

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
**«Национальный исследовательский**  
**Нижегородский государственный университет**  
**им. Н.И. Лобачевского»**  
**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра: Теории управления и динамики систем**  
**(ТУДС)**

Направление подготовки: «**Фундаментальная информатика и**  
**информационные технологии»**

Профиль подготовки: «**Инженерия программного обеспечения»**

## **ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА** **БАКАЛАВРА**

Тема:  
**«Нахождение неустойчивых периодических орбит**  
**хаотического аттрактора с помощью покрытия короткими**  
**отрезками траекторий»**

**Выполнил:**  
студент группы 3821Б1ФИ2  
Канаков Роман Андреевич

---

Подпись

**Научный руководитель:**  
Гринес Евгений Александрович

---

Подпись

Нижний Новгород  
2025

# Содержание

<b>1 Введение</b>	<b>4</b>
<b>2 Базовые сведения</b>	<b>5</b>
2.1 Динамические системы . . . . .	5
2.2 Хаос . . . . .	6
<b>3 Алгоритм</b>	<b>8</b>
3.1 Первые версии . . . . .	8
3.1.1 Описание . . . . .	8
3.1.2 Причины отбраковки . . . . .	13
3.2 Итоговый вариант . . . . .	13
3.2.1 Сечение Пуанкаре . . . . .	13
3.2.2 Рекуррентные матрицы и их место в алгоритме . . . . .	14
3.2.3 Оптимизированный метод стрельбы . . . . .	14
3.2.4 Фильтрация орбит с помощью сечения Пуанкаре . . . . .	16
<b>4 Анализ результатов и вывод</b>	<b>17</b>
<b>5 Руководство пользователя</b>	<b>20</b>
5.1 Введение . . . . .	20
5.2 Первая страница . . . . .	20
5.3 Вторая страница . . . . .	22
5.4 Третья страница . . . . .	23
5.5 Чётвертая страница . . . . .	28
<b>6 Руководство программиста</b>	<b>29</b>
6.1 Введение . . . . .	29
6.2 Описание структуры программы . . . . .	30
6.3 Анализ логирования . . . . .	31
<b>7 Приложения</b>	<b>34</b>
7.1 Приложение 1 . . . . .	34
7.2 Приложение 2 . . . . .	34
7.3 Приложение 3 . . . . .	35
7.4 Приложение 4 . . . . .	37
7.4.1 Исходный код файла <code>main.jl</code> . . . . .	37
7.4.2 Исходный код файла <code>MainPage.jl</code> . . . . .	37

7.4.3	Исходный код файла <code>SecondPage.jl</code>	40
7.4.4	Исходный код файла <code>ThirdPage.jl</code>	43
7.4.5	Исходный код файла <code>FourthPage.jl</code>	44
7.4.6	Исходный код файла <code>poincare.jl</code>	55
7.4.7	Исходный код файла <code>LM.jl</code>	65
7.4.8	Исходный код файла <code>util.jl</code>	66

# 1 Введение

Одни и те же динамические системы могут быть представлены во многих обличьях. Вопрос инвариантности характеристик является первостепенным в теории динамических систем. Аттракторы – это важная часть исследования нелинейной динамики. Для простых аттракторов (устойчивое состояние равновесия или устойчивый предельный цикл) существует развитая теория, корни которой уходят в конец XIX века. Изучение инвариантов хаотических аттракторов до сих пор является открытой областью исследований. Существует несколько разных подходов. Хорошо известно, что внутри хаотического аттрактора находится очень много неустойчивых предельных циклов, расположение которых определяет структуру аттрактора. Важной задачей в изучении свойств хаотического аттрактора является умение находить периодические орбиты внутри него.

Периодическая орбита – это траектория, которая возвращается в свою начальную точку через конечное время  $T$ . Математически это можно выразить следующим образом:

$$\bar{\mathbf{x}} = \varphi(\bar{\mathbf{x}}, 0) = \varphi(\bar{\mathbf{x}}, T),$$

где  $\varphi$  – оператор эволюции системы, а  $\bar{\mathbf{x}}$  – любая точка траектории  $\mathbf{x}$ .

В нелинейных динамических системах периодические орбиты являются топологическим инвариантом [1]. Это означает, что они сохраняют свои свойства при любом выборе координат или представлении динамики. Инвариантность делает периодические орбиты особенно полезными для анализа хаотических систем.

Число периодических орбит, их распределение и их свойства раскрывают структуру хаотических аттракторов. Если мы хотим узнать, являются ли два экспериментальных аттрактора одинаковыми (с точностью до возможного плавного изменения координат), или если мы хотим утверждать, что теоретическая модель точно воспроизводит экспериментальный аттрактор, эта информация имеет решающее значение для осмыслинного сравнения [2].

Периодические орбиты имеют приложения не только в определении структуры хаотического аттрактора, они также могут помочь в численном вычислении статистических свойств. В статье [3] приведены множество формул, позволяющие находить различные показатели системы, например старший показатель Ляпунова.

Многие реалистичные системы различной природы, описываемые дифференциальными уравнениями, обнаруживают хаотическое поведение при численном моделировании динамики на компьютере или в эксперименте.

Можно ли утверждать, что наблюдаемые в численных расчётах и в эксперименте сложные непериодические колебательные режимы это динамический хаос, феномен той же природы, как и в модельных системах? Это серьезный и трудный вопрос, поскольку реалистичные системы, как правило, не допускают простого и далеко идущего теоретического анализа, как элементарные примеры.

Вполне вероятно, что не существует идеального алгоритма для нахождения неустойчивых предельных циклов в экспериментально полученных данных. Тем не менее, для теоретических моделей периодические орбиты можно найти до определённой точности. Исследованию этой возможности и посвящена данная работа.

## 2 Базовые сведения

### 2.1 Динамические системы

Термин динамическая система может описывать большой спектр процессов и быть применен практически в любом разделе науки. Говоря упрощённо, это математическая модель, описывающая эволюцию набора переменных во времени согласно определённым законам. Формально динамическая система задаётся

1. Фазовым пространством.
2. Временем.
3. Законом эволюции.

Существует множество классификаций динамических систем, например:

1. Дискретные и непрерывные.
  - Дискретные динамические системы описываются итерационными отображениями:
$$\mathbf{x}_{n+1} = f(\mathbf{x}_n),$$
где  $f$  – динамическое правило или уравнение движения, определяющее временную эволюцию системы. Время здесь является дискретной величиной.
  - Непрерывные системы задаются дифференциальными уравнениями
$$\dot{\mathbf{x}} \equiv \frac{d\mathbf{x}}{dt} = f(\mathbf{x})$$
Время в таких системах непрерывно, а  $f$  берет текущее состояние и возвращает скорость его изменения. Можно сказать, что  $f$  имеет семантику векторного поля.

2. Диссипативные и консервативные.

- Система называется диссипативной, если для почти всех начальных значений  $\mathbf{x}$  объем  $\mathbf{v}$  инфинитезимально малого шара начальных условий, следующего за траекторией, начинающейся в  $\mathbf{x}$ , будет стремиться к нулю по мере  $t \rightarrow \infty$  [4]. Другими словами, диссипативные системы характеризуются сжатием фазового объема.
- В физике свойство консервативности понимается как сохранение полной механической энергии, а в присутствии трения (то есть для неконсервативных систем) механическая энергия рассеивается (или диссирирует) и превращается в тепло. В нашем контексте это означает, что такие системы всегда сохраняют свой фазовый объем [5].

3. Автономные и неавтономные.

В автономных системах правые части уравнений явно не зависят от времени.

## 2.2 Хаос

Хаос в общем случае довольно сложно определить, существует несколько взаимодополняющих определений:

1. Хаос возможен благодаря экспоненциальной дивергенции сходимости близких траекторий, то есть при положительности старшего показателя Ляпунова. Это концептуально является самым базовым индикатором детерминистического хаоса [6]. Но это не единственное условие, поскольку только лишь положительность старшего показателя Ляпунова не ведёт к хаосу. Она говорит о том, что две траектории будут бесконечно расходиться. Нам же нужно, чтобы они так или иначе встретились вновь. Другими словами, хаос возникает не только из-за расхождения траекторий, но и из-за того, что они сходятся и пересекаются в ограниченной области фазового пространства [7].
2. Хаотические аттракторы всюду плотны<sup>1</sup> периодическими орбитами [8]. Это означает, что любая точка хаотического аттрактора лежит рядом с неустойчивым предельным циклом.
3. Хаотический режим определяется [9]
  - (a) Детерминированностью
  - (b) Ограниченностю
  - (c) Непериодичностью
  - (d) Чувствительностью к начальным условиям

В двух разных определениях фигурирует слово ‘детерминированный’. Казалось бы, какое оно может иметь отношение к хаосу? На самом деле, прямое. Дело в том, что детерминированность никак не противоречит хаосу, так как в любой хаотической системе (без возмущений) можно однозначно определить эволюцию произвольного начального значения на заданный промежуток времени. Хаос, с этой точки зрения, говорит лишь о том, что малейшее изменение этого начального условия приведёт к значительным изменениям состояний системы и значительным расхождениям траекторий, при этом всё равно стремящихся к тому же самому аттрактору. Это возможно благодаря неустойчивости, присущей фазовым траекториям хаотической системы.

Довольно просто заметить чувствительных системы к малейшему изменению начальных условий, например, на модели Лоренца — это одна из самых известных хаотических систем, обладающая очень удобными для изучения свойствами. Она ещё не раз будет фигурировать в этой работе. Система Лоренца задается системой уравнений

$$\begin{cases} \dot{x} = \sigma \cdot (y - x) \\ \dot{y} = -x \cdot z + \rho \cdot x - y \\ \dot{z} = x \cdot y - \beta \cdot z \end{cases}$$

Рассмотрим следующий рисунок, где показан набор большого числа наложенных друг на друга временных зависимостей динамической переменной **z**. Исходный код

<sup>1</sup>Напомним, что множество  $A$  называется всюду плотным во множестве  $B$ , если  $\forall b \in B, \epsilon > 0 \exists a \in A$ , что  $a \in U_\epsilon(b)$

программы на языке программирования Julia, которая генерирует эту картинку, доступен в Приложении 1.

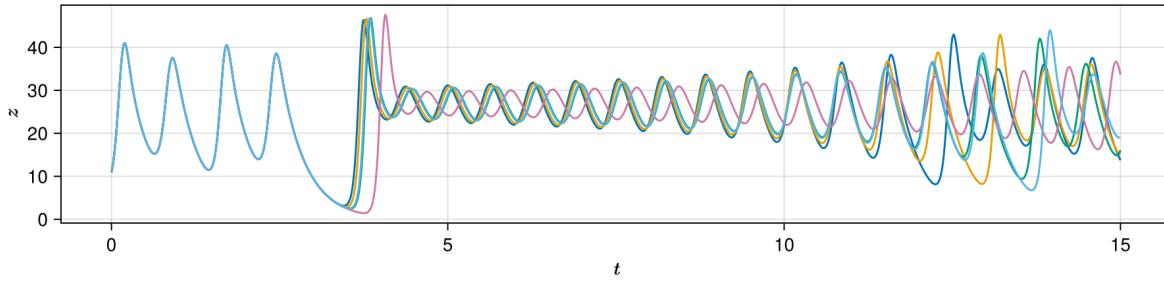


Рис. 1: Изменение поведения близких начальных значений в системе Лоренца.

Именно присутствие неустойчивости создает возможность соединить казалось бы несоединимое — динамическую природу системы, то есть предсказуемость, и хаос, то есть непредсказуемость.

Основная цель этой работы — разработать интерактивное графическое приложение, позволяющее находить и визуализировать периодические орбиты в странных хаотических аттракторах. Притягивающее множество динамической системы — это закрытое подмножество фазового пространства системы, для которого у «многих» начальных точек эволюция будет устремлена в направлении этого множества. Аттрактор — это такое притягивающее множество точек в фазовом пространстве диссипативной (и только!) системы, которое не может быть разделено на меньшие части [10]. Аттрактор является инвариантным множеством. Множество точек фазового пространства называют инвариантным в том случае, если фазовая траектория, стартующая из любой его точки, целиком принадлежит этому множеству. Любой аттрактор — инвариантное множество, но инвариантное множество не всегда аттрактор. Простейшими примерами аттрактора являются устойчивое состояние равновесия и устойчивый предельный цикл.

