Межгосударственное образовательное учреждение

высшего образования

«Белорусско-Российский университет»

Кафедра «Высшая математика»

Курсовая работа

по дисциплине «Численные методы математической физики»

на тему: «Решение краевых задач для нелинейных обыкновенный дифференциальных уравнений методом квазилинеаризации»

Выполнила:

Руководитель:

Студентка группы ПМР-211

Скрипунова Валерия Викторовна

Кандидат физико-математических наук, доцент

Маковецкий Илья Иванович

Могилёв 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc153727834)

[Основная часть 4](#_Toc153727835)

[2 Основные теоретические понятия 5](#_Toc153727836)

[2.1 Дифференциальные уравнения 5](#_Toc153727837)

[2.2 Компьютерная наука 6](#_Toc153727838)

[3 Метод квазилинеаризации 8](#_Toc153727839)

[3.1 Описание метода 8](#_Toc153727840)

[3.2 Алгоритм решения 9](#_Toc153727841)

[3.3 Пример 9](#_Toc153727842)

# **ВВЕДЕНИЕ**

*Актуальность темы исследования:*

Дифференциальные уравнения являются одним из основных математических понятий, наиболее широко применяемых при решении практических задач. Причина этого состоит в том, что при исследовании физических процессов, решении различных прикладных задач, как правило, не удаётся непосредственно найти законы, связывающие величины, характеризующие исследуемые явления. Обычно легче устанавливаются зависимости между теми же величинами и их производными или дифференциалами.

Возможности и правила составления дифференциальных уравнений определяются знанием законов той области науки, с которой связана природа изучаемой задачи. Однако на практике часто встречаются случаи, когда законы, которые могли бы позволить составить дифференциальное уравнение, неизвестны.

Для нахождения решения уравнений применяются аналитические и численные методы. Аналитические методы позволяют найти точное решение задачи, но лишь для ограниченного класса дифференциальных уравнений.

*Численные методы для обыкновенных дифференциальных уравнений*— это методы, используемые для нахождения численных приближений к решениям обыкновенных дифференциальных уравнений (ОДУ). Их использование также известно как "численное интегрирование", хотя этот термин также может относиться к вычислению интегралов.

Многие дифференциальные уравнения не могут быть решены точно. Однако для практических целей - например, в инженерном деле – часто бывает достаточно численного приближения к решению.

*Объект исследования:* численные методы для решения краевых задач обыкновенных дифференциальных уравнений

*Предмет исследования:* метод квазилинеаризации и его программная реализация, с использованием фреймворка Django высокоуровнего языка программирования Python.

*Цели и задачи курсовой работы*: научиться решать краевые задачи для нелинейных обыкновенных дифференциальных уравнений методом квазилинеаризации. Разработать программное обеспечение, реализующие метод.

*Методы исследования:*

Изучение специальной литературы, анализ и синтез полученной информации. Проведение практических вычислений, при помощи компьютерных технологий и их исследование.

*Структура работы:*

Работа состоит из содержания, введения, основной части, заключения и списка использованных источников.

# **Основная часть**

**1 Историческая справка**

Первоначально дифференциальные уравнения возникли из задач механики, в которых требовалось определить координаты тел, их скорости и ускорения, рассматриваемые как функции времени при различных воздействиях. К дифференциальным уравнениям приводили также некоторые рассмотренные в то время геометрические задачи. [1]

Основой теории дифференциальных уравнений стало дифференциальное исчисление, созданное Лейбницем и Ньютоном (1642—1727). Сам термин «дифференциальное уравнение» был предложен в 1676 году Лейбницем.

Из огромного числа работ XVIII века по дифференциальным уравнениям выделяются работы Эйлера (1707—1783) и Лагранжа (1736—1813). В этих работах была прежде развита теория малых колебаний, а следовательно — теория линейных систем дифференциальных уравнений; попутно возникли основные понятия линейной алгебры (собственные числа и векторы в n-мерном случае). Вслед за Ньютоном Лаплас и Лагранж, а позже Гаусс (1777—1855) развивают также методы теории возмущений.

Когда была доказана неразрешимость алгебраических уравнений в радикалах, Жозеф Лиувилль (1809—1882) построил аналогичную теорию для дифференциальных уравнений, установив невозможность решения ряда уравнений (в частности, таких классических, как линейные уравнения второго порядка) в элементарных функциях и квадратурах. Позже Софус Ли (1842—1899), анализируя вопрос об интегрировании уравнений в квадратурах, пришёл к необходимости подробно исследовать группы диффеоморфизмов (получившие впоследствии имя групп Ли) — так по теории дифференциальных уравнений возникла одна из самых плодотворных областей современной математики, дальнейшее развитие которой было тесно связано совсем с другими вопросами (алгебры Ли ещё раньше рассматривали Симеон-Дени Пуассон (1781—1840) и, особенно, Карл Густав Якоб Якоби (1804—1851)).

Новый этап развития теории дифференциальных уравнений начинается с работ Анри Пуанкаре (1854—1912), созданная им «качественная теория дифференциальных уравнений» вместе с теорией функций комплексных переменных легла в основу современной топологии. Качественная теория дифференциальных уравнений, или, как теперь её чаще называют, теория динамических систем, сейчас активно развивается и имеет важные применения в естествознании.

## **2 Основные теоретические понятия**

### **2.1 Дифференциальные уравнения**

*Дифференциальным уравнением* называется уравнение относительно неизвестной функции и её производных различных порядков. [2]

*Порядком дифференциального уравнения* называется порядок старшей производной. [3]

Если искомая функция зависит от одной переменной, то соответствующее дифференциальное уравнение называют *обыкновенным*. [4]

Если искомая функция зависит от нескольких переменных, то соответствующее дифференциальное уравнение называют *уравнением с частными производными*. [5]

Существуют также *стохастические дифференциальные уравнения*   
включающие случайные процессы.

Обыкновенное дифференциальное уравнение (ОДУ) *n*-го порядка в общем случае имеет вид:

где *x* – независимая переменная*, y = y(x)* – искомая функция, а – её производные, – заданная функция своих аргументов.

Если уравнение (1.1) разрешимо относительно производной n-го порядка, то его можно представить в виде:

Функция , определённая и непрерывно дифференцируемая n раз в интервале *(a, b)*, называется *решением* дифференциального уравнения (2.1) в этом интервале, если она обращает указанное уравнение в тождество, т.е.

