Министерство образования и науки Российской Федерации
ФГАОУ ВПО «УрФУ имени первого Президента России Б. Н. Ельцина»
Институт радиоэлектроники и информационных технологий - РтФ
Департамент информационных технологий и автоматики

Исследование предельных циклов нелиненой системы

ОТЧЕТ
по лабораторной работе

Преподаватель: Пименов Владимир Германович
Студент: Сухоплюев Илья Владимирович
Группа: РИ-440001

Екатеринбург
2017

Отчет

Работа описывает исследование параметризованной нелинейной системы на наличие предельных циклов. В ходе исследования системы, рассматриваются следующие вопросы: нахождение параметра системы, при котором наблюдается предельный цикл; поиск параметра, где наблюдается бифуркация поведения системы; исследование свойств обнаруженного предельного цикла; определение характера его устойчивости и исследование влияния разных видов запаздывания на систему (постоянное, переменное и непрерывное запаздывание). Данные задачи изучаются путем проведения численных экспериментов с помощью интерпретатора Python 3.5 и математических библиотек (numpy[1], matplotlib[2]).

Содержание

1 Поиск предельного цикла

Рассмотрим исследуемую систему (уравнение 1.1, ν - параметр системы). Она описывается уравнением от одной фазовой переменной x. Уравнение дифференциальное, второго порядка и, в силу слагаемого $3\dot{x}^3$, нелинейное. Решение такого уравнения аналитическими методами является довольно сложной задачей, поэтому нашим методом исследования будет построение численных экспериментов, описывающих данную систему при определенном параметре ν.

$$\ddot{x} + 3\dot{x}^3 - \nu\dot{x} + x = 0 \tag{1.1}$$

Однако, в таком виде уравнение 1.1 не является удобным для моделирования. Поэтому приведем его к канонической форме от двух переменных, с помощью замены 1.2, получив уравнение от двух фазовых переменных y_1 и y_2 (Система уравнений 1.3). В дальнейшем, мы будем пользоваться описанием нашей системы именно в таком виде.

$$\begin{cases} y_1 = x \\ y_2 = \dot{x} \end{cases} \tag{1.2}$$

$$\begin{cases} \dot{y_1} = y_2 \\ \dot{y_2} = -3y_2^3 + \nu y_2 - y_1 \end{cases} \tag{1.3}$$

Преобразовав систему к удобному для нас виду, перейдем к первой части работы – нахождения такого параметра ν, при котором наблюдается предельный цикл.

Для начала, дадим определение искомому объекту.

Определение 1 *Предельным циклом будем называть замкнутую изолированную траекторию в фазовом пространстве, подразумевая замкнутость в смысле периодичности поведения системы.*

Таким образом, нам нужно построить фазовый портрет нашей системы, на котором нужно будет обнаружить искомую замкнутую линию. Для этого, зная зависимость значения производных от их координат, можно с помощью функции *streamplot*[3] построить фазовый портрет (Программа 1, в качестве параметра для начала возьмем $\nu = 1$).

Листинг 1 Построение фазового портрета

```
# Подключение используемых библиотек
# В дальнейшем является постоянным и опускается в листингах
# Полный исходный код программы можно найти в приложении
import matplotlib.pyplot as plt
import numpy as np

# Параметр системы
nu = 1

# создание сетки 100x100 точек в области [-3;3]x[-3;3]
Y, X = np.mgrid[-3:3:100j, -3:3:100j]

# вычисление фазовых векторов на сетке
Y1 = Y
Y2 = -3 * Y ** 3 + nu * Y - X

# построение фазового портрета
fig0, ax0 = plt.subplots()
plt.streamplot(X, Y, Y1, Y2)

# показать построенные графики (опускается в дальнейшем)
plt.show()
```

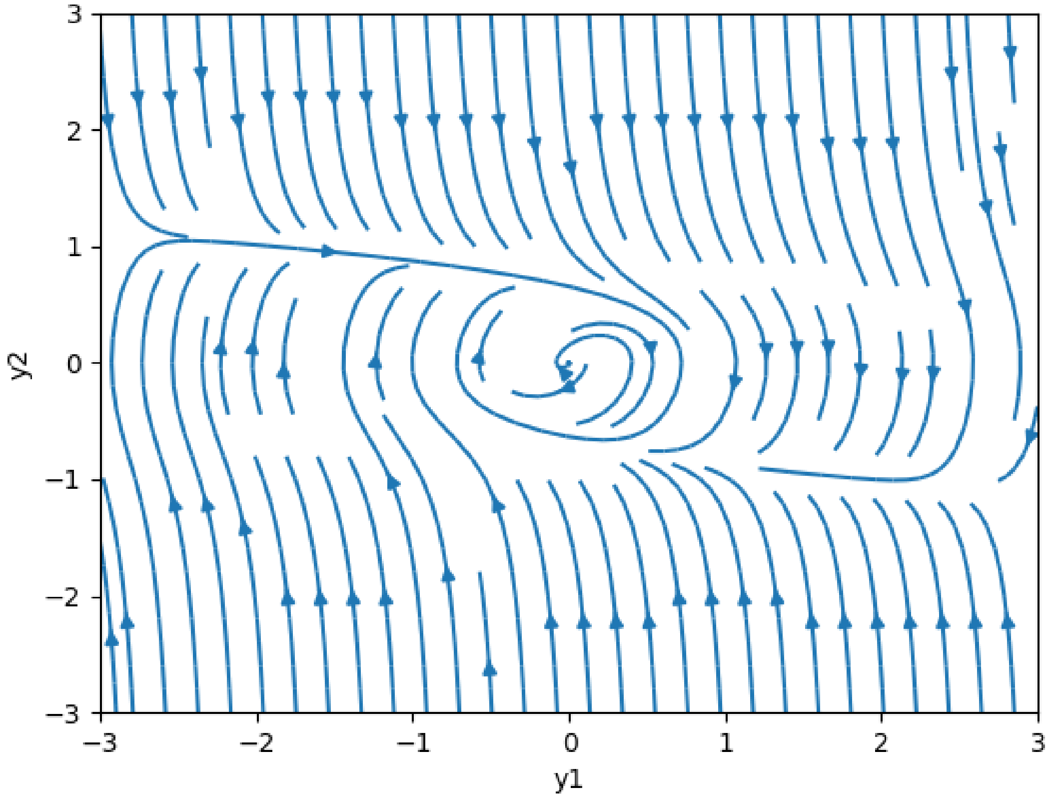

График 1.1 — Поиск предельного цикла построением фазового портрета

На графике 1.1 изображен результат работы нашей программы. В данном случае значение параметра оказалось оптимальным: можно видеть, как изоклины сходятся к наклоненному прямоугольнику в центре графика.

Теперь, чтобы убедится наверняка, что траектории сходятся вокруг этого цикла и там нет разрывов, построим две линии методом Эйлера снаружи и внутри наблюдаемого цикла (Программа 2).

Листинг 2 Использование метода Эйлера для проверки предельного цикла

```
# функция построение кривой методом Эйлера
def line(y1_0, y2_0):
    y1 = [y1_0]
    y2 = [y2_0]
    h = 0.01 # длина шага
    for i in range(2000): # 2000 - количество итераций
        y1.append(y1[i] + h*(y2[i]))
        y2.append(y2[i] + h*(-3*y2[i] ** 3 + nu*y2[i] - y1[i]))
    # отображение кривой на графике
    ax0.plot(y1, y2)

# построение двух кривых, начинающихся внутри и
# вне предполагаемого предельного цикла
line(0.1, 0.1)
line(2, 2)
```

