

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»
Национальный исследовательский университет

Институт информационных технологий, математики и механики
Кафедра алгебры, геометрии и дискретной математики

ЛАБОРАТОРНАЯ РАБОТА

«Численное решение задачи Коши для ОДУ»

Выполнила:

студент группы 381706-2
Ясакова Анастасия Евгеньевна

_____ Подпись

Проверил:

Морозов Кирилл Евгеньевич

_____ Подпись

Нижний Новгород

2020

Оглавление

Введение.....	3
Метод Рунге-Кутты.....	4
Уравнение маятника с диссипацией.....	4
Руководство пользователя.....	5
Руководство программиста.....	8
Пример работы	10
Заключение	10

Введение

Обыкновенными дифференциальными уравнениями (ОДУ) называются такие уравнения, которые содержат одну или несколько производных от искомой функции.

Численные методы решения используют алгоритм вычисления значений искомого решения на некотором дискретном множестве значений аргумента и дают приближенное решение в виде таблицы.

Рассматриваются две группы численных методов решения задачи Коши. Одношаговые методы, в которых для нахождения решения в некоторой точке отрезка используется информация лишь в одной предыдущей точке (например, методы Эйлера, Рунге–Кутты). Многошаговые методы, в которых для отыскания решения в некоторой точке используется информация о решении в нескольких предыдущих точках (например, метод Адамса).

Метод Рунге-Кутты

Метод Рунге-Кутты – наиболее популярный метод решения задачи Коши. Этот метод позволяет строить формулы расчета приближенного решения практически любого порядка точности.

Рассмотрим задачу Коши для нормальной системы второго порядка:

$$\begin{cases} y_1' = f_1(x, y_1, y_2), \\ y_2' = f_2(x, y_1, y_2), \\ y_1(x_0) = y_{10}, \\ y_2(x_0) = y_{20}. \end{cases}$$

Поскольку правые части системы ОДУ зависят от всех искомых функций (в данном случае от y_1, y_2), то приращения для $y_1(x)$ и $y_2(x)$ на каждом этапе вычисляются одновременно. Тогда метод Рунге-Кутты четвертого порядка точности для системы второго порядка имеет вид:

$$\begin{cases} y_{1i+1} = y_{1i} + \Delta y_{1i}, \\ y_{2i+1} = y_{2i} + \Delta y_{2i}, \\ \Delta y_{1i} = \frac{1}{6} \left(K_1^{(i)} + 2K_2^{(i)} + 2K_3^{(i)} + K_4^{(i)} \right), \\ \Delta y_{2i} = \frac{1}{6} \left(L_1^{(i)} + 2L_2^{(i)} + 2L_3^{(i)} + L_4^{(i)} \right). \end{cases}$$
$$K_1^{(i)} = h f_1(x_i, y_{1i}, y_{2i}), \quad L_1^{(i)} = h f_2(x_i, y_{1i}, y_{2i}),$$
$$K_2^{(i)} = h f_1\left(x_i + \frac{h}{2}, y_{1i} + \frac{K_1^{(i)}}{2}, y_{2i} + \frac{L_1^{(i)}}{2}\right),$$
$$L_2^{(i)} = h f_2\left(x_i + \frac{h}{2}, y_{1i} + \frac{K_1^{(i)}}{2}, y_{2i} + \frac{L_1^{(i)}}{2}\right),$$
$$K_3^{(i)} = h f_1\left(x_i + \frac{h}{2}, y_{1i} + \frac{K_2^{(i)}}{2}, y_{2i} + \frac{L_2^{(i)}}{2}\right),$$
$$L_3^{(i)} = h f_2\left(x_i + \frac{h}{2}, y_{1i} + \frac{K_2^{(i)}}{2}, y_{2i} + \frac{L_2^{(i)}}{2}\right),$$
$$K_4^{(i)} = h f_1\left(x_{i+1}, y_{1i} + K_3^{(i)}, y_{2i} + L_3^{(i)}\right),$$
$$L_4^{(i)} = h f_2\left(x_{i+1}, y_{1i} + K_3^{(i)}, y_{2i} + L_3^{(i)}\right).$$

