787209302325581), (7.2999999999999383, -0.3871526666666667), (7.3999999999999373, -0.3958873170731707), (7.4999999999999363, -0.4049249999999999), (7.5999999999999353, -0.4142657142857143), (7.6999999999999343, -0.4239095238095238), (7.7999999999999333, -0.4338563829787234), (7.8999999999999323, -0.4441063829787234), (7.9999999999999313, -0.4546595238095238), (8.0999999999999303, -0.4655157894736842), (8.1999999999999293, -0.4766750000000000), (8.2999999999999283, -0.4881372319047619), (8.3999999999999273, -0.4999024390243902), (8.4999999999999263, -0.5119707264957265), (8.5999999999999253, -0.5243421052631579), (8.6999999999999243, -0.5370166666666667), (8.7999999999999233, -0.5500000000000001), (8.8999999999999223, -0.5632926829268293), (8.9999999999999213, -0.5768947368421053), (9.0999999999999203, -0.5908063829787234), (9.1999999999999193, -0.6050276190476191), (9.2999999999999183, -0.6195584210526316), (9.3999999999999173, -0.6343987317073171), (9.4999999999999163, -0.6495486190476191), (9.5999999999999153, -0.6649990952380952), (9.6999999999999143, -0.6807511904761905), (9.7999999999999133, -0.6968050000000001), (9.8999999999999123, -0.7131607264957265), (9.9999999999999113, -0.7298184210526316), (10.0999999999999103, -0.7467781707317073), (10.1999999999999093, -0.7640399736842105), (10.2999999999999083, -0.7816038260869565), (10.3999999999999073, -0.7994696302631579), (10.4999999999999063, -0.817637380952381), (10.5999999999999053, -0.8361070769230769), (10.6999999999999043, -0.8548787264957265), (10.7999999999999033, -0.873952330952381), (10.8999999999999023, -0.8933279761904762), (10.9999999999999013, -0.9130056603773585), (11.0999999999999003, -0.9329853846153846), (11.1999999999998993, -0.9532671428571429), (11.2999999999998983, -0.9738519230769231), (11.3999999999998973, -0.9947397264957265), (11.4999999999998963, -1.0159315789473684), (11.5999999999998953, -1.0374274761904762), (11.6999999999998943, -1.0592284210526316), (11.7999999999998933, -1.0813344210526316), (11.8999999999998923, -1.1037454761904762), (11.9999999999998913, -1.1264615789473684), (12.0999999999998903, -1.1494827264957265), (12.1999999999998893, -1.172808923076923), (12.2999999999998883, -1.1964391707317073), (12.3999999999998873, -1.2203734761904762), (12.4999999999998863, -1.244611842105263), (12.5999999999998853, -1.269154260869565), (12.6999999999998843, -1.294000731707317), (12.7999999999998833, -1.319151260869565), (12.8999999999998823, -1.344605857142857), (12.9999999999998813, -1.370364523809524), (13.0999999999998803, -1.396427260869565), (13.1999999999998793, -1.422794076923077), (13.2999999999998783, -1.449464973684211), (13.3999999999998773, -1.476439973684211), (13.4999999999998763, -1.503719076923077), (13.5999999999998753, -1.531302280952381), (13.6999999999998743, -1.559189595238095), (13.7999999999998733, -1.587380926495726), (13.8999999999998723, -1.615876273529412), (13.9999999999998713, -1.644675736842105), (14.0999999999998703, -1.673779317073171), (14.1999999999998693, -1.703187015789474), (14.2999999999998683, -1.732898833333333), (14.3999999999998673, -1.762913773529412), (14.4999999999998663, -1.793231842105263), (14.5999999999998653, -1.823852941176471), (14.6999999999998643, -1.854777076923077), (14.7999999999998633, -1.885994260869565), (14.8999999999998623, -1.917503476190476), (14.9999999999998613, -1.949304726495726), (15.0999999999998603, -1.981398015789474), (15.1999999999998593, -2.013783342105263), (15.2999999999998583