График решения дифференциального уравнения n-го порядка называется *интегральной линией* (или *интегральной кривой*)

*Задача Коши* для дифференциального уравнения n-го порядка состоит в следующем: найти решение *y = y(x)* уравнения (1.1), удовлетворяющее начальным условиям

*Общим решением* дифференциального уравнения n-го порядка (1.1) называется функция

обладающая следующими свойствами:

1. При любых значениях произвольных постоянных она обращает уравнение (2.1) в тождество.
2. Значения постоянных можно подобрать так, чтобы она удовлетворяла условиям (1.3).

*Частным решением* дифференциального уравнения n-го порядка называется решение, получаемое из общего решения (2.4) при фиксированных значениях произвольных постоянных, т.е. функция

где – некоторые числа.

Решение дифференциального уравнения n-го порядка, в каждой точке которого нарушается единственность решения задачи Коши, называется *особым*.

*Общим интегралом* дифференциального уравнения n-го порядка называется соотношение вида

неявно определяющее общее решение этого уравнения.

*Частным интегралом* дифференциального уравнения n-го порядка называется соотношение

полученное из общего интеграла путём фиксирования значений произвольных постоянных.

*Краевой задачей* называется задача вида:

### **2.2 Компьютерная наука**

*Высокоуровневый язык программирования* — язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков — это абстракция, то есть введение смысловых конструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде (или другом низкоуровневом языке программирования) очень длинны и сложны для понимания. [1]

*Объектно-ориентированное программирование (ООП)* – это способ организации программы, позволяющий использовать один и тот же код многократно. [6]

*Класс* – сложный тип данных, включающий набор переменных и функция для управления значениями, хранящимися в этих переменных. Переменные называют *атрибутами*, *свойствами* или *полями,* а функции – *методами*.

*Django* – популярнейший в мире веб-фреймворк, написанный на языке Python, и один из наиболее распространённых в мире. [7]

*Фреймворк* (от англ. framework – каркас) – это программная библиотека, реализующая большую часть типовой функции разрабатываемого продукта. *Веб-фреймворк* – фреймворк для программирования веб-сайтов.

*HTML* — это язык разметки, который представляет простые правила оформления и компактный набор структурных и семантических элементов разметки (тегов). HTML позволяет описывать способ представления логических частей документа (заголовки, абзацы, списки и т.д.) и создавать веб-страницы разной сложности. [8]

*Тег* (html-тег, тег разметки) — управляющая символьная последовательность, которая задает способ отображения гипертекстовой информации.

*Атрибуты* — это пары вида «свойство = значение», уточняющие представление соответствующего тега

*CSS* (от англ. *Cascading Style Sheets* «каскадные таблицы стилей») — формальный язык декорирования и описания внешнего вида документа (веб-страницы). [1]

## **3 Метод квазилинеаризации**

### **3.1 Описание метода**

Пусть дана краевая задача (2.6) на отрезке [*a, b*]. Тогда, необходимо свести краевую задачу к виду:

Получим рекуррентное соотношения. Для этого перепишем уравнение из (3.1) в виде [9, 10]:

Обозначим за n-ю и (n+1)-ю итерацию и соответственно, а также потребуем равенство . Тогда

Преобразуем выражение с помощью

Тогда

где - решение линейного дифференциального уравнения.

Дальнейшее решение зависит от численного метода, с помощью которого решается линейное уравнение. Однако точность решения также зависит от выбранного метода

Решая нелинейное дифференциальное уравнение метод квазилинеаризации, мы пытаемся найти траекторию движения функции на заданном отрезке, а по результатам итераций получаем таблицы со значениями функции и её производной в различных точках на этом отрезке. Количество точек зависит от количества разбиений отрезка.

### **3.2 Алгоритм решения**

Чтобы найти решение краевой задачи для нелинейного дифференциального уравнения необходимо свести имеющеюся задачу (3.1) к виду (3.4). Выбрать начальное приближение для , затем последовательно итерировать до тех пор, пока значения модуля разности значений функции на n и n+1 итерациях не будут меньше заданной точности.

где i – текущее номер разбиения.

*Замечание:* стоит отметить, что точность полученного решения, также может зависеть от начального приближения. Чем лучше подобрано начальное приближение, тем лучше результат.

### **3.3 Пример**

Дана краевая задача

Точное аналитическое решение уравнения

Приведём задачу (3.6) к виду (3.4). Найдём производные для *f(x, y, y’)*

Тогда уравнение примет вид

Преобразуем выражение (3.8)

Возьмём в качестве начального приближения , тогда а также в качестве основного метода для решения линейного уравнения выберем метод стрельбы, с использованием метода Эйлера. Зададим точность для решения равную 0.01. Решая уравнение, получаем таблицы со значениями. Составим таблицу 1, где будут выписаны только каждое 10 значение итераций и выписаны только нулевая, первая и последняя итерации.

Таблица 1 – значения функции на итерациях

|  |  |  |  |
| --- | --- | --- | --- |
| x |  |  | y |
| 0 | 0 | 0 | 0 |
| 0.1 | 0.1 | 0.0258662659 | 0.0439482897 |
| 0.2 | 0.2 | 0.0444256865 | 0.077149026 |
| 0.3 | 0.3 | 0.0563635654 | 0.0999941332 |
| 0.4 | 0.4 | 0.0623009318 | 0.1128612434 |
| 0.5 | 0.5 | 0.0628005691 | 0.116114218 |
| 0.6 | 0.6 | 0.0583724779 | 0.1101036496 |
| 0.7 | 0.7 | 0.049478826 | 0.095167346 |
| 0.8 | 0.8 | 0.0365384348 | 0.071630796 |
| 0.9 | 0.9 | 0.0199308448 | 0.0398076197 |
| 1 | 1 | 0 | 0 |

В первую очередь проверим выполнение краевых условий для задачи (3.6).

Условия выполнены. Сверим полученное решение с аналитическим. Для этого построим два графика. Зелёным цветом обозначен график точного решения, а синим полученное с помощью метода квазилинеаризации. Полученные графики представлены на рисунке 1.