В этой работе речь идёт исключительно о диссипативных автономных трёхмерных хаотических системах с непрерывным временем. Выбор именно таких систем не случаен, каждое из перечисленных определений осмысленно:

### 1. Диссипативность.

Именно это свойство даёт возможность для существования странных хаотических аттракторов, так как аттрактор является притягивающим множеством нулевой меры в фазовом пространстве, на котором концентрируется с течением времени облако изображающих точек.

### 2. Автономность.

Неавтономные системы намного сложнее для изучения и исследований, нас интересует более “статический” характер динамики, то есть тот, который может повторяться для разных начальных значений, без привязки ко времени их возникновения.

### 3. Трёхмерность.

Поскольку в данной работе большое внимание уделено визуализации, выбор систем с размерностью больше чем 3 повлечет за собой сложности в интерпретации результатов. В непрерывных автономных диссипативных системах размерности меньше 3 хаотический режим невозможен [5].

## 3 Алгоритм

### 3.1 Первые версии

#### 3.1.1 Описание

Первым способом поиска периодических орбит была рекуррентная матрица [11]. Предположим, у нас есть траектория  $\{\mathbf{x}_i\}_{i=1}^N$  системы в фазовом пространстве, то есть элементы или “точки”  $N$  последовательных состояний системы, представленные в виде вектора.

Элементом рекуррентной матрицы является

$$R_{i,j} = \begin{cases} 1, & \text{metric}(\mathbf{x}_i, \mathbf{x}_j) \leq \epsilon \\ 0, & \text{metric}(\mathbf{x}_i, \mathbf{x}_j) > \epsilon \end{cases}$$

Где  $\text{metric}(\mathbf{x}_i, \mathbf{x}_j)$  – расстояние между точками траектории в состояниях  $i$  и  $j$  соответственно, а  $\epsilon$  – размер рассматриваемой окрестности для каждой из точек. В качестве расстояния в основном используют нормы  $l_1$ ,  $l_2$  и  $l_\infty$ :

$$\begin{aligned} l_1 &= \sum_k |\mathbf{x}_{ik} - \mathbf{x}_{jk}| \\ l_2 &= \sqrt{\sum_k (\mathbf{x}_{ik} - \mathbf{x}_{jk})^2} \\ l_p &= \sqrt{\sum_k (\mathbf{x}_{ik} - \mathbf{x}_{jk})^p} \\ l_\infty &= \max |\mathbf{x}_{ik} - \mathbf{x}_{jk}| \end{aligned}$$

Исходный код для следующего рисунка доступен в Приложении 2.

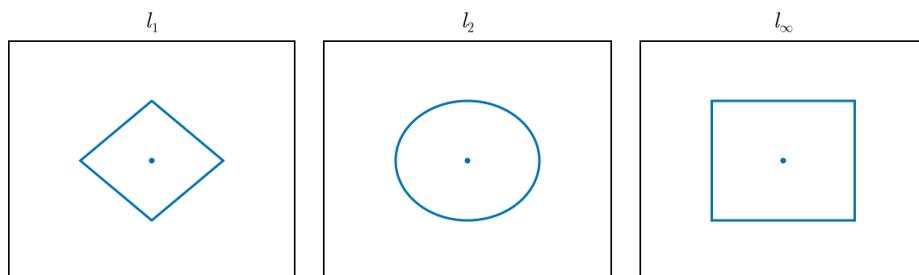


Рис. 2: Графическая интерпретация  $l$  норм.

Рекуррентная матрица показывает в какие моменты времени траектория была близка к самой себе или, другими словами, когда возникают аналогичные состояния системы. Очевидно, что главная диагональ этой матрицы всегда состоит из единиц, ведь расстояние между точками системы в один и тот же момент времени, независимо от выбранной метрики, всегда будет равно нулю.

Важнейшим и единственным параметром рекуррентной матрицы является  $\epsilon$ . Слишком маленькое значение  $\epsilon$  приведёт к тому, что рекуррентных точек практически не будет и мы не сможем сделать полноценный анализ системы. С другой стороны, если выбрать  $\epsilon$  слишком большим, то почти каждая точка будет считаться соседней с любой другой. Это приведет к тому, что большинство найденных точек не будет иметь перспектив для анализа.

На следующей картинке изображена графическая визуализация попадания пунктирной траектории в  $\epsilon$ -окрестность сплошной, исходный код программы доступен в Приложении 3.

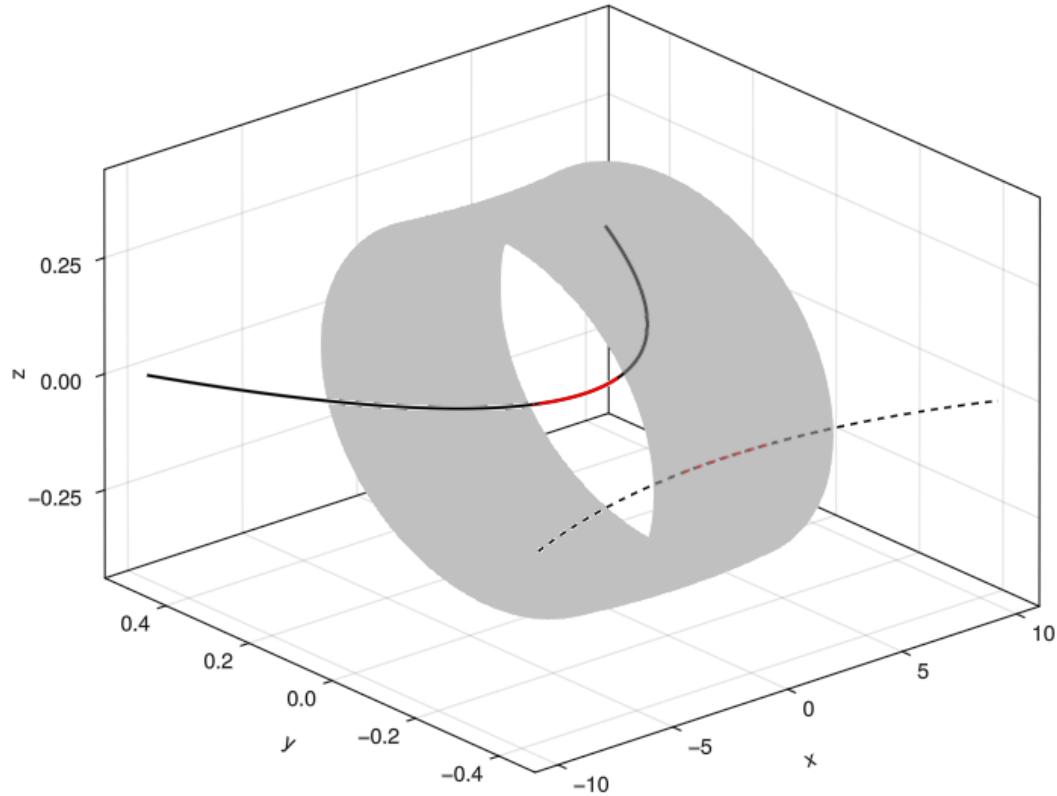


Рис. 3: Графическая интерпретация попадания точек пунктирной кривой в  $\epsilon$ -окрестность сплошной.

К сожалению, не существует оптимального алгоритма выбора  $\epsilon$ , поскольку это зависит от масштабов, в которых находится аттрактор и точности, которую требует пользователь.

Чтобы визуализировать рекуррентную матрицу используют различные цвета для её двоичных записей, например, черной точкой обозначим координату  $(i, j)$ , если  $R_{i,j} = 1$  и белой точкой, если  $R_{i,j} = 0$ .

Перед подробным описанием алгоритма, стоит упомянуть, что имеет смысл искать орбиты с небольшим временем периода, поскольку короткие циклы дают хорошие приближения странного множества и ошибка из-за пренебрежения длинными циклами может быть ограничена. Пользователь имеет возможность задавать минимальное и максимальное время первого возврата. Это позволяет находить периодические орбиты с периодом из определённого диапазона.

Рассмотрим последовательно все действия, которые проводились с рекуррентной матрицей для извлечения из неё данных, позволяющих найти и описать неустойчивые предельные циклы.

Важное замечание: на картинках используется искусственно созданная матрица, позволяющая детально раскрыть каждое из описанных действий.

- Поскольку рекуррентная матрица является симметричной, имеет смысл рассматривать только одну её половину, например, верхнюю, а нижнюю не хранить для экономии памяти.

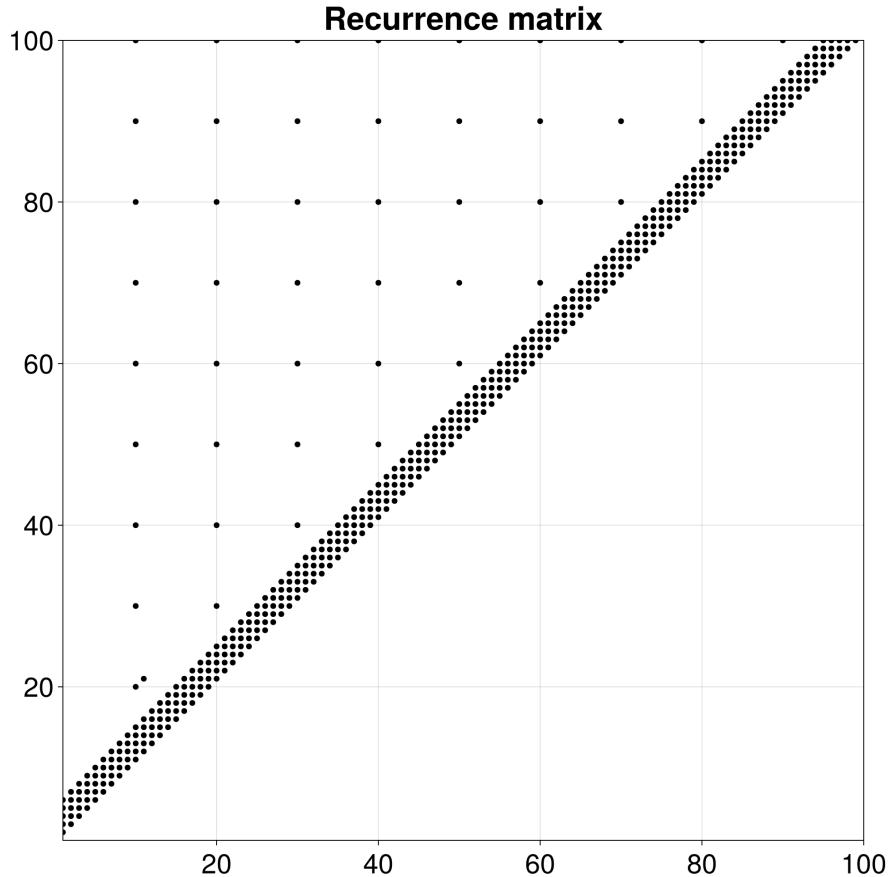


Рис. 4: Рекуррентная матрица без нижней части.

2. Убираем все точки, которые находятся выше максимального времени периода или ниже минимального. Это также экономит память и упрощает анализ.

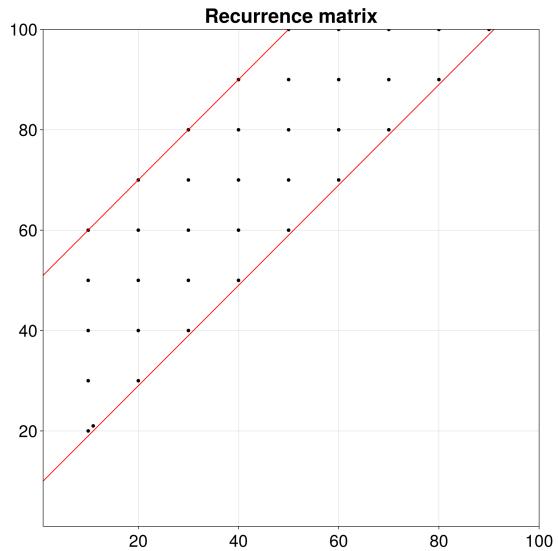


Рис. 5: Рекуррентная матрица с точками, находящимися внутри рассматриваемых пределов.

3. Наращиваем компоненту связности для каждой точки в разные стороны.