, -2.046460714285714), (15.3999999999998573, -2.079429142857143), (15.4999999999998563, -2.112688630263158), (15.5999999999998553, -2.146239170731707), (15.6999999999998543, -2.180080769230769), (15.7999999999998533, -2.214213421052632), (15.8999999999998523, -2.248637142857143), (15.9999999999998513, -2.283351923076923), (16.0999999999998503, -2.318357760869565), (16.1999999999998493, -2.353654657894737), (16.2999999999998483, -2.389242619047619), (16.3999999999998473, -2.425121642857143), (16.4999999999998463, -2.461291736842105), (16.5999999999998453, -2.49775290952381), (16.6999999999998443, -2.534505169230769), (16.7999999999998433, -2.571548515789474), (16.8999999999998423, -2.608882947368421), (16.9999999999998413, -2.646507460869565), (17.0999999999998403, -2.684422057142857), (17.1999999999998393, -2.722626736842105), (17.2999999999998383, -2.761121500000000), (17.3999999999998373, -2.800006257894737), (17.4999999999998363, -2.839181015789474), (17.5999999999998353, -2.878645773529412), (17.6999999999998343, -2.918399526315789), (17.7999999999998333, -2.959442273529412), (17.8999999999998323, -2.999774015789474), (17.9999999999998313, -3.040394757894737), (18.0999999999998303, -3.081294497368421), (18.1999999999998293, -3.122473236842105), (18.2999999999998283, -3.163930973684211), (18.3999999999998273, -3.205667714285714), (18.4999999999998263, -3.247683457894737), (18.5999999999998253, -3.289978192307692), (18.6999999999998243, -3.332551923076923), (18.7999999999998233, -3.375404657894737), (18.8999999999998223, -3.418536384615385), (18.9999999999998213, -3.461947105263158), (19.0999999999998203, -3.505636819230769), (19.1999999999998193, -3.549605526315789), (19.2999999999998183, -3.593853236842105), (19.3999999999998173, -3.638379947368421), (19.4999999999998163, -3.683185657894737), (19.5999999999998153, -3.728269368421053), (19.6999999999998143, -3.773631076923077), (19.7999999999998133, -3.819270782608696), (19.8999999999998123, -3.865188486842105), (19.9999999999998113, -3.911384189736842), (20.0999999999998103, -3.957857891707317), (20.1999999999998093, -4.004609592608696), (20.2999999999998083, -4.051639292608696), (20.3999999999998073, -4.098947992608696), (20.4999999999998063, -4.146535692608696), (20.5999999999998053, -4.194402392608696), (20.6999999999998043, -4.242548092608696), (20.7999999999998033, -4.290972792608696), (20.8999999999998023, -4.339676492608696), (20.9999999999998013, -4.388659192608696), (21.0999999999998003, -4.437920892608696), (21.1999999999997993, -4.487462592608696), (21.2999999999997983, -4.537284292608696), (21.3999999999997973, -4.587385992608696), (21.4999999999997963, -4.637767692608696), (21.5999999999997953, -4.688429392608696), (21.6999999999997943, -4.739371092608696), (21.7999999999997933, -4.790592792608696), (21.8999999999997923, -4.842094492608696), (21.9999999999997913, -4.893876192608696), (22.0999999999997903, -4.945937892608696), (22.1999999999997893, -5.000000000000000), (22.2999999999997883, -5.054172700000000), (22.3999999999997873, -5.108455300000000), (22.4999999999997863, -5.162847700000000), (22.5999999999997853, -5.217349900000000), (22.6999999999997843, -5.271961900000000), (22.7999999999997833, -5.326683700000000), (22.8999999999997823, -5.381515300000000), (22.9999999999997813, -5.436456700000000), (23.0999999999997803, -5.491507900000000), (23.1999999999997793, -5.546668900000000), (23.2999999999997783, -5.601939700000000), (23.3999999999997773, -5.657320300000