**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский химико-технологический университет**

**имени Д.И. Менделеева»**

Факультет цифровых технологий и химического инжиниринга

Кафедра информационных компьютерных технологий

**ОТЧЕТ**

**ПО КУРСОВОЙ РАБОТЕ**

**на тему:**

**«Графический калькулятор уравнений»**

Заведующая кафедрой ИКТ,

д.т.н., профессор **Кольцова Э.М.**

Руководитель работы

Старший преподаватель **Васецкий А.М.**

**СТУДЕНТ группы КС-28**  **Казанцев Л.А.**

**Москва**

**2022**

Оглавление

[Техническое задание 3](#_Toc104047127)

[Описание алгоритма 4](#_Toc104047128)

[Описание реализации 5](#_Toc104047129)

[Исходный код программы 8](#_Toc104047130)

[Тестирование 17](#_Toc104047131)

## Техническое задание

Задание 1

Решение нелинейного уравнения методом Ньютона с вариантами аппроксимации производной конечными разностями.

Задание 2

Построить точечную диаграмму функции на заданном интервале с форматированием легенды (наличие/отсутствие, цвет).

Задание 3

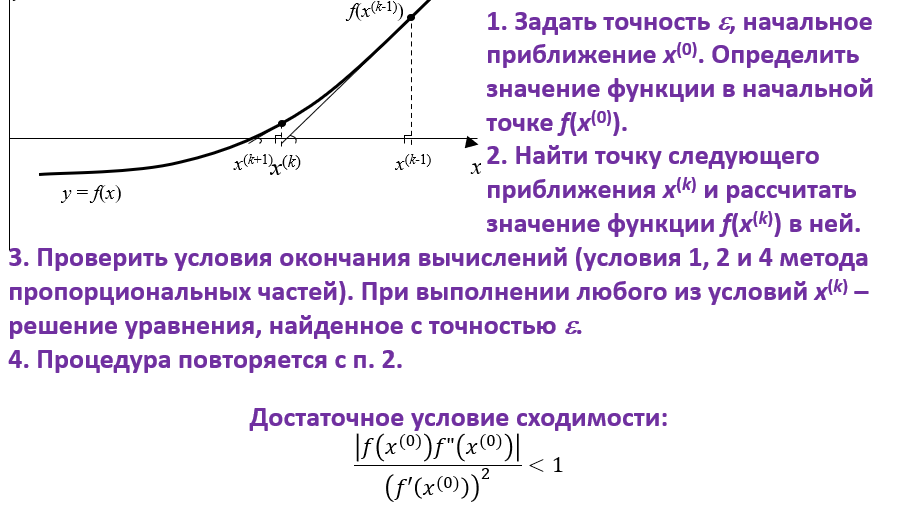
Запись исходных данных и результатов расчётов. Вывести в отдельный файл все значения функций, которые были вычислены в ходе выполнения программы.

Задание 4

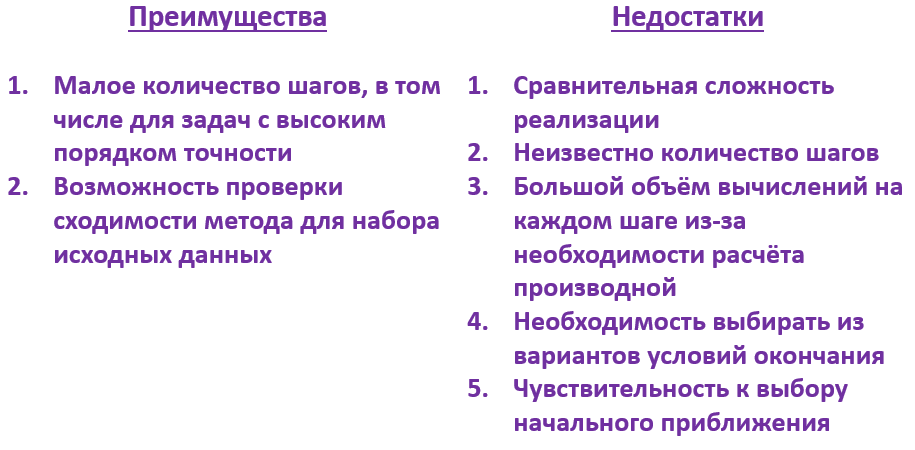
Запись в бинарный файл всех пользовательских настроек и считывание их оттуда.