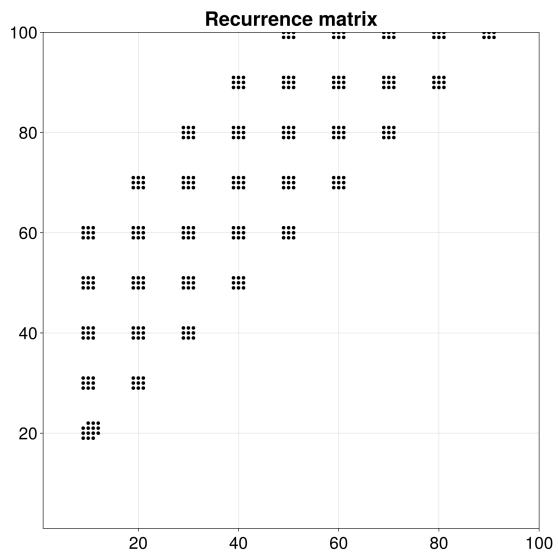


Рис. 6: Рекуррентная матрица с точками после операции дилатации.

4. Может случиться ситуация, когда несколько точек находятся друг над другом. Это может означать две вещи: либо несколько последовательных моментов времени попали в  $\epsilon$ -окрестность точки из-за медленного движения, либо это несколько разных возвратов. В любом случае нас интересует только самый первый возврат в  $\epsilon$ -окрестность. Поэтому мы будем избавляться от таких “дубликатов”.

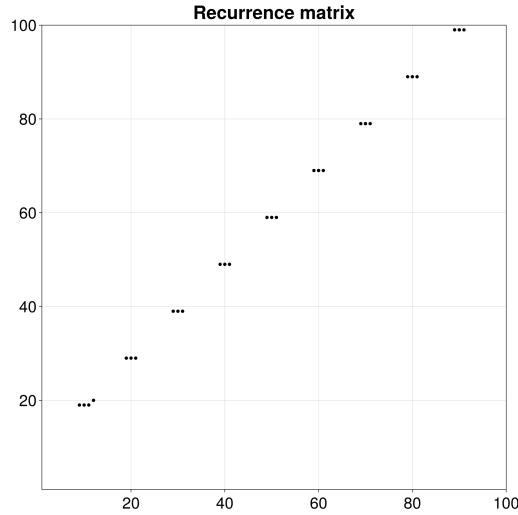


Рис. 7: Рекуррентная матрица без дубликатов.

5. Довольно частая ситуация — возникновение прямых или как мы их называем — “компонент связностей”. Это соответствует ситуации, когда траектория после первого возвращения, продолжает двигаться в  $\epsilon$ -окрестности следующих точек. Нас снова интересует только первый возврат в окрестность, поэтому избавляемся от всей компоненты, оставляя только самую левую нижнюю точку.

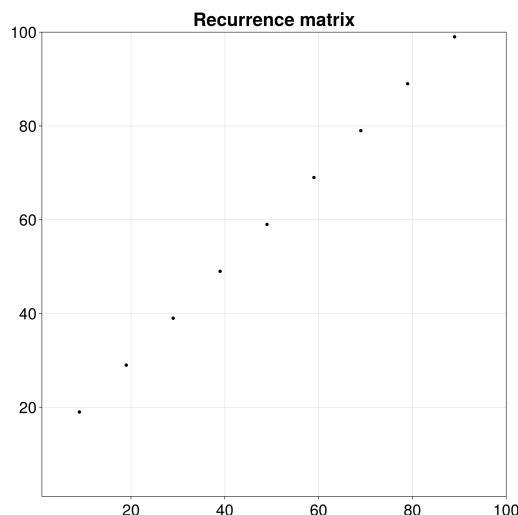


Рис. 8: Итоговая рекуррентная матрица.

В итоге мы получаем набор точек, каждая из которых представляет собой 2 момента времени — начало периодической орбиты и её первое возвращение. Из этих времён можно реконструировать орбиты, а из них сам аттрактор.

### 3.1.2 Причины отбраковки

Полученные результаты не были удовлетворительными.

Одной из главных проблем является дублирование большинством найденных периодических орбит друг друга. Несмотря на отличия в значениях точек, их динамика неразличима, а проверять каждую точку с каждой другой — очень ресурсозатратно. Это особенно очевидно, после реконструкции аттрактора из всех найденных орбит.

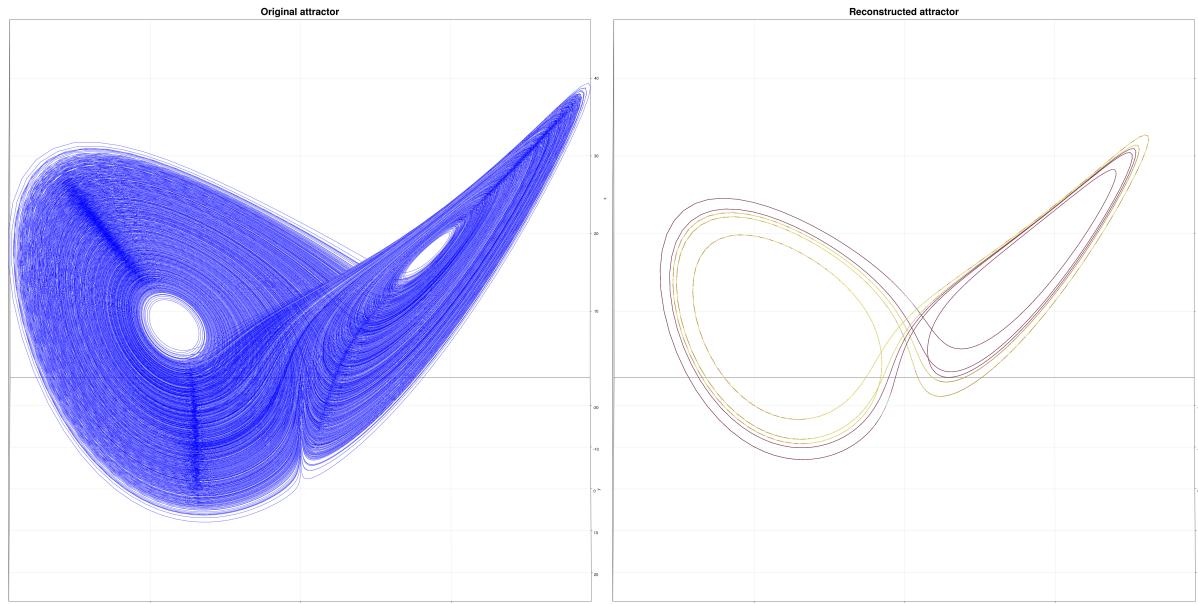


Рис. 9: Исходный и восстановленный аттрактор из найденных периодических орбит.

Для осознания всей проблемы стоит отметить, что число **различных** орбит на картинке справа — 144, каждая из которых имеет порядка 900 точек.

Ещё одной причиной, из-за которой данный алгоритм был отбракован, является то, что настоящие периодические орбиты странного аттрактора не обязательно будут лежать в длинной траектории, которая интегрируется для одного начального значения. То что мы находим является лишь приближением настоящего предельного цикла. Чтобы находить периодические орбиты более точно нужно использовать численные методы для найденного начального приближения.

## 3.2 Итоговый вариант

### 3.2.1 Сечение Пуанкаре

Одним из главных отличий итогового алгоритма от предыдущего является не только добавление интерактивного графического интерфейса, но и изменения в са-

мом поиске периодических орбит. Вместо дорогого построения рекуррентной матрицы, пользователю предлагается выбрать сечение Пуанкаре, точки которого будут использованы для нахождения неустойчивых предельных циклов. Введём определение сечения Пуанкаре более подробно.

Расположим в фазовом пространстве некоторой трёхмерной автономной динамической системы двумерную площадку  $S$  и зададим на ней некоторую систему координат  $X, Y$ . Выбор секущей поверхности в высокой степени произволен, но она должна размещаться так, чтобы интересующие нас фазовые траектории многократно её пересекали и касание было исключено [5]. Возьмём какую-нибудь точку  $(x, y)$  на секущей поверхности, выпустим из неё фазовую траекторию и проследим за этой траекторией, пока не произойдет следующее её пересечение с нашей площадкой  $S$  в некоторой точке  $(\hat{x}, \hat{y})$  с проходом в том же направлении. Следовательно, возникает некоторое отображение секущей поверхности в себя:

$$\hat{x} = f_1(x, y), \quad \hat{y} = f_2(x, y)$$

Это и есть отображение последования, или отображение Пуанкаре. Оно позволяет “преобразовать”  $D$ -мерную непрерывную динамическую систему в  $(D - 1)$ -мерную дискретную, описываемую результирующим отображением Пуанкаре.

Отображение Пуанкаре является полным представлением непрерывной динамики благодаря уникальности и гладкости решений ОДУ.

Одной из причин использования сечения Пуанкаре в алгоритме заключаются в том, что оно позволит в дальнейшем проводить фильтрацию найденных орбит и извлекать из неё дубликаты.

### 3.2.2 Рекуррентные матрицы и их место в алгоритме

Для каждой найденной точки система интегрируется на время, равное максимальному времени периода, которое задаёт пользователь. Для каждого полученного временного ряда строится первый столбец рекуррентной матрицы, который находит первый возврат к точке, лежащей на плоскости, если такой существует в указанном временном интервале. Другими словами, из первого столбца мы получаем приближительный период периодической орбиты.

У нас есть все необходимые данные для уточнения орбиты — начальное приближение и примерный период. Нужно использовать метод, уточняющий наше приближение.

### 3.2.3 Оптимизированный метод стрельбы

В статье [12] предлагается численный метод для нахождения периодических орбит. Опишем его основные идеи. Он применим к любой динамической системе, которую можно записать в виде

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \alpha, t),$$

где  $\mathbf{f}$  — это вектор-функция  $(f_1, f_2, \dots, f_N)^T$ . Каждая  $f_i$  является функцией соответствующей динамической переменной  $x_i$  вектора  $\mathbf{x} = (x_1, x_2, \dots, x_N)$ , а также параметров системы  $\alpha$  и времени  $t$ .

Если  $T$  – предположительный период орбиты, то можно ввести замену координат  $t = T \cdot \tau$ . Подстановка породит эквивалентную систему

$$\frac{d\mathbf{x}}{d\tau} = T \cdot \mathbf{f}(\mathbf{x}, \alpha, T \cdot \tau)$$

Это уравнение имеет преимущество перед исходным, поскольку граничные условия для периодического решения теперь можно выразить как  $\mathbf{x}(0) = \mathbf{x}(T)$ .

Начиная с условия  $\mathbf{x}(\tau = 0)$ , можно численно интегрировать уравнение, варьируя  $\tau$  от 0 до 1. Эта интеграция позволяет определить невязку, которая может быть записана следующим образом

$$\mathbf{R} = T \int_0^1 \mathbf{f}(\mathbf{x}, \alpha, T\tau) d\tau$$

После чего мы разделяем интервал  $[0, 1]$  на  $p$  равных частей и аппроксимируем интеграл суммой

$$\mathbf{R} \approx T \sum_{i=0}^{p-1} \mathbf{f}(\mathbf{x}(i\Delta\tau), \alpha, T(i\Delta\tau)) \cdot \Delta\tau,$$

где  $\Delta\tau$  – шаг интегрирования.

Поскольку функция  $\mathbf{f}$  описывает скорость изменения вектора  $\mathbf{x}$ , то интегрирование системы на шаг  $\Delta\tau$  дает следующую аппроксимацию изменения вектора  $\mathbf{x}$

$$\mathbf{x}((k+1)\Delta\tau) \approx \mathbf{x}(k\Delta\tau) + \mathbf{f}(\mathbf{x}(k\Delta\tau), \alpha, T(k\Delta\tau)) \cdot \Delta\tau$$

Следовательно разность  $\mathbf{x}(1+k\Delta\tau) - \mathbf{x}(k\Delta\tau)$  соответствует изменению вектора  $\mathbf{x}$  за один период  $T$ , сдвинутый на  $k\Delta\tau$ .