000), (23.4999999999997763, -5.712810700000000), (23.5999999999997753, -5.768410900000000), (23.6999999999997743, -5.824120900000000), (23.7999999999997733, -5.879940700000000), (23.8999999999997723, -5.935870300000000), (23.9999999999997713, -5.991909700000000), (24.0999999999997703, -6.048058900000000), (24.1999999999997693, -6.104317900000000), (24.2999999999997683, -6.160686700000000), (24.3999999999997673, -6.217265300000000), (24.4999999999997663, -6.273953700000000), (24.5999999999997653, -6.330751900000000), (24.6999999999997643, -6.387660000000000), (24.7999999999997633, -6.444677900000000), (24.8999999999997623, -6.501805600000000), (24.9999999999997613, -6.559043100000000), (25.0999999999997603, -6.616390400000000), (25.1999999999997593, -6.673847500000000), (25.2999999999997583, -6.731414400000000), (25.3999999999997573, -6.789091100000000), (25.4999999999997563, -6.846877600000000), (25.5999999999997553, -6.904773900000000), (25.6999999999997543, -6.962780000000000), (25.7999999999997533, -7.020895900000000), (25.8999999999997523, -7.079021600000000), (25.9999999999997513, -7.137257100000000), (26.0999999999997503, -7.195602400000000), (26.1999999999997493, -7.254057500000000), (26.2999999999997483, -7.312622400000000), (26.3999999999997473, -7.371297100000000), (26.4999999999997463, -7.430081600000000), (26.5999999999997453, -7.488975900000000), (26.6999999999997443, -7.547979900000000), (26.7999999999997433, -7.607093600000000), (26.8999999999997423, -7.666317000000000), (26.9999999999997413, -7.725650100000000), (27.0999999999997403, -7.785092900000000), (27.1999999999997393, -7.844645400000000), (27.2999999999997383, -7.904307600000000), (27.3999999999997373, -7.964079500000000), (27.4999999999997363, -8.023961100000000), (27.5999999999997353, -8.083952400000000), (27.6999999999997343, -8.144053400000000), (27.7999999999997333, -8.204264100000000), (27.8999999999997323, -8.264584500000000), (27.9999999999997313, -8.325014600000000), (28.0999999999997303, -8.385554400000000), (28.1999999999997293, -8.446203900000000), (28.2999999999997283, -8.506963100000000), (28.3999999999997273, -8.567832000000000), (28.4999999999997263, -8.628810600000000), (28.5999999999997253, -8.689898900000000), (28.6999999999997243, -8.751096900000000), (28.7999999999997233, -8.812404600000000), (28.8999999999997223, -8.873821900000000), (28.9999999999997213, -8.935348800000000), (29.0999999999997203, -8.996985300000000), (29.1999999999997193, -9.058731400000000), (29.2999999999997183, -9.120587100000000), (29.3999999999997173, -9.182552400000000), (29.4999999999997163, -9.244627300000000), (29.5999999999997153, -9.306811800000000), (29.6999999999997143, -9.369105900000000), (29.7999999999997133, -9.431509600000000), (29.8999999999997123, -9.494022900000000), (29.9999999999997113, -9.556645800000000), (30.0999999999997103, -9.619378300000000), (30.1999999999997093, -9.682220400000000), (30.2999999999997083, -9.745172100000000), (30.3999999999997073, -9.808233400000000), (30.4999999999997063, -9.871404300000000), (30.5999999999997053, -9.934684800000000), (30.6999999999997043, -10.000000000000000), (30.7999999999997033, -10.065369900000000), (30.8999999999997023, -10.130804300000000), (30.9999999999997013, -10.196303200000000), (31.0999999999997003, -10.261866600000000), (31.1999999999996993, -10.327494500000000), (31.2999999999996983, -10.393186900000000), (31.3999999999996973, -10.458943800000000), (31.4999999999996963, -10.524765200000000
```