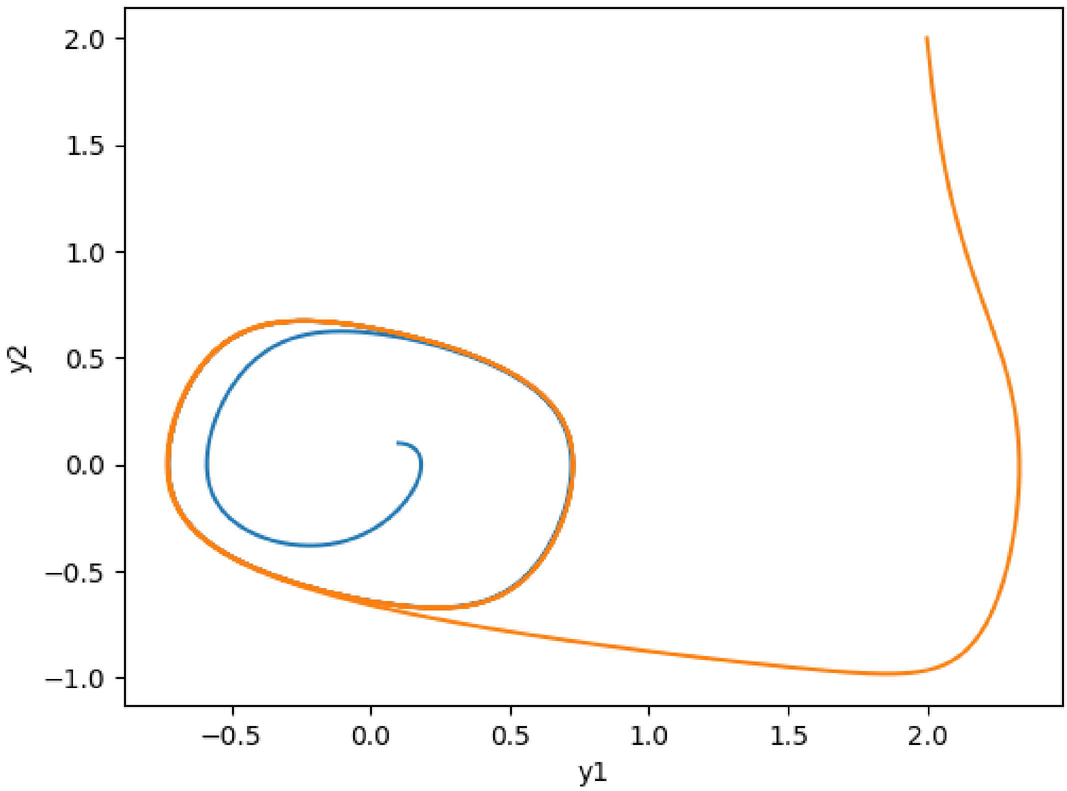

График 1.2 — Обнаружение аттрактора методом Эйлера

На рисунке 1.2 мы можем видеть две линии, начинающиеся из точек $(0.1, 0.1)$ и $(2, 2)$. Эти линии сходятся сближаются к искомому предельному циклу системы.

2 Поиск точек бифуркаций

Найдя предельный цикл в системе 1.2, мы можем перейти к следующему этапу нашего исследования – определения всех значений параметра, при которых наблюдается данный цикл.

В силу того, что наша система рассматривается дифференциальным уравнением, поведение системы будет меняться плавно на промежутках, разделенных так называемыми, точками *бифуркации*(точки, в которых происходит изменение поведения системы).

Определение 2 *Точка бифуркации - значение параметра системы, при котором наблюдается качественное изменение поведения системы.*

Чтобы нам было удобно наблюдать изменение системы от параметра без перезапуска программы, мы обернем построения в функцию и добавим в нашу программу слайдер – бегунок, которым можно будет менять значение параметра ν.

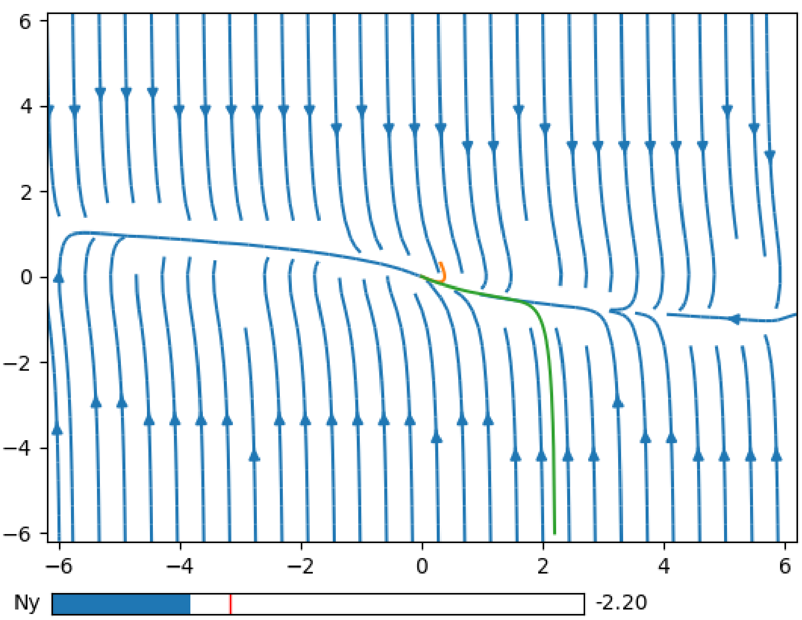

График 2.1 — Стационарная точка системы при $\nu = -2.2$

Начиная с отрицательных значений (от -10) мы наблюдаем сильное стремление к центру координат – стационарной точки системы (График 2.1).

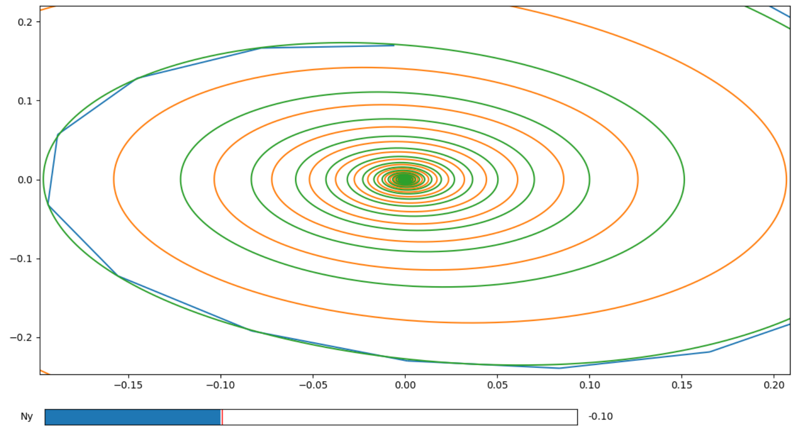

График 2.2 — Стационарная точка системы при $\nu = -0.1$

При приближении параметра к нулю, поведение системы искривляется в овальную форму, но линии медленно сходятся к нулю (График 2.2).

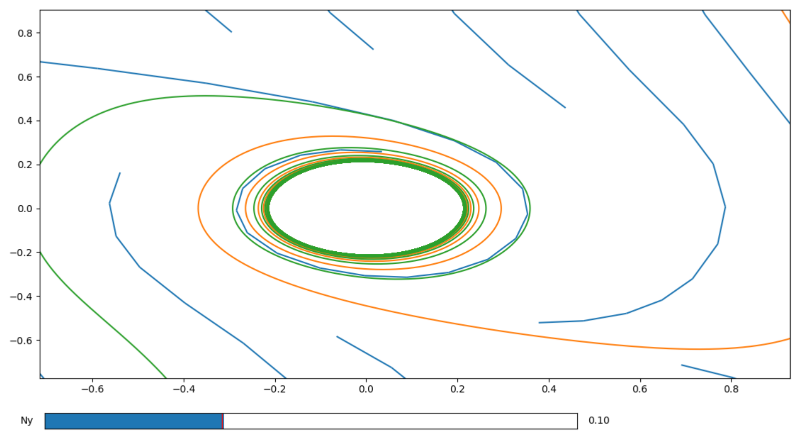

График 2.3 — Появление цикла при $\nu = 0.1$

Как только мы переступаем нулевое значение параметра, наши траектории останавливаются значительно раньше – мы начинаем наблюдать знакомый нам предельный цикл, но в меньших размерах (График 2.3).

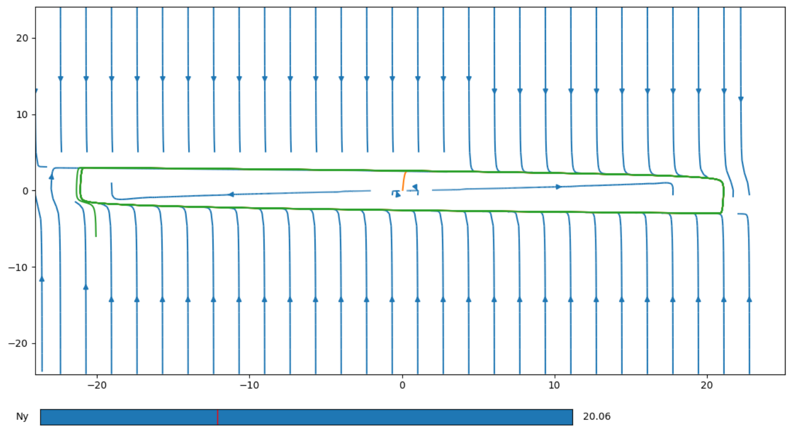

График 2.4 — Расширение предельного цикла при увеличении параметра ($\nu = 20$)

Увеличивая ν дальше, остается наблюдать за ростом цикла (График 2.4).