## Описание алгоритма

Для решения поставленной задачи был использован метод Ньютона для решения нелинейных алгебраических уравнений. Алгоритм:



Для подсчета производной первого порядка использовалась центральная конечная разность. Преимущества и недостатки данного метода:



## Описание реализации

Для выполнения курсовой работы был выбран язык программирования python 3.9. Для графической части использовались модули tkinter, matplotlib. Для преобразования введенных данных из символьного вида в представление функции были использованы функции из модуля sympy sp.lambdify() и sp.parse\_expr(). Также использовался модуль numpy для генерации исключения, если полученная с помощью lambdify математическая функция уравнения выдавала ошибку в подсчетах.

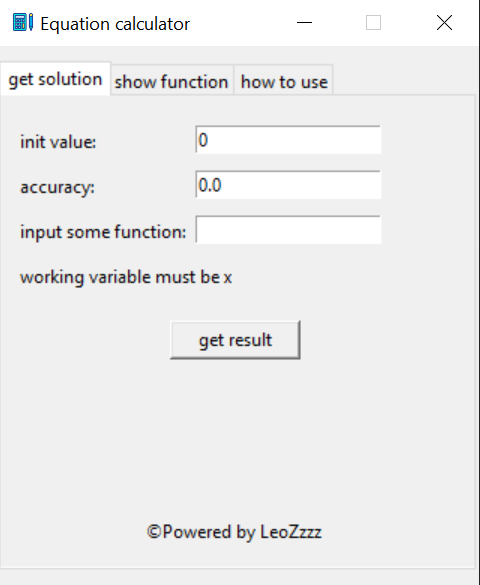
Всего в программе 5 файлов: app.py, thread.py, main\_alg.py, show\_graphic.py, binary.py.

В Первом файле app.py находится основной код приложения, его графический интерфейс, из которого происходит вызов всех остальных модулей.

Файл thread.py служит для создания потока, в котором будет выполняться основной алгоритм нахождения решения для заданных значений начального приближения, точности и исходной функции, находящийся в файле main\_alg.py.

Файл binary.py служит для записи в бинарный файл всех данных, введенных пользователем, и всех пользовательских настроек приложения.

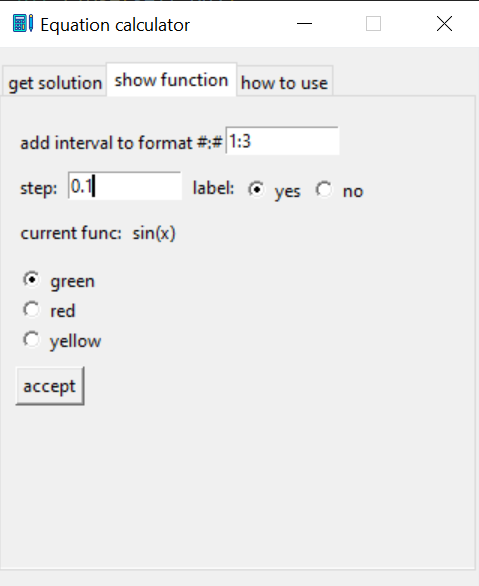
И наконец файл show\_graphic.py создает объект фигуры, который в последствии используется основным кодом программы.



Первое окно приложения

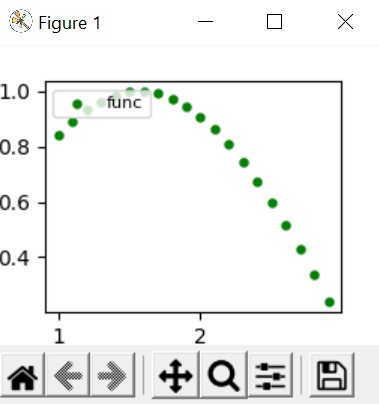
Здесь мы вводим данные для нахождения решения уравнения. После нажатия на кнопку «get result» запускается поток для вычисления значений, в котором, как говорилось ранее, запускается основной алгоритм подсчета.

Перейдем к разделу вывода графика.