Уравнение маятника с диссипацией

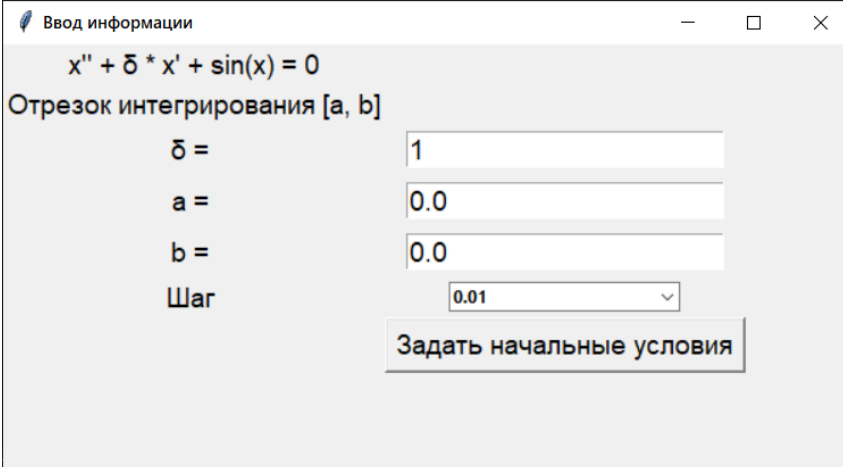
$$\ddot{x} + \delta \dot{x} + \sin x = 0$$

Путем замены $y = x'$ получаем систему двух автономных дифференциальных уравнений:

$$\begin{cases} x' = y \\ y' = -\sin(x) - \delta y \end{cases}$$

Руководство пользователя

1. При запуске пользователю будет предложено ввести параметры ДУ, задать отрезок интегрирования и шаг.



Ввод информации

$$x'' + \delta * x' + \sin(x) = 0$$

Отрезок интегрирования [a, b]

