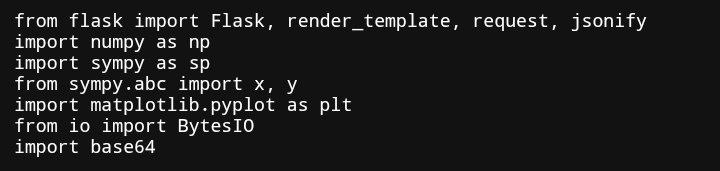
**Вычисление особых точек и их типов дифференциального уравнения**

**Актуальность**

* 1. **Образовательные цели:** Многие студенты сталкиваются с трудностями в изучении дифференциальных уравнений. Интерактивный сайт может помочь в понимании теории, предоставляя пользователям возможность визуализировать и вычислять особые точки и их типы.
* 2. **Автоматизация вычислений:** Ручные расчеты могут быть трудоемкими и подвержены ошибкам. Сайт, который автоматически вычисляет особые точки и их типы, может значительно упростить процесс и повысить точность результатов.
* 3. **Доступность информации:** Такой ресурс может стать центральным местом для получения информации о методах решения дифференциальных уравнений, что особенно важно для студентов, аспирантов и исследователей.

**Какие технологии мы использовали?**

****

**Flask** — это веб-фреймворк для языка программирования Python. Он предназначен для упрощения разработки веб-приложений и сайтов. Flask предлагает простую и гибкую структуру, позволяя разработчикам быстро создавать приложения, а также легко расширять их функциональность с помощью различных расширений.

1. **render\_template**: Это функция позволяет динамически генерировать HTML-страницы, подставляя данные в шаблоны. Например, есть HTML-файл с переменными, мы можем передать данные из нашего приложения, и Flask создаст финальную страницу с этими данными.
2. **request**: Это объект, который содержит данные о текущем HTTP-запросе. С его помощью можно получить информацию о запросе, например, параметры URL, данные формы, заголовки и файлы. Это позволяет обрабатывать входящие данные от пользователя.
3. **jsonify**: Это функция, которая преобразует данные Python (например, словари или списки) в формат JSON и возвращает их как ответ на HTTP-запрос (JSON является стандартным форматом для обмена данными между клиентом и сервером).
4. **NumPy:** Это библиотека для работы с многомерными массивами и матрицами, а также для выполнения различных математических операций над ними.
5. **SymPy:** Это библиотека для символьной математики, которая позволяет решать уравнения и производить интегрирование и дифференцирование в символическом виде.
6. [**sympy.abc**](https://sympy.abc/)**:** Это модуль в библиотеке SymPy, который предоставляет стандартные символы (например, x, y, z и т.д).
7. **Matplotlib:** Это библиотека для создания графиков.
8. **BytesIO**: Это класс из модуля io, который позволяет работать с байтовыми потоками в памяти как с файловыми объектами. Он полезен для работы с данными, которые нужно обрабатывать как файл, не записывая их на диск.
9. **base64:** Это модуль для кодирования и декодирования данных в формат Base64. Base64 используется для представления двоичных данных в текстовом формате, что удобно для передачи через текстовые протоколы, такие как HTTP.

**Разбор кода**

**Код 1:** функция analyze\_critical\_points, которая предназначена для анализа особых точек системы дифференциальных уравнений.

Здесь определяется функция analyze\_critical\_points, которая принимает два аргумента: f\_expression и g\_expression. Эти аргументы представляют собой математические выражения, задающие систему уравнений.

Блок try используется для обработки возможных ошибок, которые могут возникнуть при выполнении кода внутри него.  
Преобразование выражений в символьные объекты:

f = sp.sympify(f\_expression)

g = sp.sympify(g\_expression)

(Здесь используется функция sympify из библиотеки SymPy (sp), чтобы преобразовать входные строки (или другие форматы) в символьные выражения, которые можно использовать для дальнейших математических операций.)

Поиск особых точек: critical\_points = sp.solve([f, g], (x, y), dict=True)

В этой строке происходит решение системы уравнений f = 0 и g = 0 . Функция solve ищет значения переменных x и y , при которых оба выражения равны нулю. Параметр dict=True указывает, что результаты должны быть возвращены в виде списка словарей, где каждый словарь содержит соответствующие значения переменных.  
Инициализация списка результатов: results = []

**Код 2:** функция classify\_critical\_point предназначена для классификации особой точки системы дифференциальных уравнений на основе собственных значений.  
Определение функции: здесь определяется функция classify\_critical\_point, которая принимает один аргумент eigenvalues. Этот аргумент представляет собой словарь, где ключами являются собственные значения, а значениями — соответствующие им характеристики (например, их кратности или другие параметры). Проверка количества собственных значений: if len(evals) < 1:

return "Нет собственных значений" (Здесь проверяется, есть ли хотя бы одно собственное значение. Если их нет (длина списка evals меньше 1), функция возвращает сообщение "Нет собственных значений".) Извлечение собственных значений: lambda1 = eigenvalues[evals[0]]; lambda2 = eigenvalues[evals[1]] if len(evals) > 1 else lambda1

**Код 3:** функция plot\_phase\_portrait предназначена для построения фазового портрета системы дифференциальных уравнений. Определение функции:

def plot\_phase\_portrait(f\_expr, g\_expr, points, x\_range=(-5, 5), y\_range=(-5, 5)):

Здесь определяется функция plot\_phase\_portrait, которая принимает несколько аргументов:  
   • f\_expr: выражение для первой производной (или функции) системы.  
   • g\_expr: выражение для второй производной (или функции) системы.  
   • points: начальные точки для построения траекторий   
   • x\_range: диапазон значений по оси X (по умолчанию от -5 до 5).  
   • y\_range: диапазон значений по оси Y (по умолчанию от -5 до 5).  
  
Здесь начинается блок try, который используется для обработки возможных исключений, которые могут возникнуть при выполнении кода внутри него.

Создание функций с помощью lambdify: f\_func = sp.lambdify((x, y), f\_expr, 'numpy'); g\_func = sp.lambdify((x, y), g\_expr, 'numpy')

• sp.lambdify — это функция из библиотеки SymPy, которая преобразует символьные выражения (f\_expr и g\_expr) в функции, которые можно вызывать с помощью NumPy.   
• f\_func и g\_func теперь являются функциями, которые принимают два аргумента (x и y) и возвращают соответствующие значения.

Создание сетки значений для координат:

x\_vals = np.linspace(x\_range[0], x\_range[1], 20)

y\_vals = np.linspace(y\_range[0], y\_range[1], 20)

X, Y = np.meshgrid(x\_vals, y\_vals)

Вычисление значений векторного поля:

U = f\_func(X, Y)

V = g\_func(X, Y)

Здесь вычисляются значения векторного поля для каждой точки сетки. U будет содержать значения функции f\_func, а V — значения функции g\_func.  
В результате мы получаем два двумерных массива: U и V, которые представляют собой компоненты векторов на фазовом портрете.