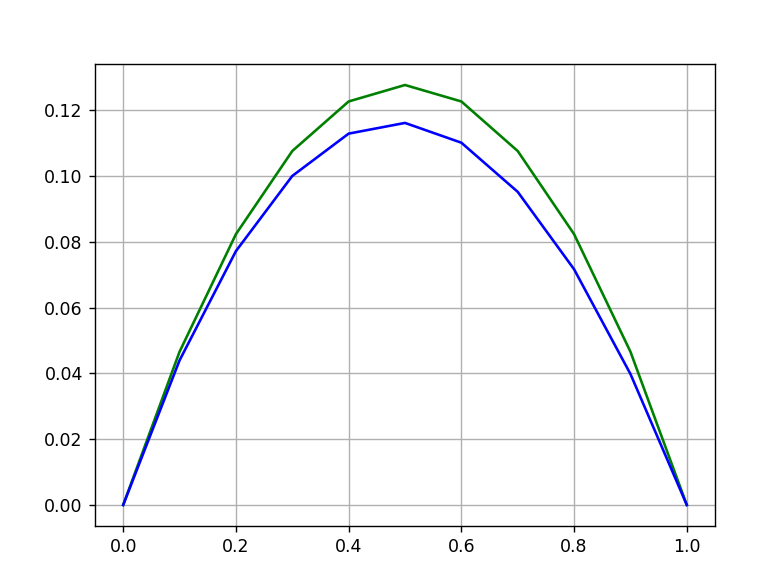


Рисунок 1 – аналитическое и приближённое решения

Как можно заметить, исходя из рисунка 1, полученное решение методом квазилинеаризации достаточно близко к аналитическому. Следовательно задача решена.

Проверим вышеописанное замечание, которое гласит, что точность зависит от того, насколько хорошо подобрано начальное приближение. Повторно решим уравнение, но уже с приближением , тогда . Составим таблицу 2.

Таблица 2 – значения функции на итерациях

|  |  |  |  |
| --- | --- | --- | --- |
| x |  |  | y |
| 0 | 0 | 0 | 0 |
| 0.1 | 0 | 0.045 | 0.0442302123 |
| 0.2 | 0 | 0.08 | 0.0777090491 |
| 0.3 | 0 | 0.105 | 0.1008034709 |
| 0.4 | 0 | 0.12 | 0.1138679137 |
| 0.5 | 0 | 0.125 | 0.117244716 |
| 0.6 | 0 | 0.12 | 0.1112645314 |
| 0.7 | 0 | 0.105 | 0.0962467283 |
| 0.8 | 0 | 0.08 | 0.072499774 |
| 0.9 | 0 | 0.045 | 0.0403216079 |
| 1 | 0 | 0 | 0 |

Как и в прошлый раз, построим два графика, а также добавим к ним график из полученного решения с приближением и сделаем вывод о точности решения. Зелёным цветом обозначен график аналитического решения, синим – график решения с начальным приближением , красным – график с начальным приближением . Полученные графики представлены на рисунке 2.

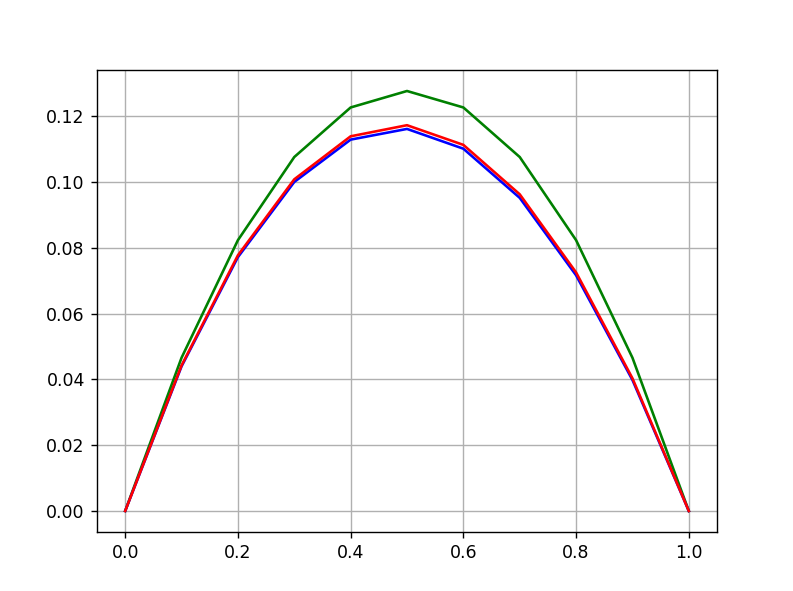


Рисунок 2 – графики решений

Обращаясь к рисунку 2, можем заметить, что при начальном приближении равном 0, точность полученного решения немного больше.

## **4 Программная реализация**

В связи с тем, что метод квазилинеаризации может потребовать довольно больших математических вычислений, в качестве основного языка программирования выбран Python, где реализовано множество математических библиотек. Также исходя из возможностей реализации и последующего улучшения кода, выбран фреймворк Django для работы с веб-интерфейсом.

### **4.1 Метод**

Для программной реализации метода, выбран метод стрельбы, использующий метод Эйлера. Внутри кода описан класс Shooting, реализующий метод стрельбы. Класс содержит 9 различных методов:

    def \_\_init\_\_(self, f1, f2, x0, nu, a0, b0, a1, b1, A, B, a, b, e=0.01) -> None:

        self.f1 = f1

        self.f2 = f2

        self.x0 = x0

        self.nu = nu

        self.a0 = a0

        self.b0 = b0

        self.a1 = a1

        self.b1 = b1

        self.A = A

        self.B = B

        self.a = a

        self.b = b

        self.e = e

        self.m = int(abs(b - a) / e)

        self.h = e

Метод \_\_init\_\_(self) – конструктор класса. Имеет двенадцать обязательных параметров для своей реализации и один необязательный, который будет равен 0.01, в случае если не задан.

def \_\_str\_\_(self) -> str:

        return f"y\' = {self.f1}\nz\' = {self.f2}\n{self.a0}y({self.a}) + {self.b0}y\'({self.a}) = {self.A}\n{self.a1}y({self.b}) + {self.b1}y\'({self.b}) = {self.B}"

Метод \_\_str\_\_(self), отвечающий за строковое представление вводимой задачи. Возвращает задачу в общем виде в формате str.

def \_\_f(self, func, x, y, z) -> any:

        f = eval(func)

        return f

Метод \_\_f(self, func, x, y, z) преобразует введённую функцию в математическое выражение и вычисляет в соответствии со значениями *x, y, z*.

    def \_\_euler(self, nu) -> tuple:

        self.x = [round(self.x0, 10)]

        if self.b0 == 0:

            self.y = [self.A / self.a0]

            self.z = [nu]

        elif self.b0 != 0:

            self.z = [(self.A - self.a0 \* nu) / self.b0]

            self.y = [nu]

        for i in range(1, self.m + 1):

            self.x.append(round(self.x[0] + self.h \* i, 10))

            self.y.append(round(self.y[i - 1] + self.h \* self.\_\_f(self.f1,

                          self.x[i - 1], self.y[i - 1], self.z[i - 1]), 10))

            self.z.append(round(self.z[i - 1] + self.h \* self.\_\_f(self.f2,

                          self.x[i - 1], self.y[i - 1], self.z[i - 1]), 10))

        return self.y, self.z

Метод \_\_euler(self, nu) является программной реализацией явного метода Эйлера.