$\delta =$ 1

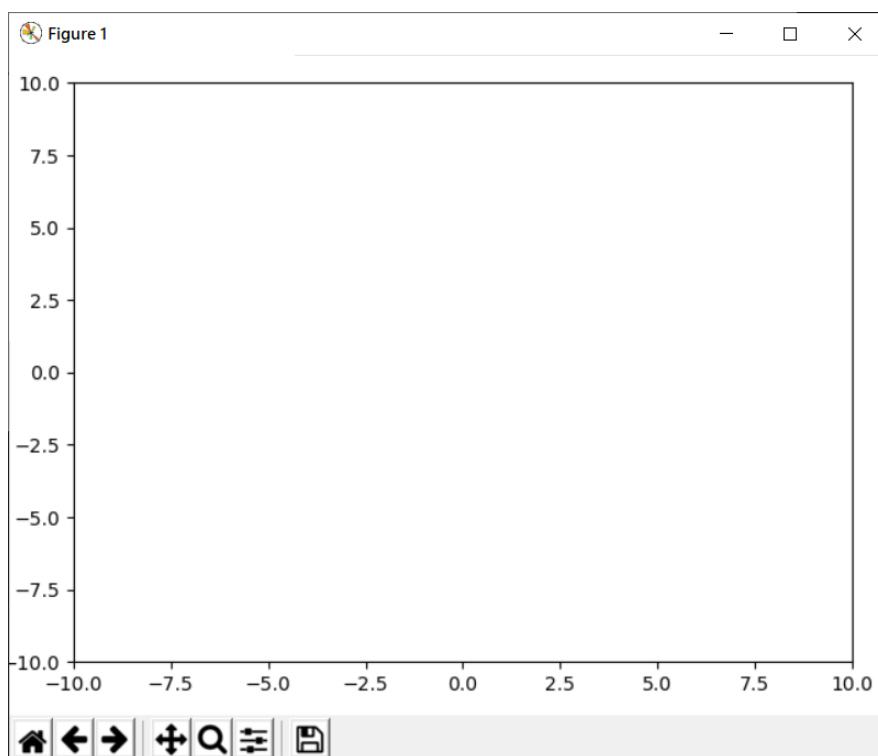
$a =$ 0.0

$b =$ 0.0

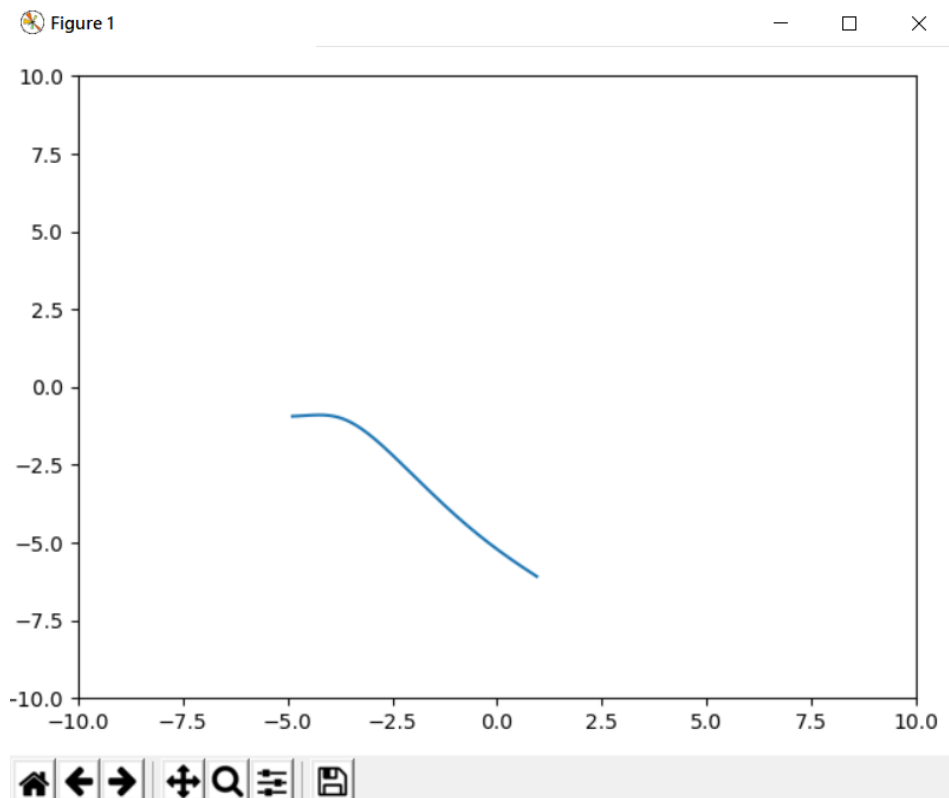
Шаг 0.01

Задать начальные условия

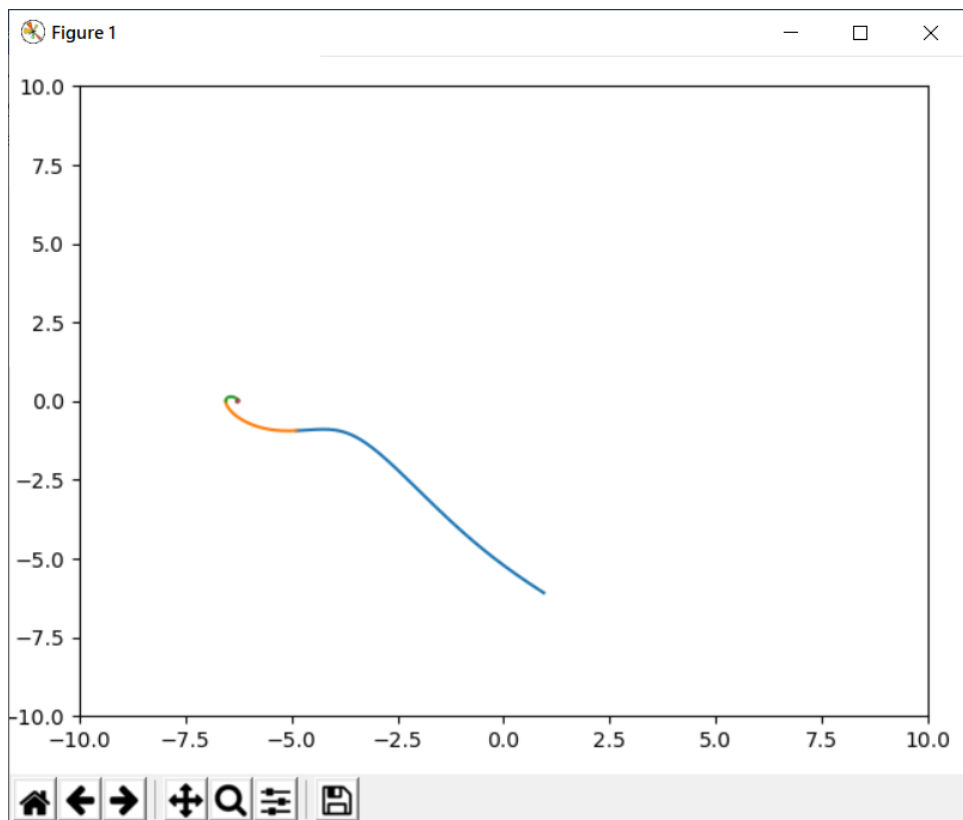
2. После ввода появится окно, где нужно задать начальные условия кликом мыши по графику.