Из полученных наблюдений можно выдвинуть гипотезу: на отрицательной полуоси исследуемая система сходится в стационарную точку; в положительной же оси наблюдается предельный цикл, который увеличивается в зависимости от параметра системы.

Стоит отметить, что наличие или отсутствие предельного цикла на границе ($\nu = 0$) мы выявить не можем, так как при приближении к параметра к нулю, чтобы быть уверенным в наличии стационарной точки или цикла, приходится увеличивать точность вычислений. В конце концов, когда точность увеличить не удается, нам остается только предполагать: толи линии сошлись к циклу, толи они не достигли нуля из-за недостаточного кол-ва шагов в методе Эйлера.

С учетом этого замечания, можно выдвинуть еще одну гипотезу: так как изменение поведения в системе происходит настолько плавно, что нам не удается уловить момент, когда мы наблюдаем стационарную точку, а когда предельный цикл. То есть мы можем говорить, что наблюдается *мягкая бифуркация системы.*

3 Исследование свойств предельного цикла

Следующим шагом в исследовании системы станет изучение свойств нашего предельного цикла при конкретном значении параметра (возьмем $\nu = 1$): нахождение его периода (от независимой переменной) и его форму. Данные свойства потребуются в следующих частях (4 и 5) для проверки характера его устойчивости.

Ставя численные эксперименты, значения могут получатся точные, но все же с погрешностью. Поэтому далее мы будем находить значение с точностью до 3-х знаков после запятой (т. е. наше значение должно расходится не более чем на $\epsilon = 0.5 * 10^{-4}$).

В программе 3 строится цикл методом точечных отображений Пуанкаре: выбирается точка на оси 0_{y1}, от которой мы начинаем двигаться по траектории до тех пор, пока снова не пересечет ось. При приближении к нашему предельному циклу, точки будут сближаться все больше и больше. Таким образом будем считать траекторию предельным циклом, когда начальная и конечная точка сблизятся по обоим координатам ближе чем на ϵ. Периодом нашего цикла будет количество затраченных шагов $(i + 1)$ помноженных на длину шага h. Как видно из работы программы, цикл имеет период $\omega = 6.663$.

Далее можно попытаться найти аналитическую форму данного цикла, но судя по графику 1.2 форма цикла не похожа на знакомые квадратичные функции и подбор аналитического вида кривой может оказаться трудной задачей. При этом мы не сможем достигнуть такой же точности, как наше поточечное решение, полученное методом Эйлера. Поэтому в следующих работах будем работать с массивами $y1$ и $y2$, описывающие наш цикл.

Листинг 3 Поиск параметров системы

```
eps = 0.5 * 10 ** -4
y1_0, y2_0 = 0.72424, 0 # начальная точка

y1 = [y1_0]
y2 = [y2_0]
h = 0.0001
for i in range(100000):
    # итерация метода Эйлера
    y1.append(y1[i] + h*(y2[i]))
    y2.append(y2[i] + h*(-3*y2[i] ** 3 + ny * y2[i] - y1[i]))
    # проверка прихода в туже точку с погрешностью
    if  np.abs(y1_0 - y1[i+1]) < eps and
        np.abs(y2_0 - y2[i+1]) < eps:
        # вывод результатов
        print("h={h}, i={i}, h*i={period}".format(
              h=h, i=i+1, period=h*(i+1)))
        return;
# Вывод программы
# h=0.0001, i=66633, h*i=6.6633000000000004
```

4 Определение устойчивости через мультипликаторы системы

В этом разделе будет проведено исследование нашего предельного цикла на асимптотическую орбитальную устойчивость. Проверку будем проводить с помощью аналога теоремы Андронова-Витта, вычислив мультипликаторы системы первого приближения вдоль исследуемого предельного цикла.

Дадим необходимые для этого определения.

Определение 3 *Система орбитально устойчива, если для его решения $\phi(t)$ и для любого $\epsilon > 0$ найдется постоянное число $\delta = \delta(\epsilon) > 0$, такое, что траектория всякого решения $X(t)$, начинающегося в δ-окрестности траектории $\phi(t)$, остается в ϵ-окрестности траектории $\phi(t)$ при всех $t \geqslant 0$.*

Определение 4 *Система асимптотически орбитально устойчива, если для орбитального решения $\phi(t)$ и всякого близкого к нему решения $X(t)$ выполняется условие 4.1. То есть любая достаточно близкая траектория сходится к орбитально устойчивому решению.*

$$\exists\alpha, \beta > 0 : \|\mathbf{X}(0) - \boldsymbol{\varphi}(0)\| < \delta \to \|\mathbf{X}(t) - \boldsymbol{\varphi}(t)\| \leq \alpha \|\mathbf{X}(0) - \boldsymbol{\varphi}(0)\| e^{-\beta t} \tag{4.1}$$

Из поставленных определений видна мотивация исследования решения на наличие асимптотической орбитальной устойчивости: подтвердив его, мы, увеличивая точность вычислений, можем быть уверены, что лучше приближаемся к необходимой траектории.

В этом нам поможет, аналог *теоремы Андронова-Витта.*

Теорема 1 ***теорема Андронова-Витта***. *Если имеется периодическое решение автономной системы и его система первого приближения имеет два мультипликатора, один равный единице, а второй по-модулю меньше единицы, то полученное периодическое решение асимптотически орбитально устойчиво.*

Мультипликаторами называются значения величин, полученных в результате алгоритма, изученного на лекционных занятиях:

— Рассмотрим систему $\dot{x} = F(x)$ и периодическое решение $\eta(t)$;

— Выразим линеаризированную систему $\dot{y} = F'(\eta(t))y$ вдоль данного решения;

— Вычислим ее вдоль периодического решения с н.у. (1, 0) и (0, 1);

— Получим матрицу монодромии $\Phi = (\phi_1, \phi_2)$, где ϕ_1, ϕ_2 - решения системы полученные на предыдущем шаге.

— Собственные числа матрицы монодромии мы и будем считать мультипликаторами системы.

Осталось, провести поставленные шаги. Рассмотрим нашу систему 4.2:

$$\begin{cases} \dot{y_1} = y_2 = f_1(y_1, y_2) \\ \dot{y_2} = -3y_2^3 + \nu y_2 - y_1 = f_2(y_1, y_2) \end{cases} \tag{4.2}$$

Посчитаем Якобиан нашей системы (уравнения 4.3):

$$\frac{\partial f_1}{\partial y_1} = 0; \qquad \frac{\partial f_1}{\partial y_2} = 1; \qquad \frac{\partial f_2}{\partial y_1} = -1; \qquad \frac{\partial f_2}{\partial y_2} = -9y_2^2 + \nu; \tag{4.3}$$

И строим линеаризированную систему вдоль цикла $\eta(t)$:

$$\begin{cases} \dot{Y_1} = Y_2 \\ \dot{Y_2} = -Y_1 + (-9\eta_2^2(t) + \nu)Y_2 \end{cases} \tag{4.4}$$

В листинге 4 мы, методом Эйлера решаем полученную систему, находим матрицу монодромии и вычисляем ее Собственные числа (Eigenvalues). Как видно из вывода программы, вычисленные мультипликаторы удовлетворяют условиям рассмотренной теоремы с искомой точностью до 3-х знаков ($8.59 * 10^{-4}$ и 1), таким исследуемый цикл асимптотически орбитально устойчив.