    def \_\_f3(self, nu) -> any:

        y, z = self.\_\_euler(nu)

        f = self.a1 \* y[-1] + self.b1 \* z[-1] - self.B

        return f

    def \_\_num(self, nu2, nu1) -> any:

        try:

            return nu2 - ((self.mean(nu2) \* (nu2 - nu1)) / (self.mean(nu2) - self.mean(nu1)))

        except ZeroDivisionError:

            return nu2

    def \_\_start(self) -> any:

        i = 1

        while True:

            if (round(self.nu[i], 10) == round(self.nu[i - 1], 10)):

                del self.nu[-1]

                break

            o = self.\_\_num(self.nu[i], self.nu[i - 1])

            self.nu.append(o)

            i += 1

        return self.nu[-1]

    def mean(self, nu) -> any:

        return self.\_\_f3(nu)

Методы \_\_f3(nu), \_\_num(self, nu2, nu1) и \_\_start(self) являются программной реализацией метода стрельбы с использованием метода Эйлера. В отличие от вышеупомянутых трёх, также используется метод mean, который является общедоступным и по своей сути является доступом к методу \_\_f3.

    def Data(self) -> tuple:

        o = self.\_\_start()

        y, z = self.\_\_euler(o)

        return self.x, y, z

Последний метод, который используется классом – Data(self). Основной его задачей является получить значения функции и значения производной для этой функции в точках, а также сами эти точки.

Также, помимо класса описана функция Condition(), которая является реализацией метода квазилинеаризации.

def Сondition(f1, f2, x0, nu, a0, b0, a1, b1, A, B, a, b, e, q, p, koef = 0):

    Y = [[0 for i in range(int(1 / e) + 1)]]

    k = 1

    l = 1

    flag = True

    while flag:

        f2\_n = sp.simplify(f2)

        f2\_n = f2\_n.subs("q", q)

        f2\_n = f2\_n.subs("p", p)

        f2\_n = str(f2\_n)

        obj = Shooting(f1, f2\_n, x0, nu, a0, b0, a1, b1, A, B, a, b, e)

        x, y, z = obj.Data()

        q = z[k]

        p = y[k]

        Y.append(y)

        for i in range(len(Y[l])):

            if i == len(Y[l]) - 1:

                flag = False

            if abs(Y[l][i] - Y[l - 1][i]) < e:

                continue

            else:

                break

        l += 1

        k += int(1 / (e \* 10))

    if koef == 0:

        koef = int(1 / (e \* 10))

    d = {

        "x": x[::koef],

        "y": y[::koef],

        "z": z[::koef],

    }

    return d

Воспользуемся примером из раздела 3. И воспользуемся программной реализацией, чтобы сравнить результаты. Возьмём в качестве начального приближения . Полученный результат представлен на рисунке 3.



Рисунок 3 – полученный результат

Сравнивая значения в таблице 2 и на рисунке 3, можно сделать вывод, что программная реализация работает корректно.

### **4.2 Сайт с использованием фреймворка Django**

Весь проект разбит на огромное количество составляющих. В этом разделе будут описаны только файлы требующие внимания, где были проведены изменения. Сам проект поделён на файл запуска, и два каталога. В первом содержатся настройки всего проекта, во втором (“main”) находятся основные программы.

Первый каталог состоит из 5 файлов:

1. \_\_init\_\_.py
2. asgi.py
3. settings.py
4. urls.py
5. wsgi.py

В случае нашего проекта файлы “asgi.py” и “wsgi.py” не использовались, так как нет необходимых задач, для которых предназначены эти файлы.

В “setting.py”, помимо стандартного кода, было подключено ещё два приложения и обозначен путь к статическим файлам.

“urls.py” – файл, отвечающий за страницы сайта. В данном случае было принято решение о переносе данного файла в каталог “main”, для более удобной работы. Для этого в “main” создан одноимённый файл, а в код для основного файла добавлена команда include.

from django.urls import path, include

urlpatterns = [

    path(' ', include('main.urls')),

]

Каталог “main” же состоит “admins.py”, “apps.py”, “models.py”, “tests.py”, “urls.py”, “views.py”, а также из двух подкаталогов: static, в котором хранятся статичные данные, например, css файлы, и templates, в котором хранятся html файлы.

Файлы “admins.py”, “apps.py” и “tests.py”, не будут рассматриваться так как не были задействованы во время написания курсовой работы.

Первый файл, требующий внимания - “models.py” (приложение E), так как внутри этого файла находится уже ранее описанный код для метода квазилинеаризации (4.1).

Следующий файл, “views.py” (приложение F). В нём находится информация о том, какие данные нужно получить и вывести на веб страницу, также для работы с этим файлом потребуются html и css файлы, которые находятся в приложении.

from django.shortcuts import render

from .models import \*

def index(request):

    return render(request, "index.html")

def result(request):

    f1 = str(request.GET["f1"])

    f2 = str(request.GET["f2"])

    x0 = int(request.GET["x0"])

    nu = [float(request.GET["nu1"]), float(request.GET["nu2"])]

    a0 = int(request.GET["a0"])

    b0 = int(request.GET["b0"])

    a1 = int(request.GET["a1"])

    b1 = int(request.GET["b1"])

    A = int(request.GET["A"])

    B = int(request.GET["B"])

    a = int(request.GET["a"])

    b = int(request.GET["b"])

    e = float(request.GET["e"])

    q = float(request.GET["q"])

    p = float(request.GET["p"])

    k = int(request.GET["k"])

    d = Сondition(f1, f2, x0, nu, a0, b0, a1, b1, A, B, a, b, e, q, p, k)

    return render(request, 'result.html', {"data": d})

Внутри файла есть две функции: index() и result(). Метод index() передаёт веб страницу “index.html” (приложение B), с помощью метода render(). Функция result() сначала собирает данные из форм, после чего использует раннее описанные метод Condition(), и возвращает данный вместе со страницей “result.html” (приложение С), которые позже на самой странице выводятся в таблицу. Оба html файла используют “base.html” как шаблон (приложение А), а также таблицу стилей “style.css” (приложение D).

Последним будет разобран файл “urls.py”. Суть файла заключается в работе с отдельным страницами сайта и ссылками на них. Без корректного указания ссылок по сайту будет очень трудно переключаться.

from django.contrib import admin

from django.urls import path

from .views import \*

urlpatterns = [

    path(' ', index, name="home"),

    path('result/', result, name="result"),

    path('admin/', admin.site.urls),

]

Метод path() отвечает за вывод страницы в браузере, где первым элементом указывается ссылка на страницу, вторым действие, которое будет выполнено при переходе по этой ссылке, а также дополнительно можно указать название ссылки, чтобы можно было намного проще перемещаться между ними.