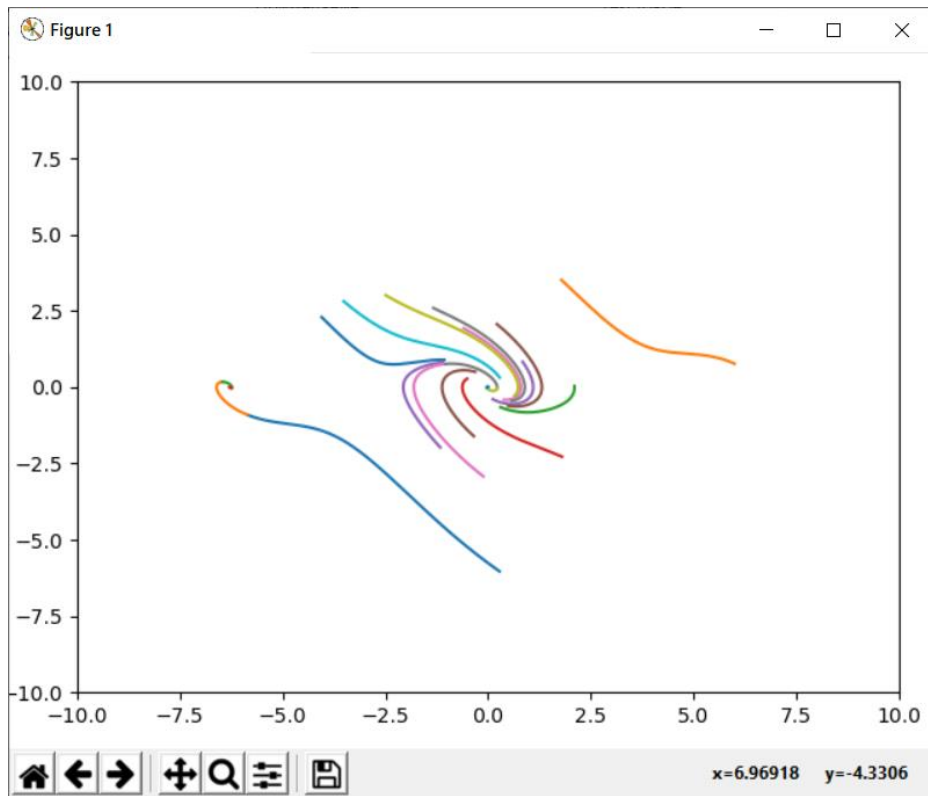
3. После ввода начальных условий появится фазовая траектория.



4. Для того, чтобы посмотреть продолжение фазовой траектории, нужно нажать пробел.



5. Построить новую фазовую траекторию можно кликом мыши в нужное место на графике.



6. Для того, чтобы очистить график, нужно закрыть окно с фазовыми траекториями. Возвращаемся к первому пункту.

Руководство программиста

Программа написана на Python с помощью библиотек matplotlib для визуализации графиков и tkinter для реализации интерфейса.

В функции entry реализовано взаимодействие с графиком.

```
def entry(_a, _b, _c, _h):
    fig, ax = plt.subplots()
    global a, b, c, h
    a = _a
    b = _b
    c = _c
    h = _h

    id1 = fig.canvas.mpl_connect('button_press_event', decision)
    # клик мыши для начальных условий
    id2 = fig.canvas.mpl_connect('key_press_event', continue_)
    # нажатие клавиши для продолжения траектории

    plt.tight_layout()
    plt.axis([-10, 10, -10, 10])
    plt.show()
```

В функции decision реализовано построение фазовой траектории.

```
def decision(event):
    x0 = event.xdata
    y0 = event.ydata # получение начальных условий

    x = []
    y = []

    global a, b, c, h, x_, y_, count
    count = 0

    t_i = a
    x_i = x0
    y_i = y0

    i = a
    while (i <= b):
        k11 = h * y_i
        k12 = h * (-math.sin(x_i) - c * y_i)

        k21 = h * (y_i + k12/2)
        k22 = h * (-math.sin(x_i + k11 / 2) - c * (y_i + k12 / 2))

        k31 = h * (y_i + k22 / 2)
        k32 = h * (-math.sin(x_i + k21 / 2) - c * (y_i + k22 / 2))

        k41 = h * (y_i + k32 / 2)
        k42 = h * (-math.sin(x_i + k31 / 2) - c * (y_i + k32 / 2))
```



```

x_i = x_i + 1/6 * (k11 + 2 * k21 + 2 * k31 + k41)
y_i = y_i + 1/6 * (k12 + 2 * k22 + 2 * k32 + k42)

x.append(x_i)
y.append(y_i)
i = i + h

x_ = x.pop()
x.append(x_)
y_ = y.pop()
y.append(y_)

plt.plot(x, y)
plt.show()