Раздел для вывода графика функции уравнения

Сам график выводится в отдельном окне:

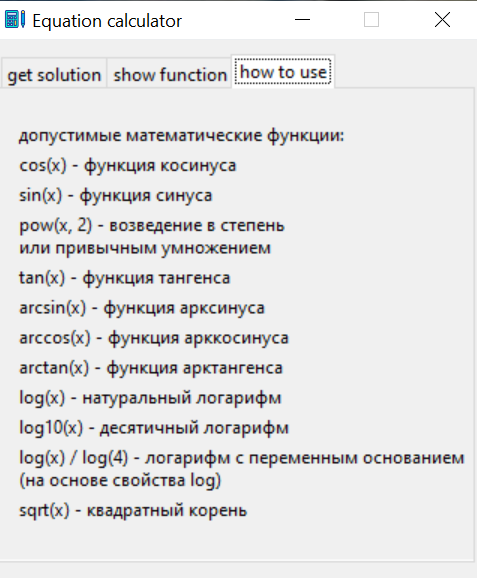


Окно вывода графика функции

Это полноценное окно раздела matplotlib.pyplot со всеми возможностями для взаимодействия с выведенным графиком.

Для примера была взята функция синуса на интервале от 1 до 3 и с шагом 0.1. В этом разделе есть различные настройки отображения, такие как интервал, шаг, наличие легенды и цвет графика. После нажатия кнопки «accept» запускается функция, в которой осуществляется проверка всех полученных из формы данных и, если все правильно, выводится график.

Перейдем к последнему разделу, разделу документации.



Раздел документации

В этом разделе описано в каком виде нужно задавать математические функции, чтобы алгоритм смог их преобразовать для нахождения решения.

В программе реализована обработка различных ошибок, например, ошибки подсчета, ошибки, возникающие при вводе данных, ошибки, возникающие при отсутствии решения у уравнения.

## Исходный код программы

Файл app.py

from tkinter import \*

from tkinter import messagebox

from tkinter.ttk import Notebook

import main\_alg as m

import binary as b

import matplotlib.pyplot as plt

import show\_graphic as show

from thread import \*

import asyncio

import numpy as np

FUNCTION\_LENGTH = 50

class App(Tk):

        \_\_flag\_\_ = True

        \_\_flag1\_\_ = False

        def \_\_init\_\_(self):

            super().\_\_init\_\_()

            self.geometry("320x360+450+180")

            self.resizable(width=False, height=False)

            self.title("Equation calculator")

            self.iconbitmap('calc\_ico.ico')

            self.func()

        def func(self):

            self.notes = Notebook(self)

            self.notes.pack(pady=10, expand=True)

            self.frame1 = Frame(self.notes, width=400, height=380)

            self.frame2 = Frame(self.notes, width=400, height=380)

            self.frame3 = Frame(self.notes, width=400, height=380)

            self.notes.add(self.frame1, text="get solution")

            self.notes.add(self.frame2, text="show function")

            self.notes.add(self.frame3, text="how to use")

            self.aioloop = starting\_process()

            self.release\_info()

            self.show\_graphs()

            self.doc()

            self.notes.bind('<ButtonRelease-1>', self.hadler)

            self.mainloop()

        def release\_info(self):

            self.text = StringVar()

            self.text\_ = DoubleVar()

            self.text\_\_ = IntVar()

            self.field\_ = Entry(self.frame1, textvariable=self.text, width=20)

            self.field\_\_ = Entry(self.frame1, textvariable=self.text\_, width=20)

            self.show\_ = Label(self.frame1, text="input some function:")

            self.show\_\_ = Label(self.frame1, text="accuracy:")

            self.press = Button(self.frame1, text="get result", command=lambda: asyncio.run\_coroutine\_threadsafe(self.do\_this(), self.aioloop), width=11)

            self.field\_yo = Entry(self.frame1, textvariable=self.text\_\_, width=20)

            self.power = Label(self.frame1, text="©Powered by LeoZzzz")

            self.show\_yo = Label(self.frame1, text="init value:")

            self.show\_inf\_ = Label(self.frame1, text="working variable must be x")

            self.show\_yo.place(x=10,y=20)

            self.field\_yo.place(x=130, y=20)

            self.field\_.place(x=130, y=80)

            self.show\_.place(x=10, y=80)

            self.show\_\_.place(x=10, y=50)

            self.field\_\_.place(x=130, y=50)

            self.show\_inf\_.place(x=10, y=110)

            self.power.place(x=94, y=280)

            self.press.place(x=113, y=150)

        async def do\_this(self):

          try:

              if len(self.text.get()) <= FUNCTION\_LENGTH:

                  b.bin\_first\_in(self.text\_\_.get(), self.text\_.get(), self.text.get())

                  data = b.bin\_first\_out()

                  x, self.prepared\_func = m.alg(data[0], data[1], data[2])

                  if x == "error\_":

                      messagebox.showerror("Alert", "Проверьте правильность введенных данных!")