Невязку можно записать явно следующим образом

$$\mathbf{R} = (\mathbf{x}(1) - \mathbf{x}(0), \mathbf{x}(1 + \Delta\tau) - \mathbf{x}(\Delta\tau), \dots, \mathbf{x}(1 + (p-1)\Delta\tau) - \mathbf{x}((p-1)\Delta\tau))$$

А дальше решается задача нахождения суммы наименьших квадратов

$$C(\alpha) = \frac{1}{2} \sum_{k=1}^m \left( \frac{\mathbf{R}}{w_k} \right)^2$$

с помощью алгоритма Левенберга-Марквардта. В этой работе используется готовая реализация этого метода из библиотеки NonlinearSolve. Выбор именно этого алгоритма не случаен, поскольку он обладает рядом специфичных свойств, которые делают его лучше по сравнению с другими методами оптимизации:

1. Гибридный подход: метод Левенберга-Марквардта часто называют средним между методами Гаусса-Ньютона и градиентного спуска. Это объясняется тем, что он обладает быстрой сходимостью вблизи минимума – как метод Ньютона, и устойчивостью вдали от минимума – как градиентный спуск.
2. Позволяет работать с вырожденными матрицами и не требует положительной определённости матрицы Гессе как метод Гаусса-Ньютона.
3. Быстро сходится, несмотря на высокую размерность.

### 3.2.4 Фильтрация орбит с помощью сечения Пуанкаре

По завершении метода стрельбы мы имеем уточнённую начальную точку орбиты и её период. Для того, чтобы определить совпадающие орбиты нужно проинтегрировать систему с начальным значением в найденной точке на время, равное периоду. Для полученной периодической орбиты нужно найти точки пересечения с сечением Пуанкаре, на котором лежала начальная точка до применения метода стрельбы. После чего мы получаем дискретный набор точек на плоскости для каждой орбиты. Для того, чтобы найти одинаковые орбиты проще всего сначала провести группировку по количеству точек пересечений. Если орбита имеет уникальное относительно остальных орбит число пересечений с плоскостью Пуанкаре, то эта орбита не имеет аналогов среди найденных. В противном случае требуется дальнейший анализ. Предлагается следующий алгоритм сравнения орбит:

#### 1. Определение начального положения.

Поскольку для циклической последовательности не имеет значения с какой именно точки она начинается, нужно определить одинаковое начальное положение для каждой из них.

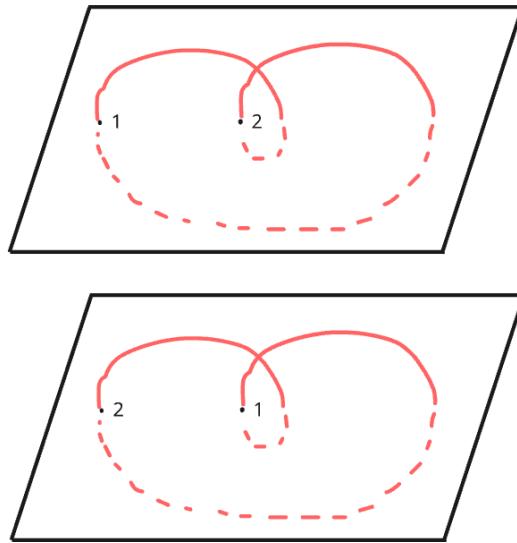


Рис. 10: Пример двух одинаковых орбит и их пересечений на плоскоти Пуанкаре.

Для предотвращения подобных ситуаций будем фильтровать точки по координате ‘ $x$ ’, то есть каждый набор будет начинаться с точки, у которой первая компонента минимальна относительно других точек.

#### 2. Попарное сравнение точек.

Рассмотрим пример с тремя орбитами, каждая из которых пересекает плоскость в одной точке. То есть нужно сравнить всего лишь 3 точки:

$$[1.0, 2.0], [1.1, 2.2], [3.0, 5.0]$$

Сравниваем каждую компоненту каждой точки с каждой другой. Если расстояние между ними больше заданного порога  $\delta$ , то запоминаем номера двух орбит

и считаем их разными.

Например, если  $\delta = 0.5$ , то после этого шага получим, что орбиты 1 и 3, а также 2 и 3 – разные. Очевидно, что это эквивалентно тому, что 1 и 2 орбиты одинаковые.

### 3. Построение графа и нахождение связных компонент.

Теперь необходимо из полученных данных нужно понять, какие все-таки орбиты отличаются.

Эта задача сводится к построению графа, где вершинами являются номера орбит, а ребро между двумя вершинами  $i$  и  $j$  существует, если на предыдущем шаге мы не определили, что  $i$  и  $j$  – разные.

После чего решается задача нахождения связных компонент в неориентированном графе. Связная компонента – это подмножество вершин графа, где каждая пара вершин связана путем, и ни одна из вершин не связана с вершинами вне этого подмножества.

Выходом метода являются списки номеров орбит, которые являются одинаковыми.

Итоговый алгоритм использует единственную плоскость Пуанкаре для всех начальных значений. Если бы это было иначе, описанный выше алгоритм не сработал бы, так как одинаковые орбиты могут иметь разные по значению точки пересечения с сечением Пуанкаре и, что ещё сильнее усложняет задачу, разные по количеству.

## 4 Анализ результатов и вывод

Полученное приложение является удобным средством для поиска периодических орбит в хаотических аттракторах. Анимация каждой из найденных орбит, позволяет взглянуть на устройство аттракторов под разными углами. Удобная и понятная система логирования делает результаты работы алгоритм более простыми для понимания.

Приложение работает очень эффективно, что позволяет пользователю узнать интересующую информацию о системе буквально за секунды. В качестве приведём некоторые периодические орбиты, которые были получены по результатам работы алгоритма. Для системы Лоренца было найдено несколько орбит, но конкретно эта очень похожа на орбиты, обнаруженные в статье [13]:

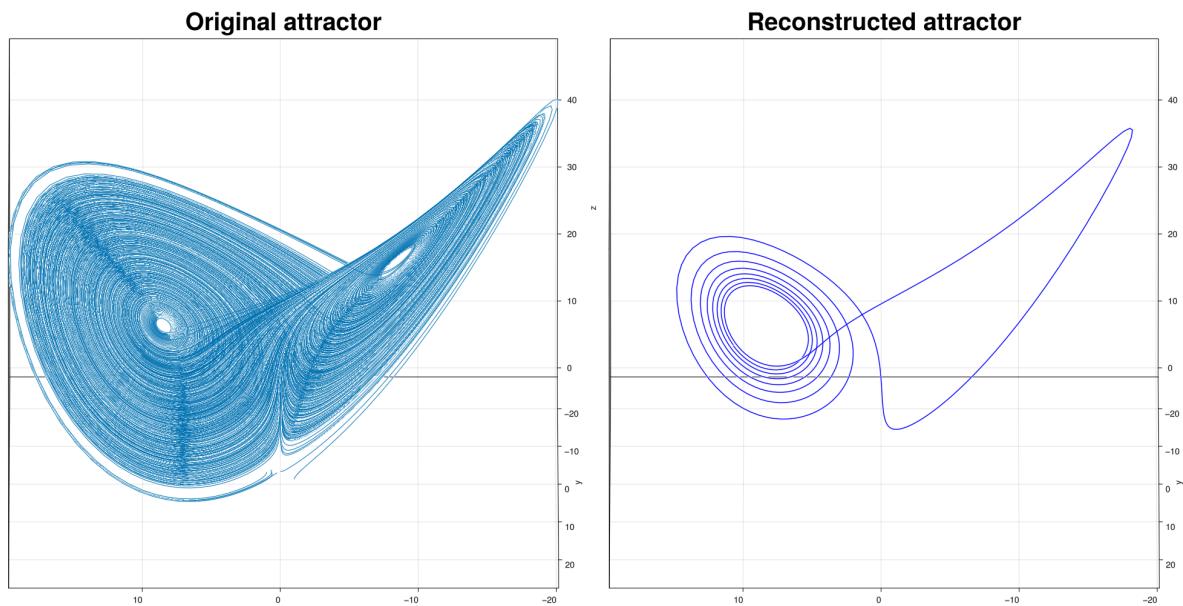


Рис. 11: Аттрактор системы Лоренца и найденная периодическая орбита.

Для модели Рёсслера:

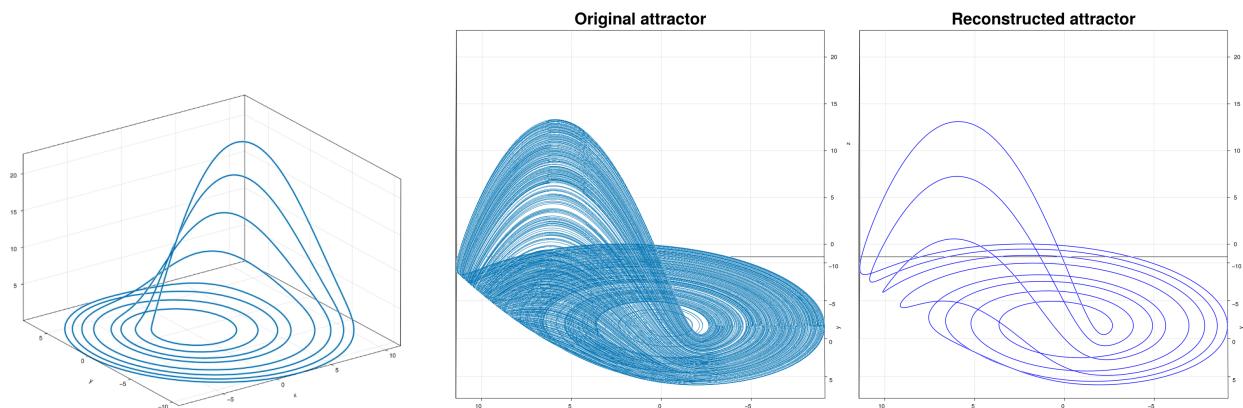


Рис. 12: Периодическая орбита для системы Рёсслера.

В системе Рикитаке была найдена маленькая периодическая траектория:

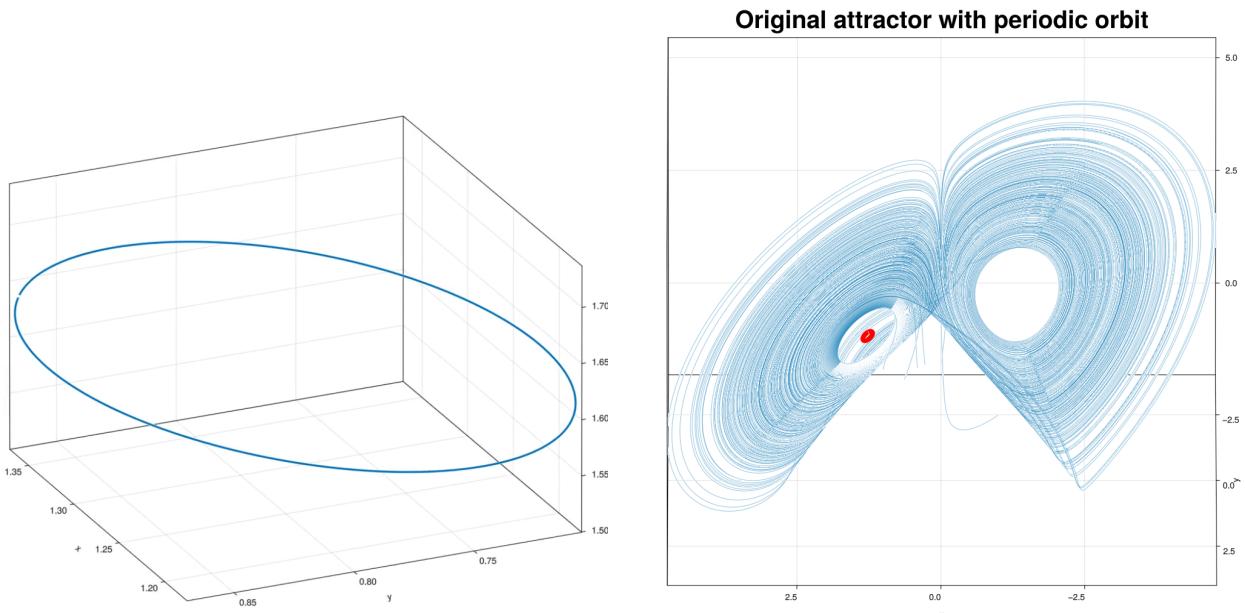


Рис. 13: Периодическая орбита в аттракторе Рикитаке.