Листинг 4 Вычисление мультипликаторов

```
def linear_form(y1_0, y2_0, cycle_y1, cycle_y2):
    # Решение линеаризованной системы вдоль цикла
    y1 = [y1_0]
    y2 = [y2_0]
    for i in range(len(cycle_y1)):
        y1.append(y1[i] + h * (y2[i]))
        y2.append(y2[i] + h * (-y1[i] +
                       (-9*cycle_y2[i] ** 2 + ny) * y2[i]))
    return [y1[-1], y2[-1]]

cycle_y1, cycle_y2 = line(0.72424, 0) # вычисление цикла
# решение линеаризированной системы с н. у. (1, 0) и (0, 1)
f1 = linear_form(1, 0, cycle_y1, cycle_y2)
f2 = linear_form(0, 1, cycle_y1, cycle_y2)
f = np.array([ # Матрица монодромии
    [f1[0], f2[0]],
    [f1[1], f2[1]],
])
print("F-matrix:")
print(f)
# Вычисление собственных чисел матрицы монодромии
p = np.linalg.eig(f)
print("Eigenvalues:")
print(p[0])
# Вывод программы:
# F-matrix:
# [[  8.56578243e-04  -3.73826069e-06]
#  [  5.05107751e-01   1.00012266e+00]]
# Eigenvalues:
# [  8.58467858e-04   1.00012077e+00]
```

5 Определение устойчивости с помощью метода Пуанкаре

В этом разделе, рассмотрим другой метод определения асимптотической орбитальной устойчивости циклического решения, предложенный Анри Пуанкаре. Он определен только для двумерных автономных систем, что нам подходит.

Теорема 2 *Теорема Пуанкаре. периодическое решение системы будет асимптотически орбитально устойчивым тогда и только тогда, когда интеграл дивергенции вдоль решения будет меньше нуля.*

Таким образом, нам остается лишь взять вычисленные в уравнении 4.3 производные, составляющие дивергенцию системы. После чего вычислить интеграл (возьмем метод прямоугольников). Программа 5 производит данное вычисление, и результат подтверждает полученный в предыдущей работе результат: предельный цикл удовлетворяет критерию Пуанкаре, значит асимптотически орбитально устойчив.

Листинг 5 Вычисление интеграла от дивергенции системы

```
def integral_from_div(cycle_y1, cycle_y2):
    """
    Вычисление интеграла от дивергенции системы
    вдоль цикла
    """
    sum = 0
    for j in range(len(cycle_y1)):
        sum += -9 * cycle_y2[j] ** 2 + ny
    sum *= h
    return sum
# Основная программа
cycle_y1, cycle_y2 = line(0.724197, 0)
s = integral_from_div(cycle_y1, cycle_y2)
print("Integral of the divergence: {}".format(s))
# Вывод программы:
# Integral of the divergence: -7.059887304427718
```

6 Влияние постоянного запаздывания на систему

С этой главы мы познакомимся с *функциональными дифференциальными уравнениями (ФДУ)*, а точнее с системами в которых присутствует элемент запаздывания и как он влияет на поведение исследуемой системы.

Определение 5 *Функциональным дифференциальным уравнением с постоянным запаздыванием, мы будем называть уравнение вида 6.3.*

$$\begin{cases} \dot{x} = f(t, x, x_t(t-\tau)) \\ x(0) = x_0 \\ x_{t_0}(\cdot) = \Big\{ y_0(t); t \in [\tau; 0) \Big\} \end{cases} \tag{6.1}$$

Посмотрев на него, можно выделить основные особенности от обычных дифференциальных уравнений, помимо зависимости от t и x, у нас появляется зависимость от значения фазовой переменной в момент $x_t(t-\tau)$ (Здесь и далее, x_t - *функция предыстории*, описывающая значения фазовой переменной на интервале $[t-\tau; t)$). Так же, чтобы получить однозначное решение системы 6.3, понадобилось задать предысторию до момента $t_0 = 0$ (третье равенство системы).

6.1 Добавление запаздывания по y_1 в систему

Проведем первый эксперимент с постоянным запаздыванием. Для этого, введем в нашу систему слагаемое $\alpha * y_1(t-\tau)$ (Ур. 6.2) При параметре $\alpha = 0$ оно не влияет на систему, следовательно мы будем наблюдать наш предельный цикл. Постепенно изменяя этот параметр мы будем наблюдать на изменения характера системы.

$$\begin{cases} \dot{y_1} = y_2 + \alpha * y_1(t-\tau) \\ \dot{y_2} = -3y_2^3 \; + \nu y_2 - y_1 \end{cases} \tag{6.2}$$

В программе 6 показано основное изменения нашего эксперимента. Теперь во входные параметры мы уже передаем не одну точку, а массив размера len(y1_0), и при вычислении следующего шага в методе Эйлера, мы обращаемся к (i - len(y1_0) + 1)-му элементу массива.

В эксперименте рассматриваются два начальных условия: система находилась в точке (0.1, 0.1) в 100 первых итерациях, и в точке (2, 2) в 100 последних итерациях.

Также стоит отметить, модернизацию нашей программы в техническом виде, с помощью слайдера оказалось неудобным отлаживать подбор необходимого параметра. Поэтому, чтобы упростить эксперименты и не перезапускать программу, вместо слайдера был поставлен TextBox - поле для ввода текста, куда мы можем ввести любое, необходимое нам, значение (пересчет графика произойдет при нажатии на клавишу Enter). При желании, с подробностями использования этого виджета можно ознакомиться в приложении.

Листинг 6 Программирование постоянной задержки

```
def line(y1_0, y2_0, alpha):
    # решение задачи методом Эйлера
    y1 = copy.deepcopy(y1_0)
    y2 = copy.deepcopy(y2_0)
    h = 0.03
    for i in range(len(y1_0)-1, 20000):
        # Запаздывание влияет на y1
        y1.append(y1[i] + h * (y2[i] +
                                 alpha * y1[i - len(y1_0) + 1]))
        y2.append(y2[i] +
                       h * (-3*y2[i] ** 3 + ny * y2[i] - y1[i]))
    ax.plot(y1, y2)

# ...
# Расчет примеров
line([0.1 for a in range(100)],
     [0.1 for a in range(100)],
     alpha)
line([2 for a in range(100)],
     [2 for a in range(100)],
     alpha)
```

Сразу стоит отметить, что из-за внесения задержки, система начинала приобретать непредсказуемый характер. В некоторых случаях это вызывало переполнение переменных, что мешало пронаблюдать эксперимент.

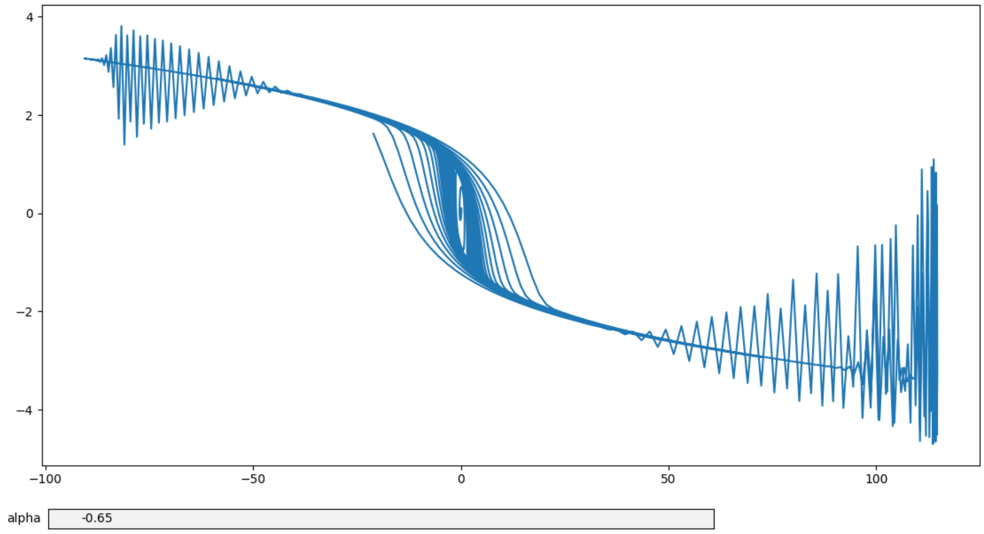

График 6.1 — ($\alpha = -0.65$) Решение от точки (0.1, 0.1) хаотично расходится от центра. Решение от точки (2, 2) не смогло построится