                      #self.field\_\_.delete("0", END)

                      #self.field\_.delete("0", END)

                      #self.field\_yo.delete("0", END)

                      self.\_\_flag1\_\_ = False

                  elif x == "error":

                      messagebox.showerror("Alert", "Уравнение не сходится!")

                      self.field\_\_.delete("0", END)

                      self.field\_.delete("0", END)

                      self.field\_yo.delete("0", END)

                      self.\_\_flag1\_\_ = False

                  else:

                      messagebox.showinfo("Message", f"Congratulations! You have the result: x={'{:.4f}'.format(x)}, more data in func\_data.txt")

                      self.text\_of\_func = self.text.get()

                      self.field\_\_.delete("0", END)

                      self.field\_.delete("0", END)

                      self.field\_yo.delete("0", END)

                      self.\_\_flag1\_\_ = True

              else:

                  messagebox.showerror("Alert", "Слишком длинное выражение!")

                  #self.field\_\_.delete("0", END)

                  #self.field\_.delete("0", END)

                  #self.field\_yo.delete("0", END)

                  self.\_\_flag1\_\_ = False

          except TypeError:

                messagebox.showerror("Alert", "Попробуйте изменить входные данные...")

                #self.field\_\_.delete("0", END)

                #self.field\_.delete("0", END)

                #self.field\_yo.delete("0", END)

                self.\_\_flag1\_\_ = False

          except Exception:

              messagebox.showerror("Alert", "Что то пошло не так...")

              #self.field\_\_.delete("0", END)

              #self.field\_.delete("0", END)

              #self.field\_yo.delete("0", END)

              self.\_\_flag1\_\_ = False

        def show\_graphs(self):

                self.tumb = IntVar()

                self.tumb1 = IntVar()

                self.color = 'g'

                self.tumb.set(1)

                self.tumb1.set(1)

                self.value\_list\_x = list()

                self.value\_list\_y = list()

                self.way\_to = StringVar()

                self.step = DoubleVar()

                self.but = Button(self.frame2, text="accept", command=self.draw\_it)

                self.t = Label(self.frame2, text="add interval to format #:#")

                self.e = Entry(self.frame2, textvariable=self.way\_to,width=12)

                self.t\_ = Label(self.frame2, text="current func:")

                self.lab = Label(self.frame2, text="label:")

                self.p = Label(self.frame2, text="step:")

                self.e\_ = Entry(self.frame2, textvariable=self.step, width=12)

                self.color\_sw1 = Radiobutton(self.frame2, text="green", value=1, variable=self.tumb, command=self.checking\_c)

                self.color\_sw2 = Radiobutton(self.frame2, text="red", value=2, variable=self.tumb, command=self.checking\_c)

                self.color\_sw3 = Radiobutton(self.frame2, text="yellow", value=3, variable=self.tumb, command=self.checking\_c)

                self.label1 =  Radiobutton(self.frame2, text="yes", value=1, variable=self.tumb1, command=self.checking\_l)

                self.label2 =  Radiobutton(self.frame2, text="no", value=2, variable=self.tumb1, command=self.checking\_l)

                self.t.place(x=10, y=20)

                self.t\_.place(x=10, y=80)

                self.e.place(x=150, y=20)

                self.but.place(x=10, y=180)

                self.p.place(x=10, y=50)

                self.e\_.place(x=45, y=50)

                self.color\_sw1.place(x=10, y=110)

                self.color\_sw2.place(x=10, y=130)

                self.color\_sw3.place(x=10, y=150)

                self.lab.place(x=125, y=50)

                self.label1.place(x=160, y=50)

                self.label2.place(x=205, y=50)

        def checking\_c(self):

            value = self.tumb.get()

            if value == 1:

                self.color = 'g'

            elif value == 2:

                self.color = 'r'

            elif value == 3:

                self.color = 'y'

        def checking\_l(self):

            value = self.tumb1.get()

            if value == 1:

                self.\_\_flag\_\_ = True

            elif value == 2:

                self.\_\_flag\_\_ = False

        def hadler(self, event):

            try:

                if event and self.\_\_flag1\_\_:

                    self.t\_1 = Label(self.frame2, text=self.text\_of\_func)

                    self.t\_1.place(x=85, y=80)

                plt.close(self.obj)