В системе Чуа можно найти очень много разнообразных периодических орбит. Приведём в пример орбиту с наименьшим периодом из всех найденных.

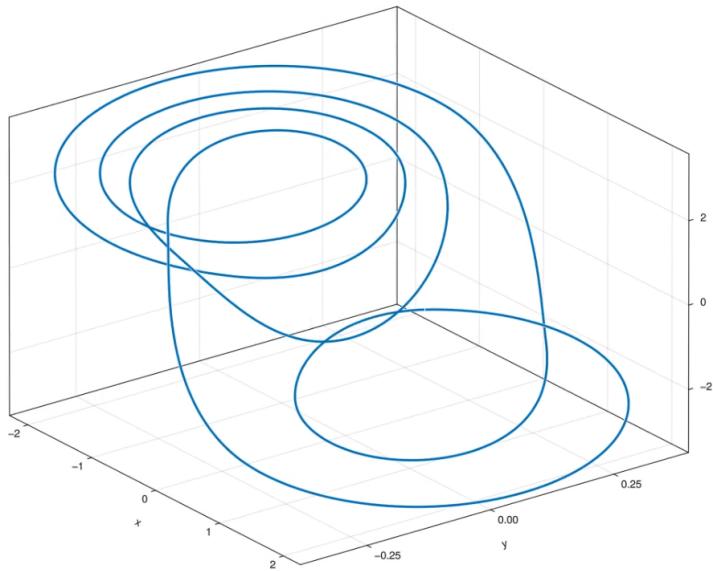


Рис. 14: Периодическая орбита системы Чуа.

Все найденные периодические орбиты короткие и могут подойти для эффективного вычисления различных показателей системы или построения шаблонов аттрактора [14]. Приложение имеет понятную структуру, удобные интерфейсы и легко подойдет для расширения. Исходный код доступен в приложении 4.

## 5 Руководство пользователя

### 5.1 Введение

В этом разделе будут описаны детали взаимодействия пользователя с графическим приложением.

Стоит оговориться, что для всех страниц:

- После нажатия на кнопку ‘Next‘ все содержимое удаляется и осуществляется переход на следующую страницу приложения.
- Будут описаны все возможные варианты действий, доступные пользователю, однако при переходе к следующему окну никакие изменения не будут применены, и все настройки останутся по умолчанию.

### 5.2 Первая страница

После запуска программы появляется первое окно, в котором пользователю предлагается выбрать одну из перечисленных динамических систем.

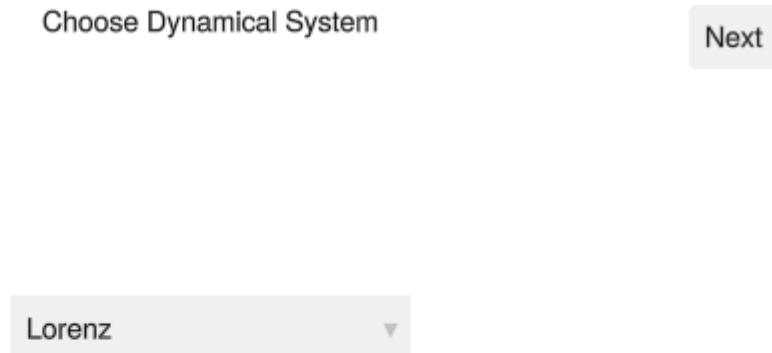


Рис. 15: Первая страница приложения.

Изначально выбрана система Лоренца, но это можно изменить, раскрыв меню.



Рис. 16: Выпывающее меню с популярными системами.

В самом низу располагается 'Enter', которая добавлена для дальнейшего расширения приложения и должна будет позволять написать уравнение системы самому.



Рис. 17: Примерный вариант того, как будут записываться уравнения системы.

### 5.3 Вторая страница

Здесь пользователю предоставляется возможность ввести время, на которое будет интегрирована система, и начальные условия.

Enter initial states

Next

Evaluation time =

$u_1 =$

$u_2 =$

$u_3 =$

Рис. 18: Интерфейс второй страницы.

Количество начальных значений можно регулировать с помощью кнопок ‘+’ и ‘-’, которые позволяют добавлять и удалять начальные значения соответственно. По умолчанию новые значения будут иметь псевдослучайное значение из интервала от 0 до 1.

## 5.4 Третья страница

Будем рассматривать её для трёх начальных условий.

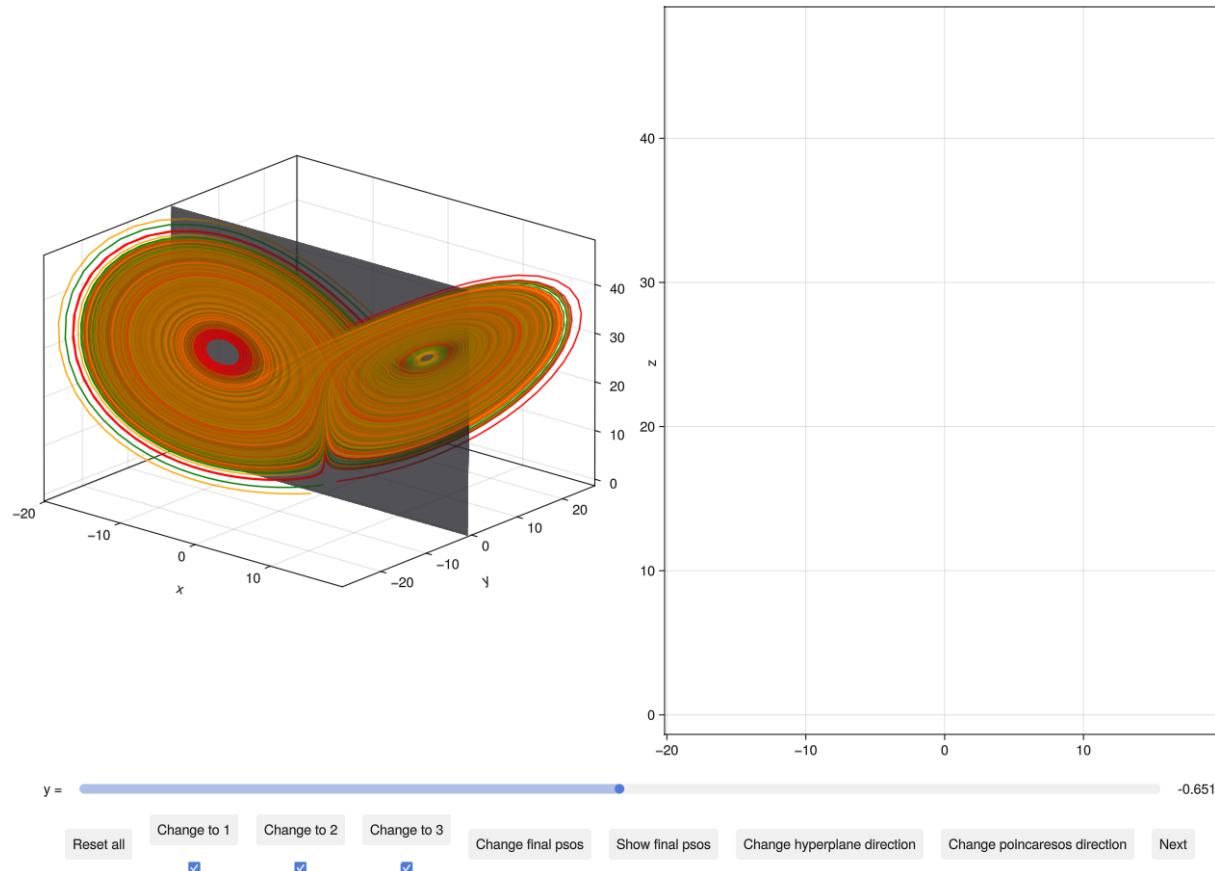


Рис. 19: Третья страница.

Слева расположена траектория системы, которую можно рассмотреть с разных сторон, нажав на неё левой кнопкой мыши и покрутив. Разные начальные условия изображены разными цветами. По середине атTRACTора можно заметить плоскость, которая и является сечением Пуанкаре. Для того, чтобы сдвинуть это сечение в одном из направлений необходимо передвинуть ползунок.

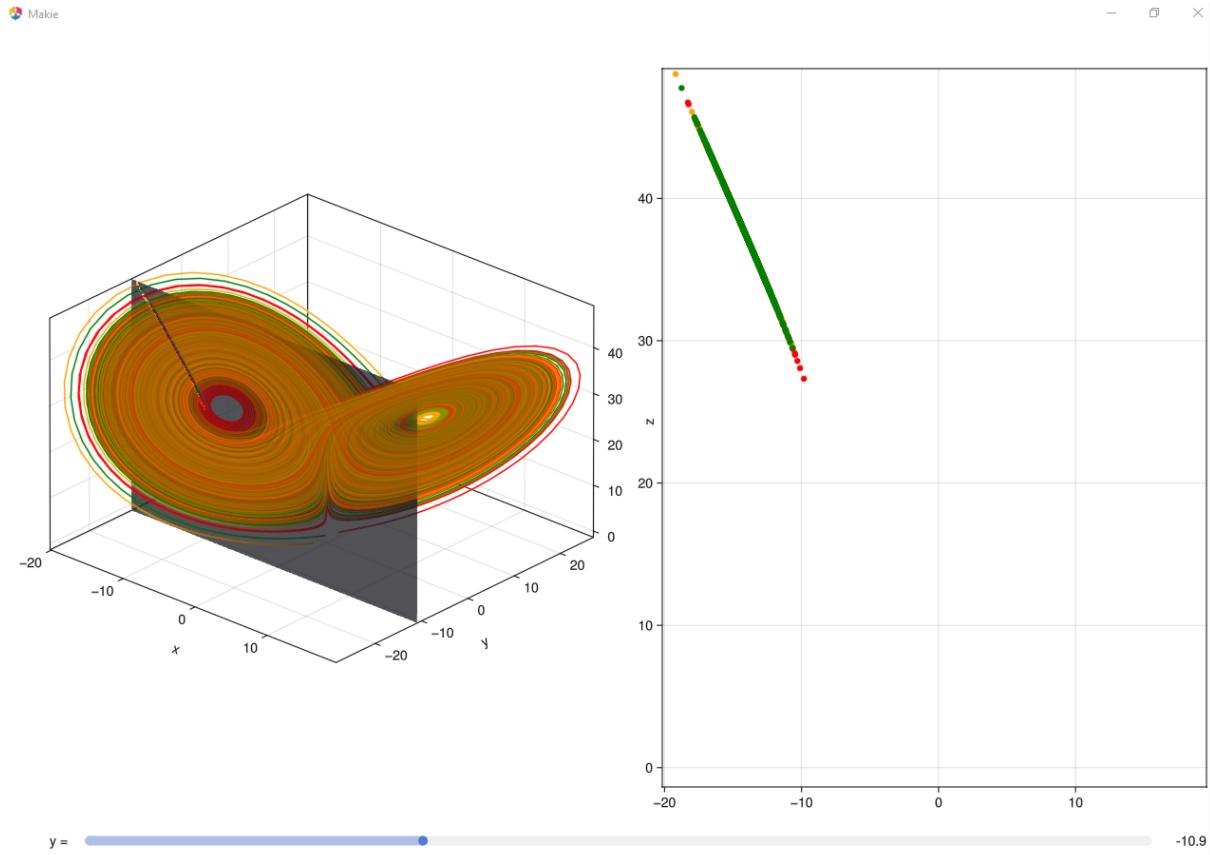


Рис. 20: Пересечение плоскости Пуанкаре аттрактором и отображение найденных точек.

Поскольку анализировать траектории для сразу всех начальных значений может быть не слишком удобно, можно посмотреть на каждую из них в отдельности, нажав на кнопку ‘Change to  $i$ ’, где  $i$  – номер интересующего начального значения.



Рис. 21: Кнопки для выбора каждого начального условия в отдельности.

Это может быть полезно для некоторых систем, например, у системы Чуа для некоторых начальных значений возможна нестабильность и устремление траектории в бесконечность. Переключатель под каждой кнопкой, позволяет пользователю использовать или не использовать определённое начальное значение для последующего анализа.