```

В функции `continue_` реализовано построение продолжения фазовой траектории.

```

def continue_(event):
    global a, b, c, h, x_, y_, count
    count = count + 1
    t_i = b * count
    x_i = x_
    y_i = y_

    x = []
    y = []

    i = t_i
    while (i <= b * (count + 1)):
        k11 = h * y_i
        k12 = h * (-math.sin(x_i) - c * y_i)

        k21 = h * (y_i + k12/2)
        k22 = h * (-math.sin(x_i + k11 / 2) - c * (y_i + k12 / 2))

        k31 = h * (y_i + k22 / 2)
        k32 = h * (-math.sin(x_i + k21 / 2) - c * (y_i + k22 / 2))

        k41 = h * (y_i + k32 / 2)
        k42 = h * (-math.sin(x_i + k31 / 2) - c * (y_i + k32 / 2))

        x_i = x_i + 1/6 * (k11 + 2 * k21 + 2 * k31 + k41)
        y_i = y_i + 1/6 * (k12 + 2 * k22 + 2 * k32 + k42)

        x.append(x_i)
        y.append(y_i)
        i = i + h

    x_ = x.pop()
    x.append(x_)
    y_ = y.pop()
    y.append(y_)

    plt.plot(x, y)
    plt.show()

```

В основной части написана реализация интерфейса программы.

Пример работы

Ввод информации

$x'' + \delta * x' + \sin(x) = 0$

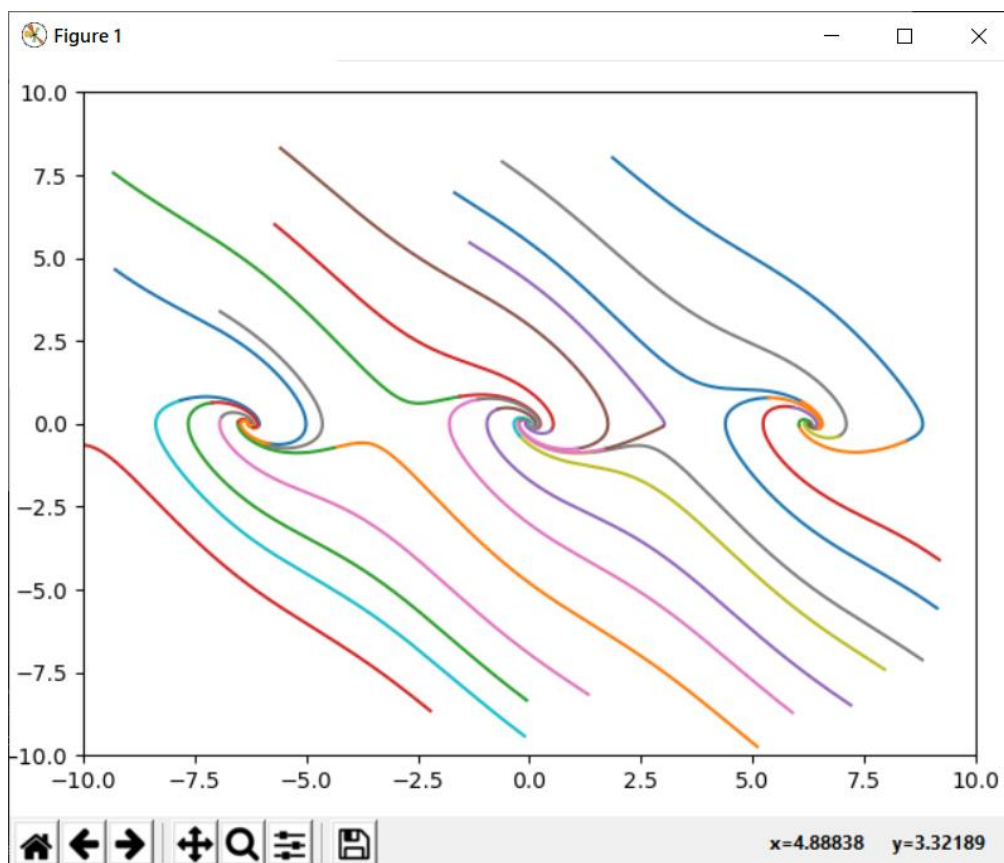
Отрезок интегрирования [a, b]

$\delta =$

$a =$

$b =$

Шаг



Заключение

В процессе работы была изучена задача реализации метода Рунге-Кутты и построения фазовых траекторий.