                #self.graph.get\_tk\_widget().destroy()

                self.e.delete("0", END)

                self.e\_.delete("0", END)

            except:

                plugy = "plug"

        def draw\_it(self):

            if self.\_\_flag1\_\_ and self.way\_to.get() and self.step.get():

                try:

                    np.seterr(all='raise')

                    self.value\_list\_x = list()

                    self.value\_list\_y = list()

                    value =  int(self.way\_to.get().split(':')[1])

                    start\_value = int(self.way\_to.get().split(':')[0])

                    b.bin\_second\_in(start\_value, value, self.step.get(), self.\_\_flag\_\_, self.color)

                    data\_ = b.bin\_second\_out()

                    while  data\_[0] <= data\_[1]:

                       self.value\_list\_x.append(data\_[0])

                       self.value\_list\_y.append(self.prepared\_func(data\_[0]))

                       data\_[0] = data\_[0] + data\_[2]

                    try:

                        if self.obj: #and self.graph:

                            plt.close(self.obj)

                            #self.graph.get\_tk\_widget().destroy()

                    except:

                        pass

                    self.obj = show.fig(self.value\_list\_x, self.value\_list\_y, data\_[4], data\_[3])

                    self.obj.show()

                    #self.graph = FigureCanvasTkAgg(self.obj, self.frame2)

                    #self.graph.get\_tk\_widget().place(x=75,y=110,width=260,height=200)

                    #self.graph.draw()

                except FloatingPointError:

                    messagebox.showerror("Alert", "Недопустимый интервал для данной функции")

                    self.e.delete("0", END)

                    self.e\_.delete("0", END)

                except Exception:

                    messagebox.showerror("Alert", "Некорректные данные в форме")

                    self.e.delete("0", END)

                    self.e\_.delete("0", END)

        def doc(self):

            self.header = Label(self.frame3, text="допустимые математические функции:")

            self.f = Label(self.frame3, text="cos(x) - функция косинуса")

            self.f1 = Label(self.frame3, text="sin(x) - функция синуса")

            self.f2 = Label(self.frame3, text="pow(x, 2) - возведение в степень \nили привычным умножением", justify=LEFT)

            self.f3 = Label(self.frame3, text="tan(x) - функция тангенса")

            self.f4 = Label(self.frame3, text="arcsin(x) - функция арксинуса")

            self.f5 = Label(self.frame3, text="arccos(x) - функция арккосинуса")

            self.f6 = Label(self.frame3, text="arctan(x) - функция арктангенса")

            self.f7 = Label(self.frame3, text="log(x) - натуральный логарифм")

            self.f8 = Label(self.frame3, text="log10(x) - десятичный логарифм")

            self.f9 = Label(self.frame3, text="log(x) / log(4) - логарифм с переменным основанием \n(на основе свойства log)", justify=LEFT)

            self.f10 = Label(self.frame3, text="sqrt(x) - квадратный корень")

            self.header.place(x=10, y=20)

            self.f.place(x=10, y=40)

            self.f1.place(x=10, y=60)

            self.f2.place(x=10, y=80)

            self.f3.place(x=10, y=115)

            self.f4.place(x=10, y=135)

            self.f5.place(x=10, y=155)

            self.f6.place(x=10, y=175)

            self.f7.place(x=10, y=195)

            self.f8.place(x=10, y=215)

            self.f9.place(x=10, y=235)

            self.f10.place(x=10, y=270)

if \_\_name\_\_ == "\_\_main\_\_":

   App()

Файл main\_alg.py

import sympy as sp

import numpy as np

Filename = "func\_data.txt"

def deriv(x, prepared\_func, e):

    return ((prepared\_func(x + e) - prepared\_func(x - e)) / (2 \* e))

def deriv2(x, prepared\_func, e):

    return ((prepared\_func(x + 2\*e) - 2 \* prepared\_func(x) + prepared\_func(x - 2\*e) / (4 \* e \*\* 2)))

def alg(\*args):

    try:

        np.seterr(all='raise')

        results\_x = list()

        results\_y = list()

        prepared\_func = sp.lambdify("x", sp.parse\_expr(args[2]), "numpy")

        x\_until = args[0]

        x = x\_until

        results\_x.append(x)