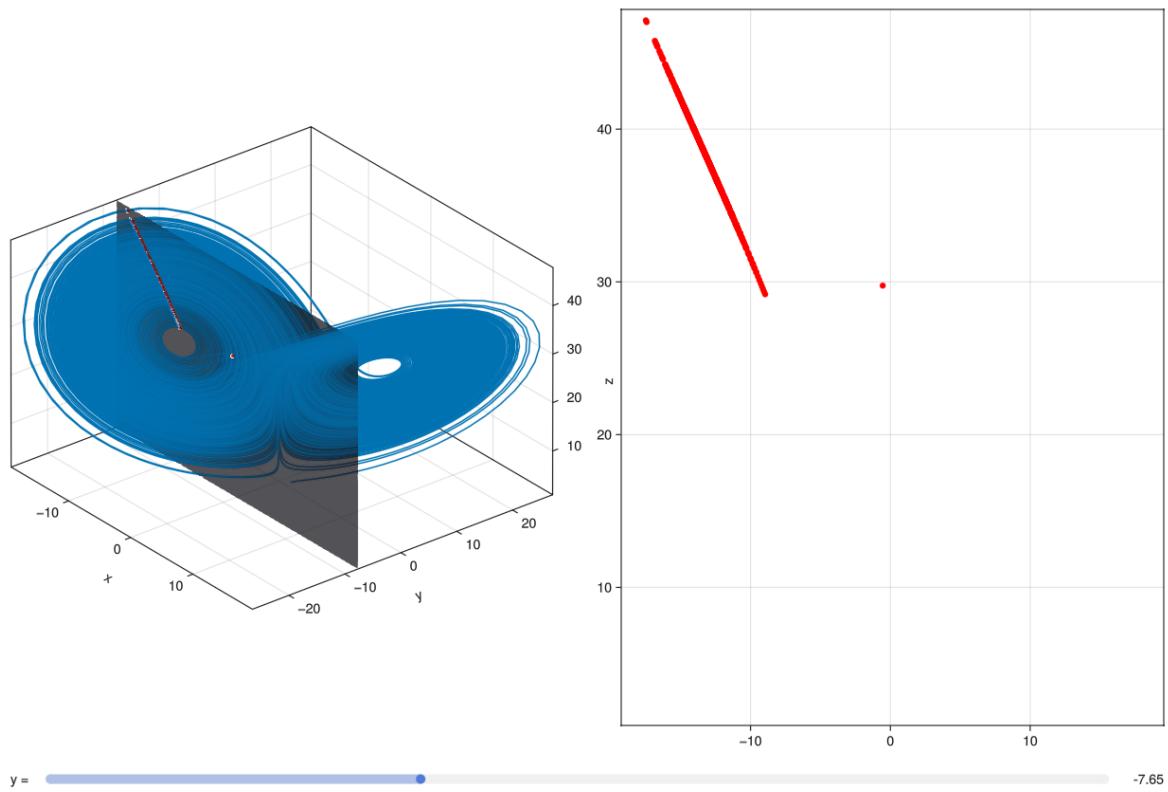


Рис. 22: Аттрактор после выбора одного из начальных значений.

Стоит отметить, что цвет пересечения выбранной траектории с секущей будет соответствовать цвету на общей картинке, а чтобы вернуться к ней, достаточно нажать на кнопку ‘Reset all’.

Поскольку одним из главных критериев для выбора сечения Пуанкаре – это отсутствие касания с точками, предусмотрена возможность рассматривать три разных сечения: для первого вектором нормали является ось  $OX$ , для второго –  $OY$ , а для третьего –  $OZ$ . По нажатию на кнопку ‘Change hyperplane direction’ произойдет изменение направления сечения.

**Change hyperplane direction**

**Change poincaresos direction**

Рис. 23: Кнопки для изменения направления плоскости и направления пересечений.

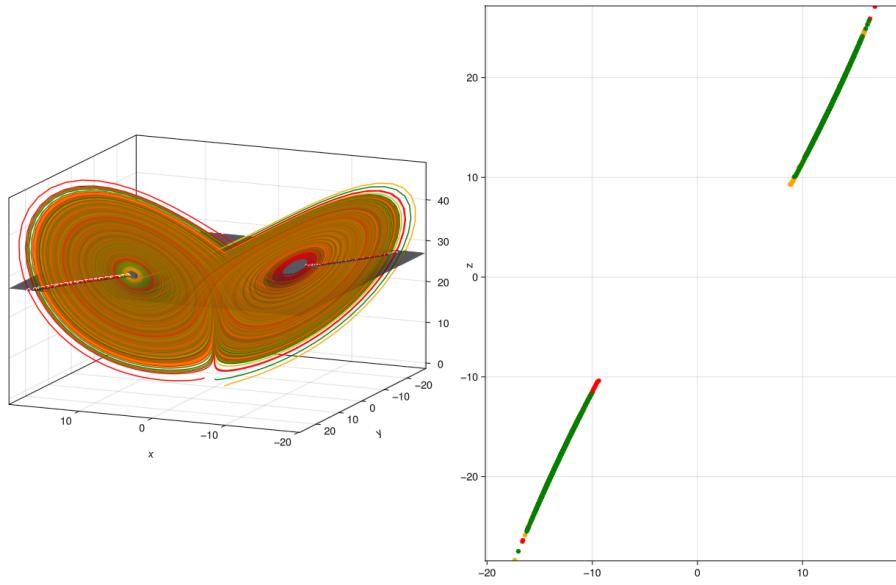


Рис. 24: Аттрактор после изменения направления плоскости.

Точка принадлежит сечению Пуанкаре тогда и только тогда, когда она пересекает его в определённом направлении. Это направление можно изменить с помощью кнопки ‘Change poincaresos direction’.

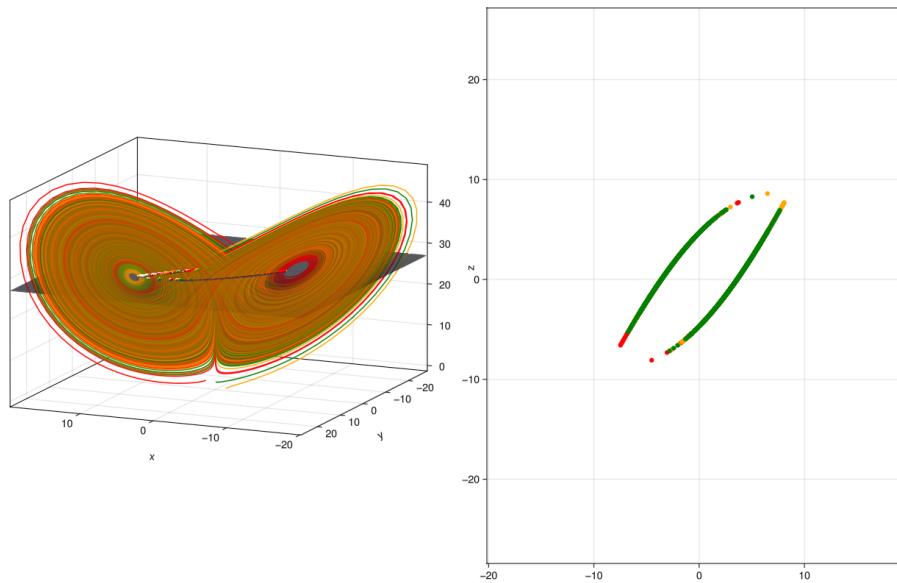


Рис. 25: Аттрактор после изменения направления, в котором учитываются пересечения с плоскостью.

Главное, что должен сделать пользователь — это выбрать интересующую его плоскость Пуанкаре. Для того, чтобы сохранить понравившийся ему результат нужно нажать на кнопку ‘Change final psos’.

**Change final psos**

**Show final psos**

Рис. 26: Кнопки для изменения и просмотра финального сечения Пуанкаре.

При этом для того, чтобы увидеть выбранное сечение достаточно нажать на кнопку ‘Show final psos’.

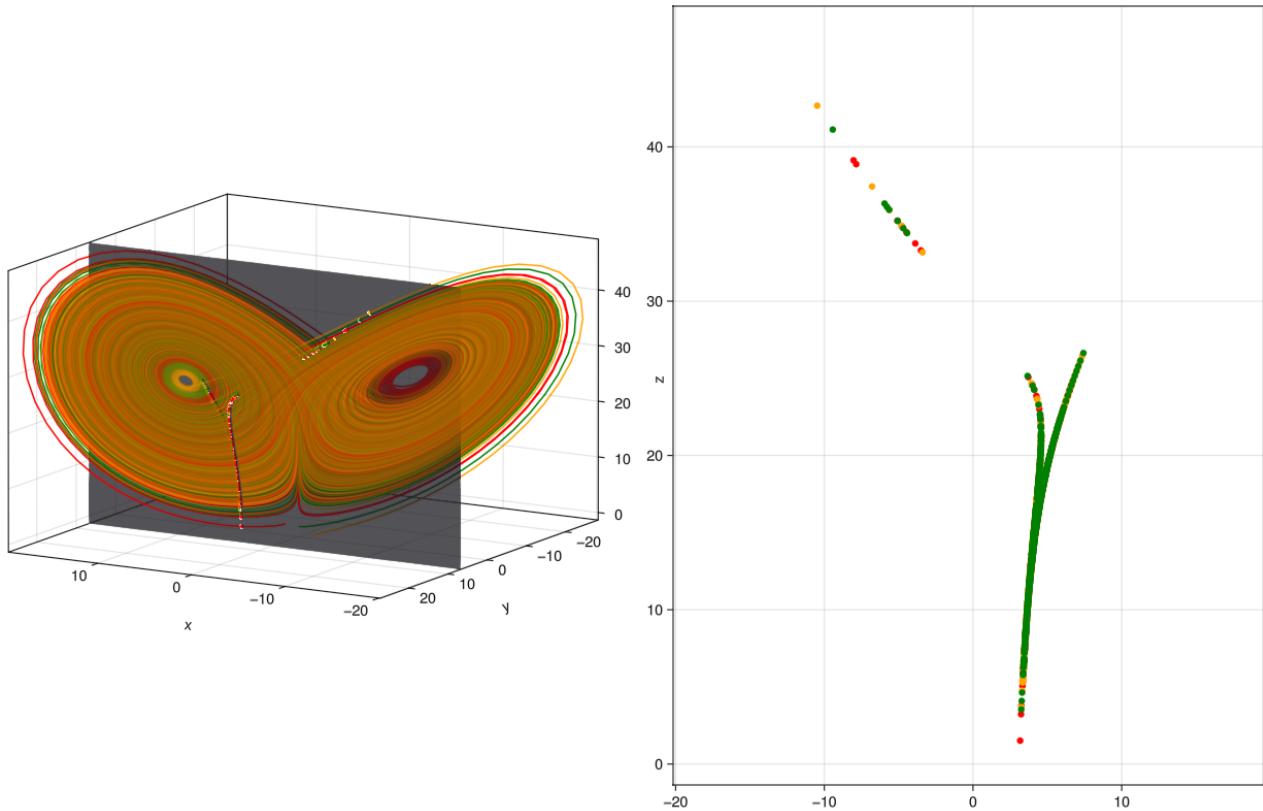


Рис. 27: Аттрактор и выбранное по умолчанию сечение Пуанкаре.

Пользователь может и не выбирать финальное сечение. В таком случае оно будет определено как плоскость, которая имеет наибольшее количество точек пересечения. Но это может привести к плохому выбору плоскости и отсутствию периодических орбит.

## 5.5 Чётвертая страница

Это последняя страница приложения, в которой пользователю предоставляется возможность задать следующие параметры для поиска периодических орбит:

1.  $t_{max}$ .  
Максимально возможное время периода.
2.  $t_{min}$ .  
Минимально возможное время периода.
3.  $\epsilon$ .  
Параметр, определяющий  $\epsilon$ -окрестность.
4.  $\delta$ .  
Этот параметр отвечает за фильтрацию орбит.

Enter  $\epsilon$ , maximum and minimum period      **Next**

$t_{max} =$	<input type="text" value="30.0"/>
$t_{min} =$	<input type="text" value="1.0"/>
$\epsilon =$	<input type="text" value="0.01"/>
$\delta =$	<input type="text" value="0.05"/>

Save recurrence matrices?

Save orbits animation?

Рис. 28: Четвёртая страница.