Запустим на локальном сервере веб сайт и проверим его работоспособность. На рисунках 3 и 4 представлены страница с формой и результат соответственно.

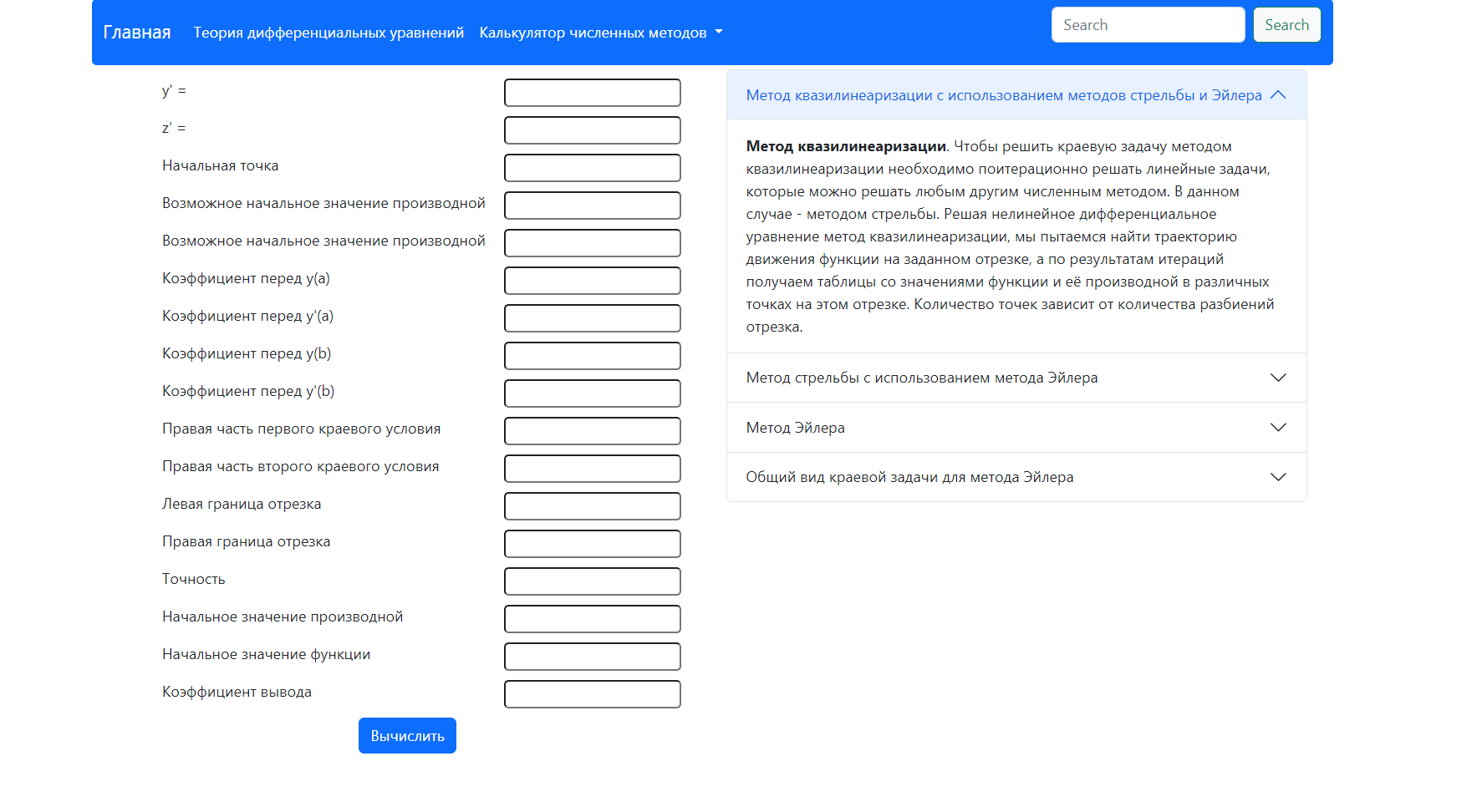


Рисунок 3 – внешний вид сайта

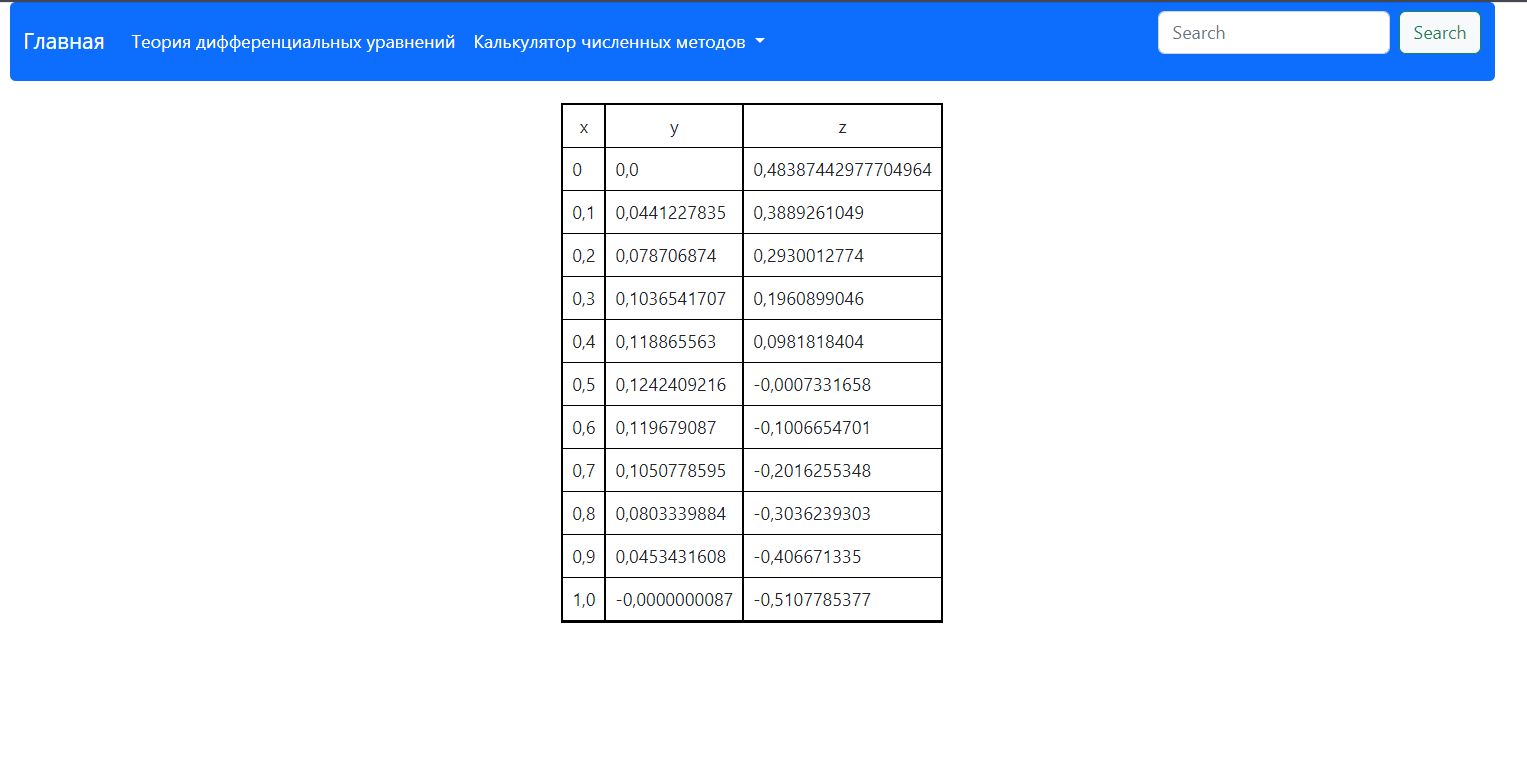


Рисунок 4 – результат работы программы

Как можно заметить полученные данные совпадают с решением уравнения, описанным выше. Следовательно, всё работает корректно.

# **ЗАКЛЮЧЕНИЕ**

В ходе написания курсовой работы был изучен метод квазилинеаризации для решения нелинейных краевых задач. Также во время сбора информации необходимой для выполнения задачи, была выявлена проблема с огромной нехваткой качественной информации, описывающих метод квазилинеаризации, как среди русскоязычной литературы, так и англоязычной. Также, при написании работы были дополнительно изучены другие численные методы, например, метод стрельбы, с помощью которого можно решить задачи такого типа. Стоит отметить, что погрешность данного метода, напрямую зависит от того, какими численными метода решать линейные уравнения, получаемые на каждой итерации. А также, что использование данного метода не является целесообразным, в связи его огромной затратностью и погрешностью, решение такого же уравнения просто методом стрельбы в большинстве случаев будет намного выгоднее и намного точнее.

Веб-интерфейс, описанный в четвёртом разделе имеет огромный список возможностей для улучшения, в связи с чем, является довольно перспективным проектом, если использовать его как онлайн калькулятор по численным метода.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/. – Дата доступа: 12.22.2022.
2. Гусак, А.А. Высшая математика. Учебник для студентов вузов. Т.2. / А.А. Гусак. – Минск: ТетраСистемс, 2004. – 435 с.
3. Петровский, И. Г. Лекции по теории обыкновенных дифференциальных уравнений / И. Г. Петровский. – Москва : «Наука», 1970. – 208 с.
4. Эльсгольц, Л. Э. Дифференциальные уравнения и вариационное исчисление / Л. Э. Эльсгольц. – Москва : «Наука», 1969. – 424 с.
5. Пономарев, К.К. Составление дифференциальных уравнений / К.К. Пономарев, А. А. Витт, С. Э. Хайкин. – Минск : «Вышэйшая школа», 1973. – 560 с.
6. Прохоренок, Н.А. Python 3 PyQt 5 разработка приложений / Н.А. Прохоренок, В.А. Дронов. – Санкт-Петербург : БХВ-Петербург, 2019. – 832 с.
7. Дронов, В.А. Django 3.0. Практика создания веб-сайтов на Python. / В.А. Дронов. – СПб : БХВ-Петербург, 2022. – 704 с.