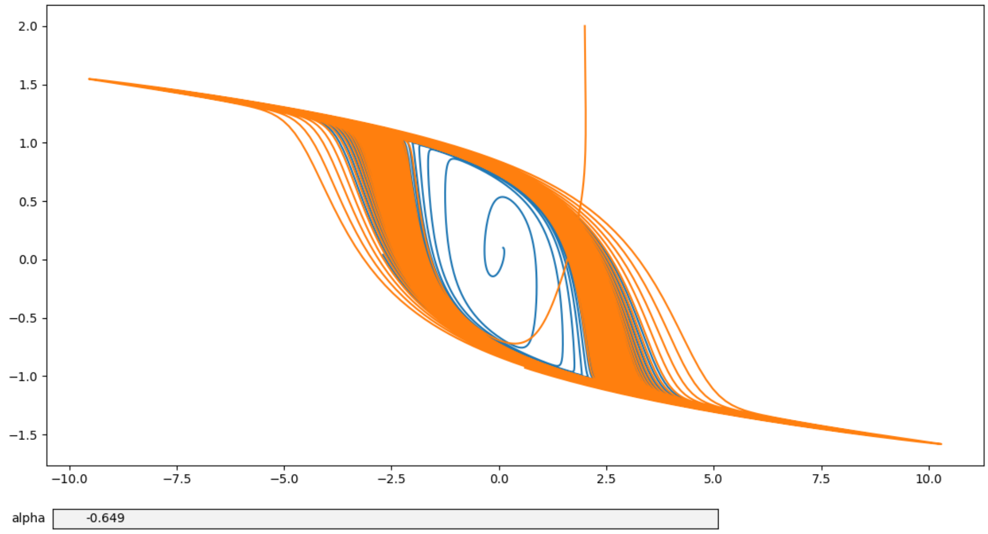

График 6.2 — ($\alpha = -0.649$) Оба решения построились. Отчетливо наблюдается жесткость системы, цикл сохранился, но об изолированности говорить не приходится

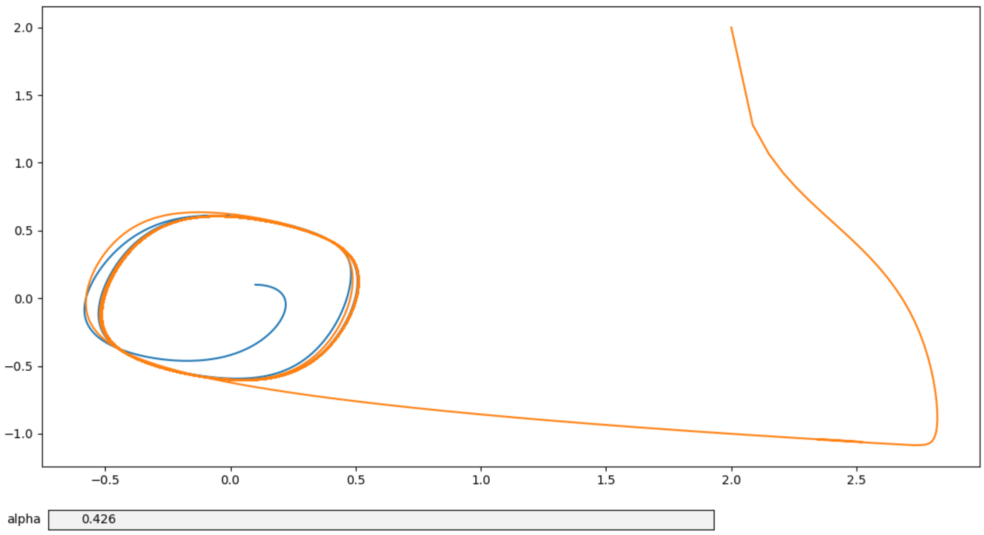

График 6.3 — ($\alpha = 0.426$) Решение все еще сходится к циклу, но постепенно разбалтывается

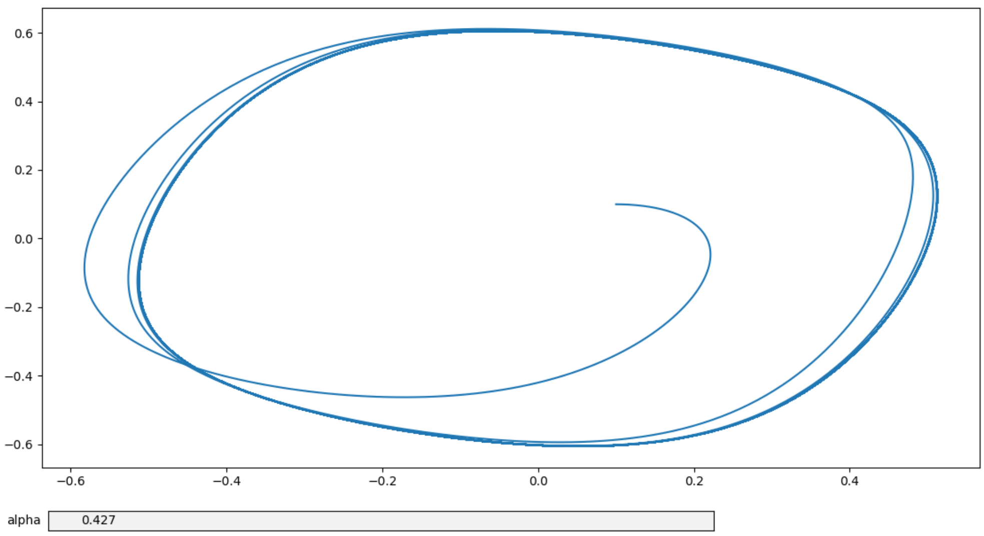

График 6.4 — ($\alpha = 0.427$) Второе решение перестало моделироваться. Первое все еще сходится к циклу

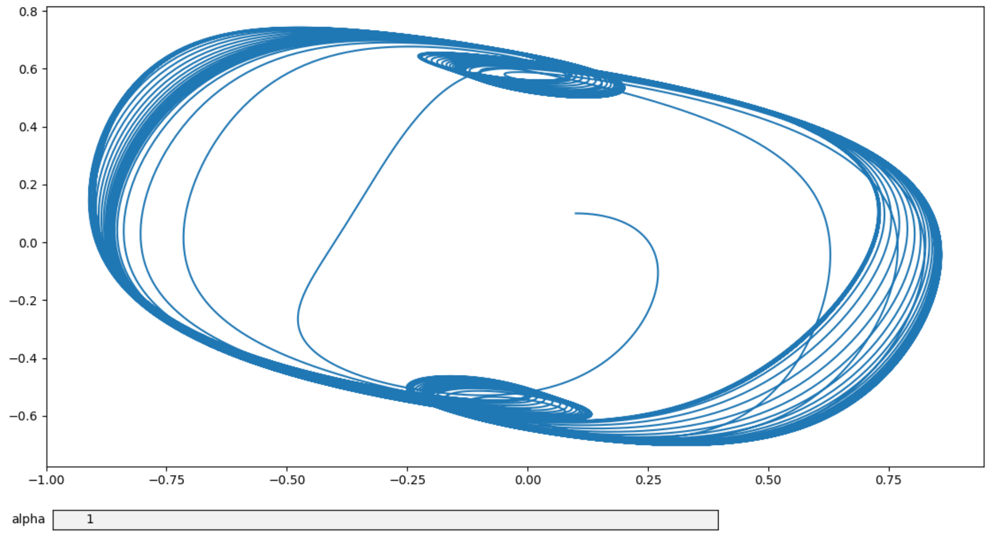

График 6.5 — ($\alpha = 1$) У первого решения появляются две петельки и траектория становится более хаотична

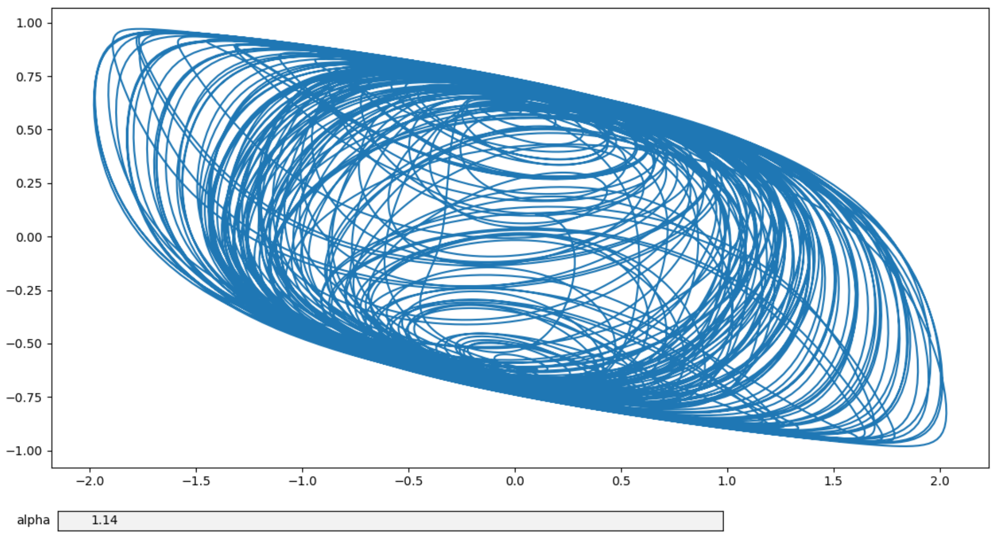

График 6.6 — ($\alpha = 1.24$) Первое решение хаотично изменятся в пределах цикла