Также пользователю предоставляется выбор – сохранять ли рекуррентные матрицы и остальные картинки и сохранять ли анимацию орбит. При этом имеет смысл запускать алгоритм даже без графического представления, так как в программе предусмотрена система логирования, исходя из которой можно сделать выводы о результатах. Файл с логами можно найти в “<основная директория приложения>/logs/<название системы>/<название системы>\_i.txt”. Подробности, раскрывающие структуру приложения будут приведены в следующем разделе.

В следующем листинге приведен пример логирования для системы Лоренца:

Листинг 1: Логи для системы Лоренца.

```
t_max = 30.0, t_min = 1.0
ε = 0.01, δ = 0.05

Initial states are
u[1] = [1.0, 1.0, 1.0]
u[2] = [-1.0, -1.31596, -1.354461]
u[3] = [0.0, -0.25, 0.42081]

Poincare surface of sections defined by:
- Direction of movement along the axis y
- Value in that direction is -1.1523647141002435
- Direction of plane intersection comes with increasing
  coordinate (towards the positive normal)

Orbit 1
Begin state = [-0.6345982819363, -1.1523647141002, 9.825307999057]
End state   = [-0.6379665138511, -1.1583373829853, 9.83047100262]
Δ = 0.008583377295610334
T = 9.09
Psos intersection number is 6

Orbit 3
Begin state = [4.756903371969, -1.1523674315647, 30.254192440262]
End state   = [6.077974253953, -0.984495100698, 32.22301124463]
Δ = 2.3769006876924625
T = 11.269836201147363
Distance has been increased. Skipping orbit

Total orbits = 2

6 intersections have orbits with indexes
[i = 1]

11 intersections have orbits with indexes
[i = 2]
```

## 6 Руководство программиста

### 6.1 Введение

Программа реализована на языке программирования Julia версии 1.11.4.

## 6.2 Описание структуры программы

Приведём структуру файлов в виде древовидной иерархии:

```
||_ jl/
|   ||_ Algorithms/
|   |   ~_ LM.jl
|   ||_ Pages/
|   |   ||_ MainPage.jl
|   |   ||_ SecondPage.jl
|   |   ||_ ThirdPage.jl
|   |   ~_ FourthPage.jl
|   ||_ util.jl
|   ||_ poincare.jl
|   ~_ main.jl

||_ logs/
|   ||_ <system_name_1>/
|   |   ||_ <system_name_1>_1.txt
|   |   ||_ ...
|   |   ~_ <system_name_1>_N.txt
|
|   ~_ <system_name_2>/
|       ||_ <system_name_2>_1.txt
|       ||_ ...
|       ~_ <system_name_2>_M.txt

||_ Mkv/
|   ||_ <system_name_1>/
|   |   ||_ Orbit_1.txt
|   |   ||_ ...
|   |   ~_ Orbit_K.txt
|
|   ~_ <system_name_2>/
|       ||_ Orbit_1.txt
|       ||_ ...
|       ~_ Orbit_L.txt
|
~_ Png/
    ||_ <system_name_1>/
    |   ||_ Recurrence_Matrix_1.txt
    |   ||_ ...
    |   ~_ Recurrence_Matrix_P.txt
    |
    ~_ <system_name_2>/
        ||_ Recurrence_Matrix_1.txt
        ||_ ...
        ~_ Recurrence_Matrix_J.txt
```

Теперь последовательно опишем каждый из разделов:

## 1. Директория Algorithms

Папка предназначена для хранения файлов с исходным кодом, которые описывают различные численные методы и алгоритмы, используемые для уточнения начального приближения.

Файл LM.jl описывает метод стрельбы с использованием алгоритма Левенберга-Марквардта.

## 2. Директория Pages

Содержит четыре файла, каждый из которых включает функции, вызываемые из main.jl файла, и инициализирующие содержимое страницы.

## 3. Файлы util.jl, poincare.jl и main.jl

- util.jl содержит функцию, позволяющую удалять с экрана все элементы.
- poincare.jl содержит набор функций, некоторые из которых визуализируют третью страницу приложения, а некоторые нужны для правильной обработки поданных туда условий и начальных значений.
- main.jl – файл, собирающий всю программу воедино и содержащий функцию main.

## 4. Директория logs

Содержит папки для различных систем, в которых хранятся файлы с логами для соответствующей системы.

## 5. Директория Mkv

Для каждой из систем содержит анимацию найденных периодических орбит.

## 6. Директория Png

Содержит рекуррентные матрицы для каждой точки, у которой есть возвращение в  $\epsilon$ -окрестность этой точки.

## 6.3 Анализ логирования

Логирование – важная часть работы, поскольку она позволяет оценить результаты без графической интерпретации. Рассмотрим этот процесс подробнее:

1. Сначала описываются все параметры, которые задал пользователь:  $t_{max}$ ,  $t_{min}$ ,  $\epsilon$ ,  $\delta$  и начальные значения.
2. Не менее важным является описание плоскости Пуанкаре: направление вектора нормали к ней, значение свободной компоненты и используемое направление пересечения.
3. Перечисление всех найденных орбит. Для каждой указывается:
  - Номер.
  - Начальное состояние.

- Конечное состояние.
  - Расстояние между этими состояниями.
  - Период.
  - Количество точек пересечения с плоскостью Пуанкаре.
4. Поскольку в случае, когда применение метода стрельбы ухудшило расстояние между началом и концом периода мы пропускаем эту орбиту из дальнейшего анализа, удобно узнать сколько орбит удовлетворило этому условию.
  5. Заключительная часть, где выводятся группы орбит: сначала по количеству точек пересечения, а если в группе таковых несколько — информация о том сколько из них совпали и с кем.

## Список литературы

- [1] Predrag Cvitanović. Invariant measurement of strange sets in terms of cycles. *Physical Review Letters*, 61(24):2729, 1988.
- [2] Ditzia Auerbach, Predrag Cvitanović, Jean-Pierre Eckmann, Gemunu Gunaratne, and Itamar Procaccia. Exploring chaotic motion through periodic orbits. *Physical Review Letters*, 58(23):2387, 1987.
- [3] Predrag Cvitanović. Dynamical averaging in terms of periodic orbits. *Physica D: Nonlinear Phenomena*, 83(1-3):109–123, 1995.
- [4] George Datseris and Ulrich Parlitz. *Nonlinear dynamics: a concise introduction interlaced with code*. Springer Nature, 2022.
- [5] Кузнецов. Динамический хаос. 2001.
- [6] Masaki Sano and Yasuji Sawada. Measurement of the lyapunov spectrum from a chaotic time series. *Physical review letters*, 55(10):1082, 1985.
- [7] Predrag Cvitanovic, Roberto Artuso, Ronnie Mainieri, Gregor Tanner, Gábor Vattay, Niall Whelan, and Andreas Wirzba. Chaos: classical and quantum. *ChaosBook.org (Niels Bohr Institute, Copenhagen 2005)*, 69:25, 2005.
- [8] Zbigniew Galias. A fast method to find periodic orbits in chaotic attractors with applications to the rössler system. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(9):3659–3663, 2023.
- [9] Robert Gilmore and Marc Lefranc. *The topology of chaos: Alice in stretch and squeezeland*. John Wiley & Sons, 2012.
- [10] J. W. Milnor. Attractor. *Scholarpedia*, 1(11):1815, 2006. revision #186525.
- [11] Norbert Marwan, M Carmen Romano, Marco Thiel, and Jürgen Kurths. Recurrence plots for the analysis of complex systems. *Physics reports*, 438(5-6):237–329, 2007.
- [12] Wynand Dednam and Andre E Botha. Optimized shooting method for finding periodic orbits of nonlinear dynamical systems. *Engineering with Computers*, 31:749–762, 2015.
- [13] Chengwei Dong. Topological classification of periodic orbits in lorenz system. *Chinese Physics B*, 27(8):080501, 2018.
- [14] Joan S Birman and Robert F Williams. Knotted periodic orbits in dynamical systems i: Lorenz’s equations. *Topology*, 22(1):47–82, 1983.

## 7 Приложения

### 7.1 Приложение 1

Листинг 2: Исходный код для Рис. 1.

```
using DynamicalSystems
using CairoMakie
using LaTeXStrings

function main()
    Lorenz = Systems.lorenz()

    u0 = [10, 10, 10.9]

    total_time = 15

    range = -0.001:0.0005:0.001

    tr = [trajectory(Lorenz, total_time, u0 .+ ε; Dt = 0.01)
          for ε in range]

    CairoMakie.activate!()
    fig = Figure(size = (919, 237))
    ax = Axis(fig[1, 1]; xlabel = L"\mathbf{t}", ylabel = L"\mathbf{z}")

    for i in 1:length(range)
        lines!(ax, tr[i][2], columns(tr[i][1])[3])
    end

    save_path = joinpath(dirname(@_FILE_), "1.png")
    save(save_path, fig)
end

main()
```

### 7.2 Приложение 2

Листинг 3: Исходный код для Рис. 2.

```
using CairoMakie
using LaTeXStrings

function main()
    fig = Figure(size = (600, 200))

    ax1 = Axis(fig[1, 1], title = L"l_1")
```

```

ax2 = Axis(fig[1, 2], title = L"l_2")
axi = Axis(fig[1, 3], title = L"l_\infty")

for ax in [ax1, ax2, axi]
    limits!(ax, -2, 2, -2, 2)
    ax.xticksvisible = false
    ax.yticksvisible = false
    ax.xticklabelsvisible = false
    ax.yticklabelsvisible = false
    ax.xgridvisible = false
    ax.ygridvisible = false
end

θ = LinRange(0, 2π, 1000)

l1_x = [sign(cos(t)) * abs(cos(t))^2 for t in θ]
l1_y = [sign(sin(t)) * abs(sin(t))^2 for t in θ]
lines!(ax1, l1_x, l1_y)
scatter!(ax1, 0, 0, markersize = 5)

l2_x = [cos(t) for t in θ]
l2_y = [sin(t) for t in θ]
lines!(ax2, l2_x, l2_y)
scatter!(ax2, 0, 0, markersize = 5)

li_x = [1, 1, -1, -1, 1]
li_y = [1, -1, -1, 1, 1]
lines!(axi, li_x, li_y)
scatter!(axi, 0, 0, markersize = 5)

save_path = joinpath(dirname(@__FILE__), "2.png")
save(save_path, fig)
end

main()

```

### 7.3 Приложение 3

Листинг 4: Исходный код для Рис. 3.

```

using GLMakie

function main()
    ε           = 0.4
    length      = 10
    n_points   = 100

```

```

x      = range(-length, stop = length, length = n_points)
y_main = x .^ 2 * 0.005
y_dash = .-(x .^ 2 * 0.001)
z      = zeros(n_points)

l, r = -2, 2

θ = LinRange(0, 2π, 100)

filtered_indices = findall(l .≤ x .≤ r)

len = size(filtered_indices, 1)

t = range(l, r, len)

X = [x[filtered_indices[i]] + t[i] for θ_i in θ, i in 1:len]
Y = [y_main[filtered_indices[i]] + ε * cos(θ_i)
      for θ_i in θ, i in 1:len]
Z = [ε * sin(θ_i) for θ_i in θ, t_i in t]

fig = Figure(size = (800, 600))
ax = Axis3(fig[1, 1])

lines!(ax, x, y_main, z, color = :black, linewidth = 2)
lines!(ax, x, y_dash .- ε * 0.85, z, color = :black, linestyle = :dash)

lines!(ax, x[filtered_indices],
       y_main[filtered_indices],
       z[filtered_indices],
       color = :red, linewidth = 2)

lines!(ax, x[filtered_indices],
       y_dash[filtered_indices] .- ε * 0.85,
       z[filtered_indices],
       color = :red, linestyle = :dash)

surface!(ax, X, Y, Z,
         color = fill(:gray, size(X)...),
         alpha = 0.5, shading = NoShading)

save_path = joinpath(dirname(@__FILE__), "3.png")
save(save_path, fig)
end

main()