        #if abs(prepared\_func(x\_until) \* deriv2(x\_until, prepared\_func=prepared\_func, e=args[1])) / (deriv(x\_until, prepared\_func=prepared\_func, e=args[1]) \*\* 2) >= 1:

              #return "error", 1

        while True:

            x\_until = x

            x = x\_until - (prepared\_func(x\_until) / deriv(x\_until, prepared\_func=prepared\_func, e=args[1]))

            results\_x.append(x)

            if prepared\_func(x) == 0 or (abs(prepared\_func(x)) <= args[1] and abs(x - x\_until) <= args[1]) or abs((prepared\_func(x) - prepared\_func(x\_until)) / deriv(x, prepared\_func=prepared\_func, e=args[1])) <= args[1]:

                break

        for value in results\_x:

            results\_y.append(prepared\_func(value))

        write\_data(results\_x, results\_y)

        return x, prepared\_func

    except  FloatingPointError:

        return 1

    except TypeError:

        return "error\_", 1

    except Exception:

        return 1

def write\_data(x, y):

    with open(Filename, "w") as file:

       file.write("x: ")

       for value in x:

           file.write(str('{:.4f}'.format(value) + " "))

       file.write("\n")

       file.write("y: ")

       for value in y:

           file.write(str('{:.4f}'.format(value) + " "))

Файл binary.py

import shelve as sh

file\_name = "Cur\_params"

def bin\_first\_in(\*args):

    with sh.open(file\_name, "n") as data:

        data["start\_value"] = args[0]

        data["accuracy"] = args[1]

        data["function"] = args[2]

def bin\_first\_out():

    with sh.open(file\_name, "r") as data:

        data\_ = [data["start\_value"], data["accuracy"], data["function"]]

    return data\_

def bin\_second\_in(\*args):

    with sh.open(file\_name, "n") as data:

        data["start\_value"] = args[0]

        data["value"] = args[1]

        data["step"] = args[2]

        data["flag for lb"] = args[3]

        data["color"] = args[4]

def bin\_second\_out():

    with sh.open(file\_name, "r") as data:

        data\_ = [data["start\_value"], data["value"], data["step"], data["flag for lb"], data["color"]]

    return data\_

Файл thread.py

import asyncio

import threading

from functools import partial

def \_run\_aio\_loop(loop):

    asyncio.set\_event\_loop(loop)

    loop.run\_forever()

def starting\_process():

    aioloop = asyncio.new\_event\_loop()

    t = threading.Thread(target=partial(\_run\_aio\_loop, aioloop), name="Thread\_1")

    t.daemon = True  # Необязательно, в зависимости от того, как планируется закрыть приложение

    t.start()

    return aioloop

Файл show\_graphic.py

import matplotlib.pyplot as plt

def fig(\*args):

      Fig = plt.figure(figsize=(1, 2))

      a = Fig.add\_subplot(111)

      a.scatter(args[0], args[1], s=7, label="func", color=args[2], linewidth=2)

      if args[3]:

         a.legend(fontsize=8, loc='upper left')

      return Fig

## Тестирование

1 тест:

Исходная функция: sin(x) + x \* x - 1

Начальное приближение: 0

Точность: 0.001

Выходные данные:

x: 0.0000 1.0000 0.6688 0.6371 0.6367

y: -1.0000 0.8415 0.0672 0.0007 0.0000

2 тест:

Исходная функция: x \* x + 3 \* x – 1

Начальное приближение: 0

Точность: 0.001

Выходные данные:

x: 0.0000 0.3333 0.3030 0.3028

y: -1.0000 0.1111 0.0009 0.0000

3 тест:

Исходная функция: log(x) + 3 \* x

Начальное приближение: 2

Точность: 0.001

Выходные данные:

x: 2.0000 0.0877 0.2384 0.3383 0.3499 0.3500

y: 6.6931 -2.1711 -0.7188 -0.0691 -0.0006 0.0000

*Моей любимой семье посвящается…*