6.2 Добавление запаздывания по y_2 в систему

Теперь пронаблюдаем влияние постоянной задержки, если она будет влиять на вторую координату нашей системы (Уравнение **??**). Значения задержки и начальные точки не меняем, для простоты и чистоты эксперимента.

$$\begin{cases} \dot{y_1} = y_2 \\ \dot{y_2} = -3y_2^3 + \nu y_2 - y_1 + \alpha * y_1(t-\tau) \end{cases} \tag{6.3}$$

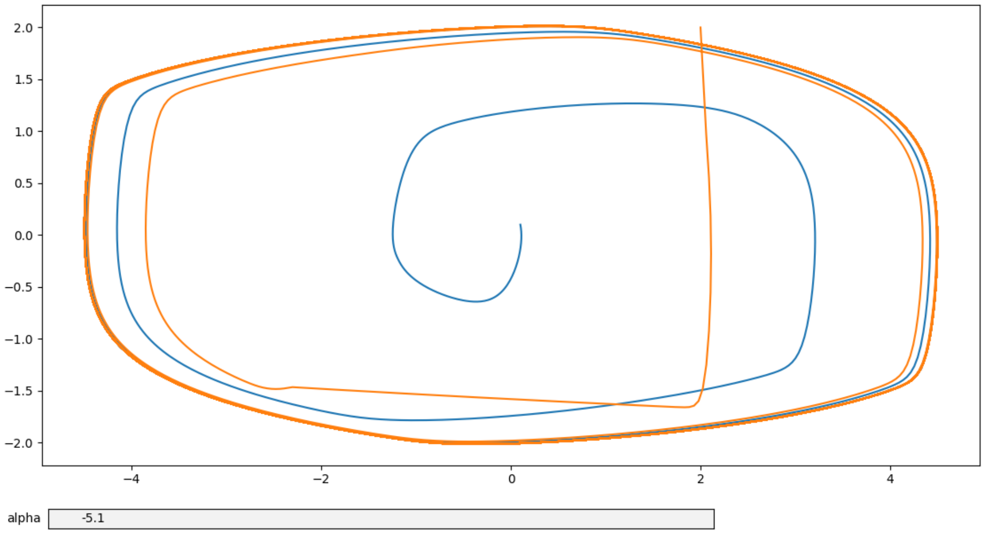

График 6.7 — ($\alpha = -5.1$) При отрицательном параметре изменяется форма цикла. Можно предполагать, что предельность данного цикла остается

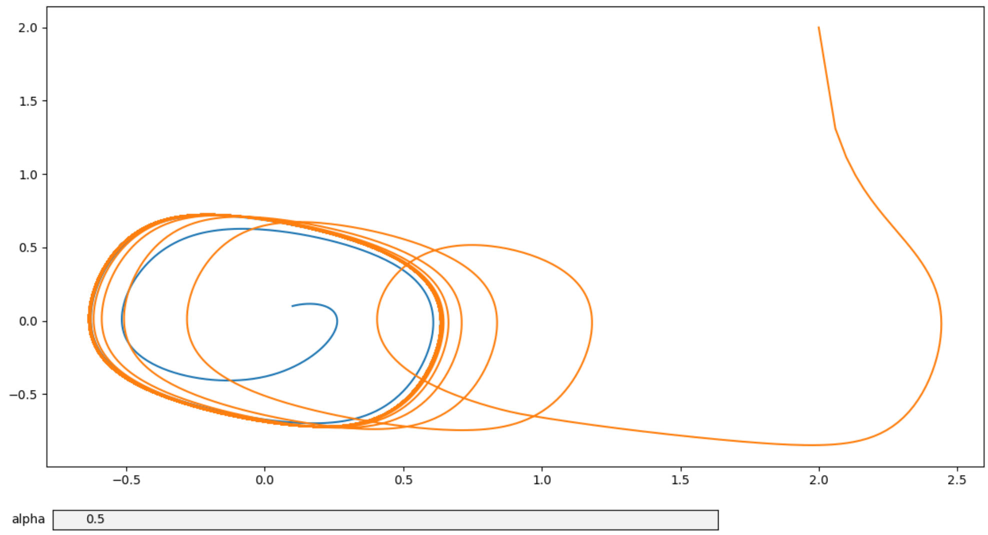

График 6.8 — ($\alpha = 0.5$) Цикл начинает раскручиваться в правую сторону

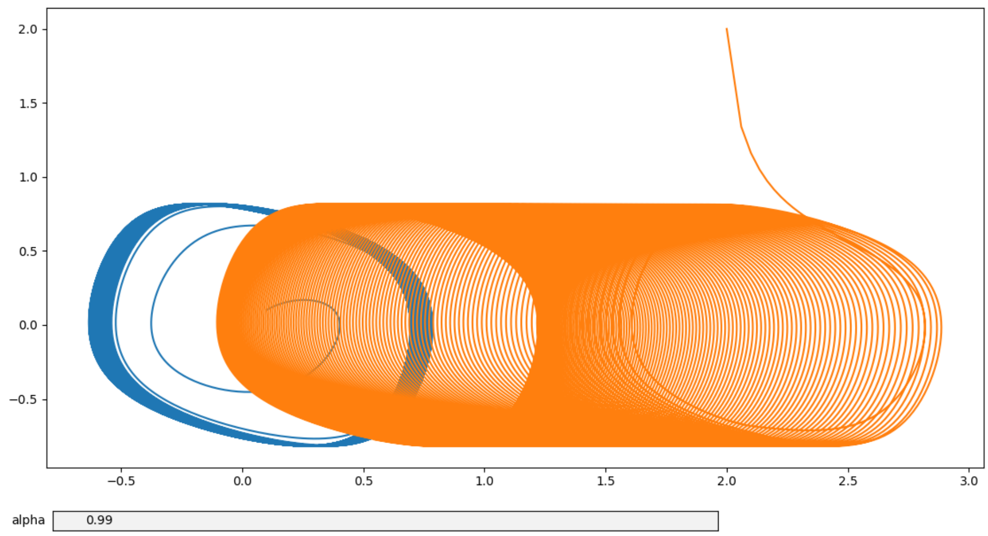

График 6.9 — ($\alpha = 0.99$) Наблюдается очень плотная спираль

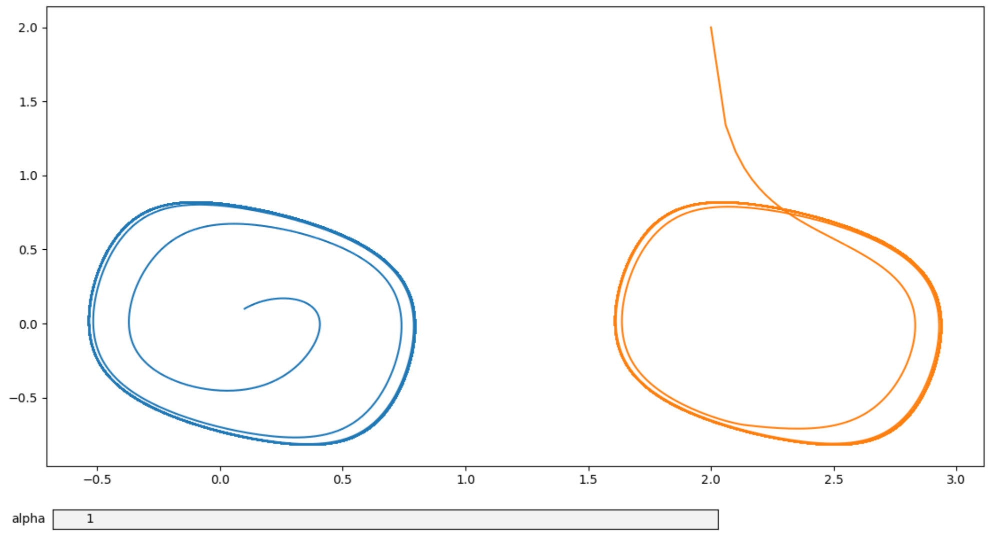

График 6.10 — ($\alpha = 1$) Потенциальная точка бифуркации. Два решения сошлись к двум разным циклам