8. Уфу [Электронный ресурс]. – Режим доступа: https://iit-web-lectures.readthedocs.io/ru/latest/www/html.html. – Дата доступа: 05.12.2023.
9. Крайнов А.Ю., Моисеева К.М. Численные методы решения краевых задач для обыкновенных дифференциальных уравнений : учеб. пособие. – Томск : STT, 2016. – 44 с
10. Беллман, Ричард Е., Квазилинеаризация и нелинейные краевые задачи / Р. E. Беллман, Р. E. Калаба ; Пер. с англ. И. А. Вателя и Ф. И. Ерешко ; Под ред. Ф. Л. Черноусько. - Москва : Мир, 1968. - 183 с.
11. Numerical Methods for Engineers [Электронный ресурс]. – Режим доступа: https://folk.ntnu.no/leifh/teaching/tkt4140/.\_main047.html. – Дата доступа: 05.12.2023.

# **ПРИЛОЖЕНИЕ А**

{% load static %}

{% load django\_bootstrap5 %}

{% bootstrap\_css %}

{% bootstrap\_javascript %}

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="{% static 'css/style.css' %}">

</head>

<nav style="border-radius: 5px;" class="navbar navbar-expand-lg navbar-dark bg-primary container ">

    <div class="container-fluid">

        <a class="navbar-brand" href="{% url 'home' %}">Главная</a>

        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"

            aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

            <span class="navbar-toggler-icon"></span>

        </button>

        <div class="collapse navbar-collapse" id="navbarSupportedContent">

            <ul class="navbar-nav me-auto mb-2 mb-lg-0">

                <li class="nav-item">

                    <a class="nav-link active" aria-current="page" href="#">Теория дифференциальных уравнений</a>

                </li>

                <li class="nav-item dropdown">

                    <a class="nav-link dropdown-toggle active" href="#" id="navbarDropdown" role="button"

                        data-bs-toggle="dropdown" aria-expanded="false">

                        Калькулятор численных методов

                    </a>

                    <ul class="dropdown-menu" aria-labelledby="navbarDropdown">

                        <li><a class="dropdown-item" href="#">Метод Эйлера</a></li>

                        <li><a class="dropdown-item" href="#">Метод Стрельбы с использованием метода Эйлера</a></li>

                        <li><a class="dropdown-item" href="#">Метод Квазилинеаризации с использованием методов стрельбы и Эйлера</a></li>