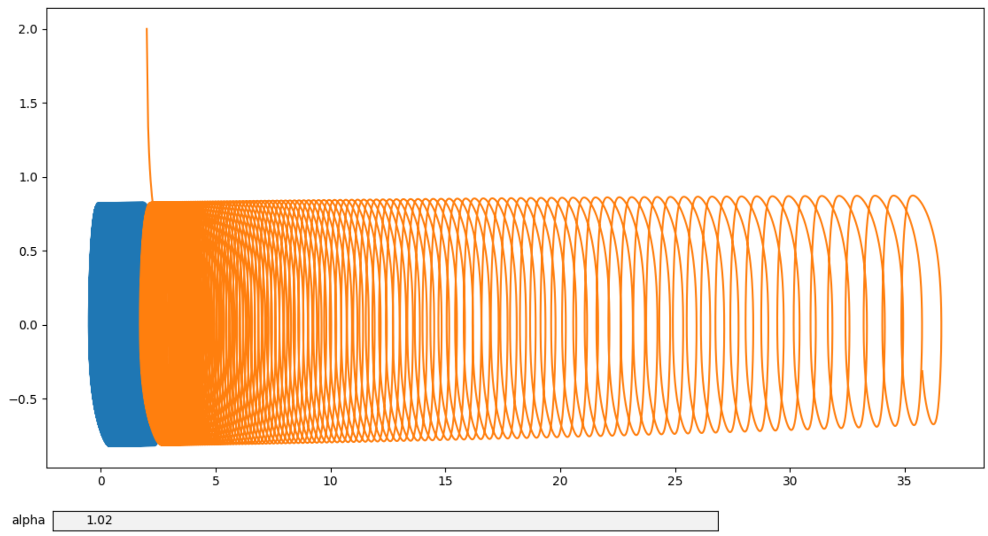

График 6.11 — ($\alpha = 1.02$) Даже небольшое увеличение α изминило систему

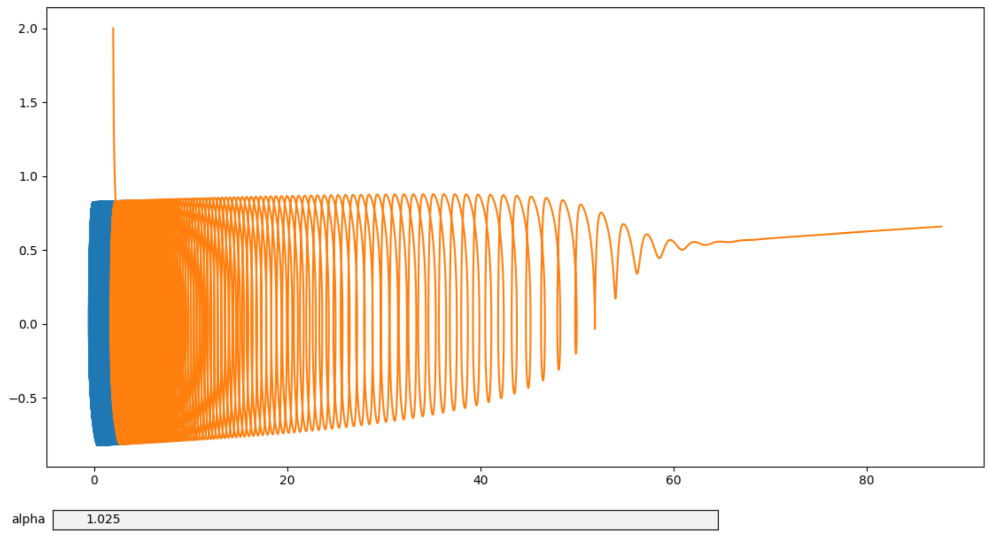

График 6.12 — ($\alpha = 1.025$) Решение начинает увеличиваться по y_1

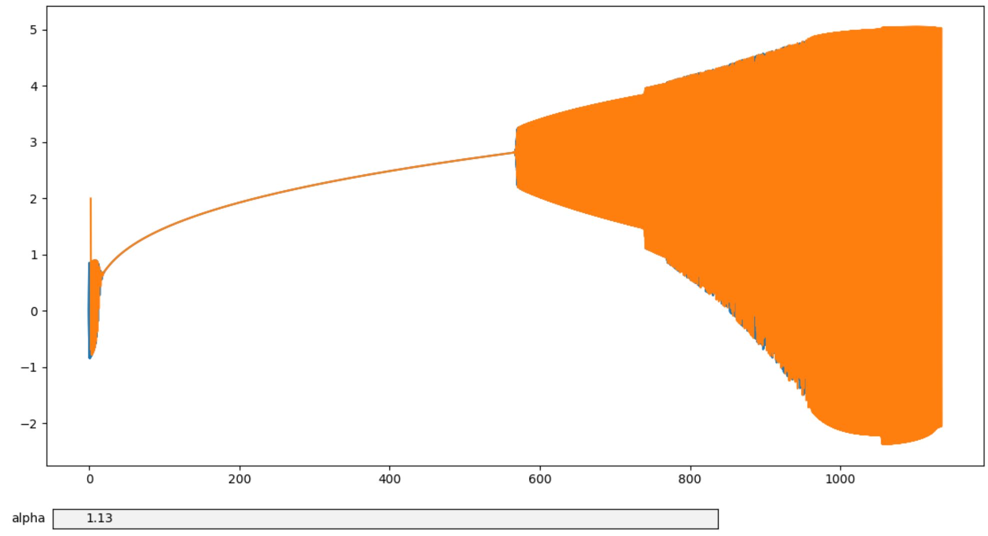

График 6.13 — ($\alpha = 1.13$) В интервале от 600 до 1000 наблюдаются резкие колебания

Дальнейшее увеличение α приводили к резкому увеличению значения координат, из-за чего вызывались ошибки в моделирующей системе.

6.3 Влияние постоянной задержки на систему. Заключение

Проведя эксперименты выше (6.1 и 6.2), не удавалось сильно изменить параметр α: сразу же начинали расходиться решения (или решение); проявлялось, своего рода, хаотичное поведение; исчезали циклы. Таким образом можно убедится в том, как сильно изменяет поведение системы даже самая простая, в плане понимания, задержка.

7 Влияние переменного запаздывания на систему

8 Влияние непрерывного запаздывания на систему

Заключение

Заключение будет, когда все закончится.

Список использованных источников

1. Numpy - библиотека для научиных вычислений на языке Python. — http://www.numpy.org/.

2. Matplotlib - библиотека для визуализации данных. — http://matplotlib.org/.

3. Функция streamplot. — http://matplotlib.org/examples/images_contours_and_fields/streamplot_demo_features.html.

Приложение А Исходный код программ

А.1 Поиск предельного цикла

```
import matplotlib.pyplot as plt
import numpy as np

# Параметр системы
nu = 1

# создание сетки 100x100 точек в области [-3;3]x[-3;3]
Y, X = np.mgrid[-3:3:100j, -3:3:100j]

# вычисление фазовых векторов на сетке
Y1 = Y
Y2 = -3 * Y ** 3 + nu * Y - X

# построение фазового портрета
fig, ax = plt.subplots()
plt.streamplot(X, Y, Y1, Y2)

# подпись осей на графике
ax.set_xlabel("y1")
ax.set_ylabel("y2")

# функция построение кривой методом Эйлера
def line(y1_0, y2_0):
    y1 = [y1_0]
    y2 = [y2_0]
    h = 0.01 # длина шага
    for i in range(2000): # 2000 - количество итераций
        y1.append(y1[i] + h*(y2[i]))
        y2.append(y2[i] +
                  h * (-3*y2[i] ** 3 + nu*y2[i] - y1[i]))
```

```
    # отображение кривой на графике
    ax.plot(y1, y2)

# построение двух кривых, начинающихся внутри и
# вне предпологаемого предельного цикла
line(0.1, 0.1)
line(2, 2)

# показать построенные графики
plt.show()
```

A.2 Исследование точек бифуркации системы

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button

ny0 = 0 # Первоначальное значения параметра

fig, ax = plt.subplots()
plt.subplots_adjust(bottom=0.15)

# подпись осей на графике
ax.set_xlabel("y1")
ax.set_ylabel("y2")

# функция построение кривой методом Эйлера
def line(y1_0, y2_0, ny):
    y1 = [y1_0]
    y2 = [y2_0]
    h = 0.003
    for i in range(50000):
        y1.append(y1[i] + h * (y2[i]))
        y2.append(y2[i] +
                  h * (-3*y2[i] ** 3 + ny * y2[i] - y1[i]))
    ax.plot(y1, y2)

def streamplot(ny):
    # увеличиваем параметры сетки в зависимости от модуля ny
    b = (4 + np.abs(ny)) # границы сетки
    c = (100 + np.abs(ny)) * 1j # число точек разбиения
    Y, X = np.mgrid[-b:b:c, -b:b:c]
    Y1 = Y
    Y2 = -3 * Y ** 3 + ny * Y - X
    ax.streamplot(X, Y, Y1, Y2)
```

```
# перерисовка графика в зависимости от параметра
# выводится фазовый портрет и две линии
def update_plot(ny):
    ax.cla()
    streamplot(ny)
    line(1. / (np.abs(ny) + 1), 1. / (np.abs(ny) + 1), ny)
    line(-ny, -6, ny)
    fig.canvas.draw_idle()

# Слайдер - чтобы менять параметр без перезапуска программы
axfreq = plt.axes([0.13, 0.05, 0.55, 0.03])
slider = Slider(axfreq, 'Ny', -10.0, 20.0, valinit=ny0)
slider.on_changed(update_plot)

step = 0.1 # шаг изменения параметра по нажатии клавиши
def _keyboard_handler(event):
    # выход при нажатии escape
    if event.key == 'escape':
        plt.close('all')
    # уменьшение параметра при нажатии стрелки "вниз"
    elif event.key == 'down':
        slider.set_val(slider.val - step)
    # увелечение параметра при нажатии стрелки "вверх"
    elif event.key == 'up':
        slider.set_val(slider.val + step)

fig.canvas.mpl_connect('key_press_event', _keyboard_handler)

update_plot(ny0)
plt.show()
```

A.3 Исследование параметров найденного предельного цикла

```
import numpy as np

ny = 1

y1_0 = 0.72424
y2_0 = 0

y1 = [y1_0]
y2 = [y2_0]
eps = 0.5 * 10 ** -4

# метод Эйлера с большей точностью
h = 0.0001
for i in range(100000):
    y1.append(y1[i] + h*(y2[i]))
    y2.append(y2[i] +
                h * (-3*y2[i] ** 3 + ny * y2[i] - y1[i]))
    # считаем цикл завершенным, когда покоординатно
    # приблизились к начальной точке ближе, чем на eps
    if (np.abs(y1_0 - y1[i+1]) < eps and
        np.abs(y2_0 - y2[i+1]) < eps):
        print("h={h}, i={i}, h*i={period}".format(
              h=h, i=i+1, period=h*(i+1))
        )
        break
```

A.4 Проверка устойчивости теоремой о мультипликаторах

```
import numpy as np


ny = 1     # Параметр системы
h = 0.0001 # Общий шаг метода Эйлера


def line(y1_0, y2_0):
    """
    Вычисление системы методом Эйлера от точки (y1_0, y2_0)
    за счет правильно подобраных в прошлой работе координат
    функция вычислит близкую к предельному циклу траекторию
    """
    y1 = [y1_0]
    y2 = [y2_0]
    eps = 0.5 * 10 ** -4
    for i in range(100000):
        y1.append(y1[i] + h * y2[i])
        y2.append(y2[i] +
                  h*(-3*y2[i] ** 3 + ny * y2[i] - y1[i]))
        if (np.abs(y1_0 - y1[i+1]) < eps and
            np.abs(y2_0 - y2[i+1]) < eps):
            return (y1, y2)

    raise Exception("Cycle not found")


def linear_form(y1_0, y2_0, cycle_y1, cycle_y2):
    """
    Решение линеаризованной системы вдоль цикла
    """
    y1 = [y1_0]
    y2 = [y2_0]
    for i in range(len(cycle_y1)):
```

```
        y1.append(y1[i] + h * (y2[i]))
        y2.append(y2[i] + h * (-y1[i] +
                       (-9*cycle_y2[i] ** 2 + ny) * y2[i]))
    return [y1[-1], y2[-1]]

# Начало программы
# вычисление поточечного описания предельного цикла
cycle_y1, cycle_y2 = line(0.72424, 0)

# решение линеаризированной системы
# с начальными условиями (1, 0) и (0, 1)
f1 = linear_form(1, 0, cycle_y1, cycle_y2)
f2 = linear_form(0, 1, cycle_y1, cycle_y2)

# Матрица монодромии
f = np.array([
    [f1[0], f2[0]],
    [f1[1], f2[1]],
])
print("F-matrix:")
print(f)

# Вычисление собственных чисел матрицы монодромии
p = np.linalg.eig(f)
print("Eigenvalues:")
print(p[0])
```

A.5 Проверка устойчивости цикла методом Пуанкаре

```
import numpy as np

ny = 1     # Параметр системы
h = 0.0001 # шаг метода Эйлера
           # и основание прямоугольников при интегрировании


def line(y1_0, y2_0):
    """
    Вычисление системы методом Эйлера от точки (y1_0, y2_0)
    за счет правильно подобраных в предыдущей работе координат
    функция вычислит близкую к предельному циклу траекторию
    """
    y1 = [y1_0]
    y2 = [y2_0]
    eps = 0.5 * 10 ** -4
    for i in range(100000):
        y1.append(y1[i] + h * (y2[i]))
        y2.append(y2[i] +
                  h*(-3*y2[i] ** 3 + ny * y2[i] - y1[i]))
        if (np.abs(y1_0 - y1[i + 1]) < eps and
            np.abs(y2_0 - y2[i + 1]) < eps):
            return (y1, y2)

    raise Exception("Cycle not found")


def integral_from_div(cycle_y1, cycle_y2):
    """
    Вычисление интеграла от дивергенции системы
    вдоль цикла
    """
```

```
    sum = 0
    for j in range(len(cycle_y1)):
        sum += -9 * cycle_y2[j] ** 2 + ny
    sum *= h
    return sum

# Основная программа
cycle_y1, cycle_y2 = line(0.72424, 0)
s = integral_from_div(cycle_y1, cycle_y2)
print("Integral of the divergence: {}".format(s))
```

A.6 Влияние постоянного запаздывания на систему

```
# Лаб. работа 6: влияние постоянного запаздывания на систему
import copy
import matplotlib.pyplot as plt
from matplotlib.widgets import TextBox


figure, ax = plt.subplots()
plt.subplots_adjust(bottom=0.15)
# подпись осей на графике
ax.set_xlabel("y1")
ax.set_ylabel("y2")


ny = 1
alpha0 = 0.01

def line(y1_0, y2_0, alpha):
    # решение задачи методом Эйлера
    y1 = copy.deepcopy(y1_0)
    y2 = copy.deepcopy(y2_0)
    h = 0.03
    for i in range(len(y1_0)-1, 20000):
        # Запаздывание влияет на y1
        y1.append(y1[i] + h * (y2[i] +
                            alpha * y1[i - len(y1_0) + 1]))
        y2.append(y2[i] +
                  h * (-3*y2[i] ** 3 + ny * y2[i] - y1[i]))

        # Запаздывание влияет на y2
        # y1.append(y1[i] + h * (y2[i]))
        # y2.append(y2[i] +
        #           h * (-3*y2[i] ** 3 + ny * y2[i] - y1[i] +
```

```
        #                alpha * y1[i - len(y1_0) + 1]))
    ax.plot(y1, y2)


def update_plot(strAlpha):
    # перерисовка графика в зависимости от параметра
    alpha = float(strAlpha)
    ax.cla()
    line(
        [0.1 for a in range(100)],
        [0.1 for a in range(100)],
        alpha)
    line(
        [2 for a in range(100)],
        [2 for a in range(100)],
        alpha)
    figure.canvas.draw_idle()


axfreq = plt.axes([0.13, 0.05, 0.55, 0.03])
textBox = TextBox(axfreq, 'alpha', initial=str(alpha0))
textBox.on_submit(update_plot)

def _keyboard_handler(event):
    # выход при нажатии escape
    if event.key == 'escape':
        plt.close('all')


figure.canvas.mpl_connect('key_press_event', _keyboard_handler)

update_plot(alpha0)
plt.show()
```

A.7 Влияние переменного запаздывания на систему

A.8 Влияние непрерывного запаздывания на систему