                    </ul>

                </li>

            </ul>

            <form class="d-flex">

                <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">

                <button class="btn btn-outline-success btn-light" type="submit">Search</button>

            </form>

        </div>

    </div>

</nav>

<body>

    {% block body %}{% endblock %}

</body>

</html>

# ПРИЛОЖЕНИЕ B

{% extends 'base.html' %}

{% block body %}

<div class="container">

    <main class="main">

        <form action="{% url 'result' %}" , class="article" method="get">

            <div class="form">

                <div class="label\_box">

                    <label for="f1">y' = </label>

                    <label for="f2">z' = </label>

                    <label for="x0">Начальная точка</label>

                    <label for="nu1">Возможное начальное значение производной</label>

                    <label for="nu2">Возможное начальное значение производной</label>

                    <label for="a0">Коэффициент перед y(a)</label>

                    <label for="b0">Коэффициент перед y'(a)</label>

                    <label for="a1">Коэффициент перед y(b)</label>

                    <label for="b1">Коэффициент перед y'(b)</label>

                    <label for="A">Правая часть первого краевого условия</label>

                    <label for="B">Правая часть второго краевого условия</label>

                    <label for="a">Левая граница отрезка</label>

                    <label for="b">Правая граница отрезка</label>

                    <label for="e">Точность</label>

                    <label for="q">Начальное значение производной</label>

                    <label for="p">Начальное значение функции</label>

                    <label for="k">Коэффициент вывода</label>

                </div>

                <div class="input\_box">

                    <input name="f1" type="text" class="input" id="f1">

                    <input name="f2" type="text" class="input" id="f2">

                    <input name="x0" type="text" class="input" id="x0">

                    <input name="nu1" type="text" class="input" id="nu1">

                    <input name="nu2" type="text" class="input" id="nu2">

                    <input name="a0" type="text" class="input" id="a0">

                    <input name="b0" type="text" class="input" id="b0">

                    <input name="a1" type="text" class="input" id="a1">

                    <input name="b1" type="text" class="input" id="b1">

                    <input name="A" type="text" class="input" id="A">

                    <input name="B" type="text" class="input" id="B">

                    <input name="a" type="text" class="input" id="a">

                    <input name="b" type="text" class="input" id="b">

                    <input name="e" type="text" class="input" id="e">

                    <input name="q" type="text" class="input" id="q">

                    <input name="p" type="text" class="input" id="p">

                    <input name="k" type="text" class="input" id="k">

                </div>

            </div>

            <div class="button\_box">

                <button type="submit" class="btn btn-primary">Вычислить</button>

            </div>

        </form>

        <div class="article">

            <div class="accordion" id="accordionExample">

                <div class="accordion-item">

                    <h2 class="accordion-header" id="headingOne">

                        <button class="accordion-button" type="button" data-bs-toggle="collapse"

                            data-bs-target="#collapseOne" aria-expanded="true" aria-controls="collapseOne">

                            Метод квазилинеаризации с использованием методов стрельбы и Эйлера

                        </button>

                    </h2>

                    <div id="collapseOne" class="accordion-collapse collapse show" aria-labelledby="headingOne"

                        data-bs-parent="#accordionExample">

                        <div class="accordion-body">

                            <strong>Метод квазилинеаризации</strong>. Чтобы решить краевую задачу методом

                            квазилинеаризации

                            необходимо поитерационно решать линейные задачи, которые можно решать любым другим численным

                            методом.

                            В данном случае - методом стрельбы.

                            Решая нелинейное дифференциальное уравнение метод

                            квазилинеаризации, мы пытаемся найти траекторию движения функции на заданном отрезке, а по

                            результатам итераций получаем таблицы со значениями функции и её производной в различных

                            точках на этом отрезке. Количество точек зависит от количества разбиений отрезка.

                        </div>

                    </div>

                </div>

                <div class="accordion-item">

                    <h2 class="accordion-header" id="headingTwo">

                        <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse"

                            data-bs-target="#collapseTwo" aria-expanded="false" aria-controls="collapseTwo">

                            Метод стрельбы с использованием метода Эйлера

                        </button>

                    </h2>

                    <div id="collapseTwo" class="accordion-collapse collapse" aria-labelledby="headingTwo"

                        data-bs-parent="#accordionExample">

                        <div class="accordion-body">

                            <strong>Метод стрельбы</strong> позволяет решать как нелинейную, так и линейную краевую

                            задачу.

                            Буквально "стреляя", подбирается значение производной функции в начальной точке, после чего

                            получаем задачу Коши. Сведённая краевая задача к задаче Коши зачастую решается довольно

                            просто, уже известными способами.

                            Решая уравнение методом стрельбы получаем траекторию функции на заданном отрезке.

                        </div>

                    </div>

                </div>

                <div class="accordion-item">

                    <h2 class="accordion-header" id="headingThree">

                        <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse"

                            data-bs-target="#collapseThree" aria-expanded="false" aria-controls="collapseThree">

                            Метод Эйлера

                        </button>

                    </h2>

                    <div id="collapseThree" class="accordion-collapse collapse" aria-labelledby="headingThree"

                        data-bs-parent="#accordionExample">

                        <div class="accordion-body">

                            <strong>Метод Эйлера</strong>. Простейший численный метод решения систем обыкновенных

                            дифференциальных уравнений.

                            Метод Эйлера является явным, одношаговым методом первого порядка точности. Он основан на

                            аппроксимации интегральной кривой кусочно-линейной функцией — так называемой ломаной Эйлера.

                        </div>

                    </div>

                </div>

                <div class="accordion-item">

                    <h2 class="accordion-header" id="headingFour">

                        <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse"

                            data-bs-target="#collapseFour" aria-expanded="false" aria-controls="collapseFour">

                            Общий вид краевой задачи для метода Эйлера

                        </button>

                    </h2>

                    <div id="collapseFour" class="accordion-collapse collapse" aria-labelledby="headingFour"

                        data-bs-parent="#accordionExample">

                        <div class="accordion-body">

                            y' = z(x)</br>

                            z' = f(x);</br>

                            a0 \* y(a) + b0 \* z'(a) = A</br>

                            a1 \* y(b) + b1 \* z'(b) = B

                        </div>

                    </div>

                </div>

            </div>

        </div>

    </main>

</div>

{% endblock %}

# ПРИЛОЖЕНИЕ C

{% extends 'base.html' %}

{% load static %}

{% block body %}

<table class="table container" id="table">

    <tbody>

        <tr class="columns x\_column">

            <td class="rows x\_row up\_row">x</td>

            {% for item in data.x %}

                <td class="rows x\_row">{{item}}</td>

            {% endfor %}

        </tr>

        <tr class="columns y\_column">

            <td class="rows y\_row up\_row">y</td>

            {% for item in data.y %}

                <td class="rows y\_row">{{item}}</td>

            {% endfor %}

        </tr>

        <tr class="columns z\_column">

            <td class="rows z\_row up\_row">z</td>

            {% for item in data.z %}

                <td class="rows z\_row">{{item}}</td>

            {% endfor %}

        </tr>

    </tbody>

</table>

{% endblock %}

# ПРИЛОЖЕНИЕ D

body

{

    margin: 0;

}

.main

{

    display: flex;

}

.form

{

    margin: 0 15px 0 15px;

    width: 100%;

    display: flex;

    justify-content: center;

}

.article

{

    width: 50%;

    margin: 5px 15px 0 15px;

}

.label\_box

{

    display: flex;

    margin: 5px;

    height: 100%;

    flex-direction: column;

}

label

{

    height: 30px;

    margin: 5px;

}

.input\_box

{

    display: flex;

    flex-direction: column;

    justify-content: space-around;

    margin: 5px;

}

.input

{

    border-radius: 5px;

    margin: 5px;

    height: 30px;

}

.button\_box

{

    text-align: center;

}

.break

{

    flex-basis: 100%;

    height: 0;

}

.table

{

    margin: 20px auto;

    display: flex;

    justify-content: center;

}

.columns {

    border: 2px solid black;

}

.rows {

    display: flex;

    flex-direction: column;

    padding: 0px 15px 10px 5px;

}

.up\_row

{

    text-align: center;

    border-bottom: 2px solid black;

}

.x\_column {

    border-right: none;

}

.z\_column {

    border-left: none;

}

tbody {

    display: flex;

}

# ПРИЛОЖЕНИЕ E

from django.db import models

import sympy as sp

# Create your models here.

class Shooting:

    def \_\_str\_\_(self) -> str:

        return f"y\' = {self.f1}\nz\' = {self.f2}\n{self.a0}y({self.a}) + {self.b0}y\'({self.a}) = {self.A}\n{self.a1}y({self.b}) + {self.b1}y\'({self.b}) = {self.B}"

    def \_\_f(self, func, x, y, z) -> any:

        f = eval(func)

        return f

    def \_\_f3(self, nu) -> any:

        y, z = self.\_\_euler(nu)

        f = self.a1 \* y[-1] + self.b1 \* z[-1] - self.B

        return f

    def \_\_num(self, nu2, nu1) -> any:

        try:

            return nu2 - ((self.mean(nu2) \* (nu2 - nu1)) / (self.mean(nu2) - self.mean(nu1)))

        except ZeroDivisionError:

            return nu2

    def \_\_start(self) -> any:

        i = 1

        while True:

            if (round(self.nu[i], 10) == round(self.nu[i - 1], 10)):

                del self.nu[-1]

                break

            o = self.\_\_num(self.nu[i], self.nu[i - 1])

            self.nu.append(o)

            i += 1

        return self.nu[-1]

    def \_\_euler(self, nu) -> tuple:

        self.x = [round(self.x0, 10)]

        if self.b0 == 0:

            self.y = [self.A / self.a0]

            self.z = [nu]

        elif self.b0 != 0:

            self.z = [(self.A - self.a0 \* nu) / self.b0]

            self.y = [nu]

        for i in range(1, self.m + 1):

            self.x.append(round(self.x[0] + self.h \* i, 10))

            self.y.append(round(self.y[i - 1] + self.h \* self.\_\_f(self.f1,

                          self.x[i - 1], self.y[i - 1], self.z[i - 1]), 10))

            self.z.append(round(self.z[i - 1] + self.h \* self.\_\_f(self.f2,

                          self.x[i - 1], self.y[i - 1], self.z[i - 1]), 10))

        return self.y, self.z

    def \_\_init\_\_(self, f1, f2, x0, nu, a0, b0, a1, b1, A, B, a, b, e=0.01) -> None:

        self.f1 = f1

        self.f2 = f2

        self.x0 = x0

        self.nu = nu

        self.a0 = a0

        self.b0 = b0

        self.a1 = a1

        self.b1 = b1

        self.A = A

        self.B = B

        self.a = a

        self.b = b

        self.e = e

        self.m = int(abs(b - a) / e)

        self.h = e

    def mean(self, nu) -> any:

        return self.\_\_f3(nu)

    def Data(self) -> tuple:

        o = self.\_\_start()

        y, z = self.\_\_euler(o)

        return self.x, y, z

def Сondition(f1, f2, x0, nu, a0, b0, a1, b1, A, B, a, b, e, q, p, koef = 0):

    Y = [[0 for i in range(int(1 / e) + 1)]]

    k = 1

    l = 1

    flag = True

    while flag:

        f2\_n = sp.simplify(f2)

        f2\_n = f2\_n.subs("q", q)

        f2\_n = f2\_n.subs("p", p)

        f2\_n = str(f2\_n)

        obj = Shooting(f1, f2\_n, x0, nu, a0, b0, a1, b1, A, B, a, b, e)

        x, y, z = obj.Data()

        q = z[k]

        p = y[k]

        Y.append(y)

        for i in range(len(Y[l])):

            if i == len(Y[l]) - 1:

                flag = False

            if abs(Y[l][i] - Y[l - 1][i]) < e:

                continue

            else:

                break

        l += 1

        k += int(1 / (e \* 10))

    if koef == 0:

        koef = int(1 / (e \* 10))

    d = {

        "x": x[::koef],

        "y": y[::koef],

        "z": z[::koef],

    }

    return d

def Main() -> any:

    q = 12

    p = 0

    k = 0

    f1 = "z"

    f2 = "-1 + 0.49 \* (q \*\* 2) - 0.98 \* (q) \* z"

    x0 = 0

    nu = [1, -1]

    a0 = 1

    b0 = 0

    a1 = 1

    b1 = 0

    A = 0

    B = 0

    a = 0

    b = 1

    e = 0.01

    d = Сondition(f1, f2, x0, nu, a0, b0, a1, b1, A, B, a, b, e, q, p, k)

    print(d)

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

    Main()

# ПРИЛОЖЕНИЕ F

from django.shortcuts import render

from .models import \*

# Create your views here.

def index(request):

    return render(request, "index.html")

def result(request):

    f1 = str(request.GET["f1"])

    f2 = str(request.GET["f2"])

    x0 = int(request.GET["x0"])

    nu = [float(request.GET["nu1"]), float(request.GET["nu2"])]

    a0 = int(request.GET["a0"])

    b0 = int(request.GET["b0"])

    a1 = int(request.GET["a1"])

    b1 = int(request.GET["b1"])

    A = int(request.GET["A"])

    B = int(request.GET["B"])

    a = int(request.GET["a"])

    b = int(request.GET["b"])

    e = float(request.GET["e"])

    q = float(request.GET["q"])

    p = float(request.GET["p"])

    k = int(request.GET["k"])

    d = Сondition(f1, f2, x0, nu, a0, b0, a1, b1, A, B, a, b, e, q, p, k)

    return render(request, 'result.html', {"data": d})