```

## 7.4 Приложение 4

### 7.4.1 Исходный код файла main.jl

```
include("Pages/MainPage.jl")
include("Pages/SecondPage.jl")
include("Pages/ThirdPage.jl")
include("Pages/FourthPage.jl")

function main()
    GLMakie.activate!()

    screen, fig, ds, rule, pageClosedM = initMainPage()

    display(screen, fig)

    pageClosedS = Observable(false)
    pageClosedT = Observable(false)
    total_time = Observable(1000)
    i_orbits = Observable(Bool[])
    u0s = Observable(Vector{Float64}[])
    system = Observable{Any}(" ")
    psos = Observable{Any}(" ")

    on(pageClosedM) do _ initSecondPage(fig, ds, rule, pageClosedS, u0s,
                                         total_time,
                                         i_orbits, system, psos)

    on(pageClosedS) do _ initThirdPage(screen, fig, u0s, total_time,
                                         system, psos,
                                         i_orbits) end

    on(pageClosedT) do _ initFourthPage(screen, fig, system, psos,
                                         i_orbits, ds)
end

main()
```

### 7.4.2 Исходный код файла MainPage.jl

```
using GLMakie
using DynamicalSystems
using GridLayoutBase

include("../util.jl")
```

```

# TODO: Make function that parse strings to DS rule
function parse_strings()
    return true
end

function create_del_button(str, grid)
    delete!(contents(grid[][3])[4])

    str[] = push!(str[], " ")

    tb4 = Textbox(grid[][3][4, 1], placeholder = "Enter 4")
    on(tb4.stored_string) do s str[][4] = s end

    del_btn = Button(grid[][3][5, 1], label = "-")
    on(del_btn.clicks) do _
        delete!(del_btn)
        delete!(tb4)
        trim!(grid[][3])

        str[] = str[][1:3]

        add_btn = Button(grid[][3][4, 1], label = "+")
        on(add_btn.clicks) do _ create_del_button(str, grid) end
    end
end

function create_button_next_main(fig, grids, ds, flag)
    buttonNext = Button(grids[1][1, 3], label = "Next",
                        halign = :right, valign = :top,
                        tellwidth = false, tellheight = false)
    on(buttonNext.clicks) do _
        if ds != "Enter"
            delete_all!(fig)
            flag[] = true
        else
            res = parse_strings()

            if res # Parse Ok
                delete_all!(fig)
                flag[] = true
            else
                @error "Parse error"
            end
        end
    end
end

```

```

function initMainPage()
    screen = GLMakie.Screen()
    fig = Figure()

    grids = Observable([GridLayout(fig[i, 1], tellwidth = false,
                                  tellheight = false) for i in 1:3])

    Label(grids[][1][1, 2], "Choose Dynamical System",
          halign = :center,
          valign = :top,
          tellwidth = false,
          tellheight = false)
    rowsize!(grids[][1], 1, Relative(1/5))

    menu = Menu(grids[][2][1, 1],
                options = ["Lorenz", "Henonheiles", "Rikitake",
                           "Gissinger", "Roessler", "Chua", "Enter"],
                default = "Lorenz",
                width = 200,
                halign = :center,
                valign = :top,
                tellwidth = false,
                tellheight = false)
    rowsize!(grids[][2], 1, Relative(1/5))

    ds = Observable("Lorenz")
    str = Observable(String[" ", " ", " "])
    flag = Observable(false)
    created_flag = false
    rule = nothing

    on(menu.selection) do s
        if s == "Enter"
            if !created_flag
                tb1 = Textbox(grids[][3][1, 1], placeholder = "Enter 1")
                on(tb1.stored_string) do s str[][1] = s end

                tb2 = Textbox(grids[][3][2, 1], placeholder = "Enter 2")
                on(tb2.stored_string) do s str[][2] = s end

                tb3 = Textbox(grids[][3][3, 1], placeholder = "Enter 3")
                on(tb3.stored_string) do s str[][3] = s end

            butCreate = Button(grids[][3][4, 1], label = "+")
            on(butCreate.clicks) do _ create_del_button(str, grids) end
        end
    end

```

```

        created_flag = true
    else
        if created_flag
            for child in contents(grids[] [3]) delete!(child) end
        trim!(grids[] [3])

        str = Observable(String[" ", " ", " "])
        created_flag = false
    end

    ds[] = s
end
end

create_button_next_main(fig, grids[], ds, flag)

return screen, fig, ds, rule, flag
end

```

#### 7.4.3 Исходный код файла SecondPage.jl

```

using DynamicalSystems, OrdinaryDiffEq
using GLMakie
using LaTeXStrings

include("../util.jl")

function create_u_boxes(grid, i, u0s, D)
    latex_expr = latexstring("u_{", i, "} = ")
    Label(grid[i, 1], latex_expr)

    for j in 1:D
        tb = Textbox(grid[i, j + 1], placeholder = string(u0s[] [i] [j]),
                     validator = Float64)
        on(tb.stored_string) do u u0s[] [i] [j] = parse(Float64, u) end
    end
end

function add_u0(u0s, grids, D)
    len = length(u0s[])
    if len < 6
        insertrows!(grids[] [3], len + 1, 1)
        u0s[] = push!(u0s[], rand(Float64, D) .% 1)
        create_u_boxes(grids[] [3], len + 1, u0s, D)
    end
end

```

```

end

function del_u0(u0s, grids, D)
    len = length(u0s[])

    content = contents(grids[] [3])
    filter!(elt->(return !(typeof(elt) <: Button)), content)
    len_contents = length(content)

    if len > 1
        for i in 0:D
            delete!(content[len_contents - D + i])
        end
    end
    trim!(grids[] [3])

    u0s[] = u0s[] [1:len - 1]
end

function create_button_next(fig, grid, flag)
    buttonNext = Button(grid[1, 3], label = "Next",
                        halign = :right, valign = :top,
                        tellwidth = false, tellheight = false)
    on(buttonNext.clicks) do _
        delete_all!(fig)
        flag[] = true
    end
end

function lorenz_u0s(u0s)
    u0s[] = push!(u0s[], [1.0, 1.0, 1.0])
    u0s[] = push!(u0s[], [-1.0, -1.31596, -1.354461])
    u0s[] = push!(u0s[], [0.0, -0.25, 0.42081])
end

function henonheiles_u0s(u0s)
    u0s[] = push!(u0s[], [0.0, -0.25, 0.42081, 0.0])
    u0s[] = push!(u0s[], [0.0, 0.1, 0.5, 0.0])
    u0s[] = push!(u0s[], [0.0, -0.31596, 0.354461, 0.0591255])
end

function rikitake_u0s(u0s)
    u0s[] = push!(u0s[], [1.0, 1.0, 1.0])
    u0s[] = push!(u0s[], [-1.0, -1.31596, -1.354461])
    u0s[] = push!(u0s[], [0.0, -0.25, 0.42081])
end

```

```

function gissinger_u0s(u0s)
    u0s[] = push!(u0s[], [1.0, 1.0, 1.0])
    u0s[] = push!(u0s[], [-1.0, -1.31596, -1.354461])
    u0s[] = push!(u0s[], [0.0, -0.25, 0.42081])
end

function roessler_u0s(u0s)
    u0s[] = push!(u0s[], [1.0, 1.0, 1.0])
    u0s[] = push!(u0s[], [-1.0, -1.31596, -1.354461])
    u0s[] = push!(u0s[], [0.0, -0.25, 0.42081])
end

function chua_u0s(u0s)
    u0s[] = push!(u0s[], [0.0, -0.25, 0.42081])
    u0s[] = push!(u0s[], [1.0, 1.0, 1.0])
    u0s[] = push!(u0s[], [-1.0, -1.31596, -1.354461])
end

function initSecondPage(fig, ds, rule, flag, u0s, total_time, system)
    grids = Observable([GridLayout(fig[i, 1], tellwidth = false,
                                    tellheight = false) for i in 1:3])

    Label(grids[][1][1, 2], "Enter initial states",
          halign = :center, valign = :top,
          tellwidth = false, tellheight = false)
    create_button_next(fig, grids[][1], flag)
    rowsize!(grids[][1], 1, Relative(1/5))

    D = 0

    if ds != "Enter"
        ds[] == "Lorenz"      && (system[] = Systems.lorenz();
lorenz_u0s(u0s))
        ds[] == "Rikitake"    && (system[] = Systems.rikitake();
rikitake_u0s(u0s))
        ds[] == "Gissinger"   && (system[] = Systems.gissinger();
gissinger_u0s(u0s))
        ds[] == "Roessler"    && (system[] = Systems.roessler();
roessler_u0s(u0s))
        ds[] == "Chua"         && (system[] = Systems.chua();
chua_u0s(u0s))
        D = dimension(system[])
    else
        # Definition of DS via parsed rule...
        system[] = Systems.lorenz()
    end
end

```

```

D = dimension(system[])

if D == 3
    lorenz_u0s(u0s)
elseif D == 4
    henonheiles_u0s(u0s)
end
end

Label(grids[][2][1, 1], "Evaluation time = ")

tb = Textbox(grids[][2][1, 2], placeholder = string(total_time[]),
             validator = Int64)
on(tb.stored_string) do t total_time[] = parse(Int, t) end

rowsize!(grids[][2], 1, Relative(1/5))

for i in 1:3 create_u_boxes(grids[][3], i, u0s, D) end

butCreate = Button(grids[][3][4, 2], label = "+")
on(butCreate.clicks) do _ add_u0(u0s, grids, D) end

butCreate = Button(grids[][3][4, 3], label = "-")
on(butCreate.clicks) do _ del_u0(u0s, grids, D) end
end

```

#### 7.4.4 Исходный код файла ThirdPage.jl

```

include("../poincare.jl")
include("../util.jl")

function initThirdPage(screen, fig, u0s, total_time, system, final_psos,
                      current_size = Makie.size(fig.scene))
    delete_all!(fig)

    trs = [trajectory(system[], total_time[], u0, Δt = 0.01) for u0 ∈
           u0s[]]
    trs = filter_orbits(trs)
    for _ in eachindex(u0s[])
        i_orbits[] = push!(i_orbits[], true)
    end

    D = dimension(system[])

    best_psos = find_best_poincaresos(trs, D, total_time[])

```

```

if D == 4
    if typeof(trs) <: Vector{Tuple{S, T}} where
        {S<:AbstractStateSpaceSet, T<:StepRangeLen}
        trs = [trs[i][1][:, 1:3] for i in eachindex(trs)]
    else
        trs = [trs[i][:, 1:3] for i in eachindex(trs)]
    end

    psos_ = []

    for k in eachindex(best_psos[])
        psos_j = []
        for j in 1:3
            push!(psos_j, (s = best_psos[] [k][j].s[:, 1:3],
                           v = best_psos[] [k][j].v))
        end
        push!(psos_, psos_j)
    end

    best_psos[] = psos_
end

scan_poincaresos(trs, screen, current_size, best_psos, final_psos,
                  i_orbits, total_time[], pageClosedT;
                  linekw = (transparency = true,))
end

```

#### 7.4.5 Исходный код файла FourthPage.jl

```

using LaTeXStrings
using CairoMakie, GLMakie
using SparseArrays
using Distances: evaluate
using LinearAlgebra
using Base.Filesystem
using LightGraphs

include("../Algorithms/LM.jl")
include("../util.jl")

# Add png

function callback(system, psos, v, j, dir, u0s, t_max, t_min, ε, δ, Δt, t
R, res = [], [])

```

```

current_dir = dirname(dirname(dirname(@__FILE__)))

logs_path = joinpath(current_dir, "logs", "$ds")

!isdir(logs_path) && mkdir(logs_path)

file_prefix = "$(ds)_"
file_extension = ".txt"

new_file_path = create_next_file(logs_path, file_prefix, file_extension)

log = open(new_file_path, "w")

write(log, "t_max = $(t_max[]), t_min = $(t_min[])\n")
write(log, "ε = $(ε[]), δ = $(δ[])\n\n")
write(log, "Initial states are\n")
for i in eachindex(u0s)
    write(log, "u[$i] = $(u0s[i])\n")
end
write(log, "\nPoincare surface of sections defined by:\n")
write(log, " - Direction of movement along the axis $((x':z')[j])\n")
write(log, " - Value in that direction is $v\n")
write(log, " - Direction of plane intersection comes with ")
write(log, "$(if dir == true "increasing coordinate (towards the positive "
            "else "decreasing coordinates (towards the negative normal
            "end))")
write(log, "\n\n")

if saveR[]
    png_path = joinpath(current_dir, "Png", "$ds")
    rm(png_path, recursive = true, force = true)
    mkdir(png_path)
end

if save0[]
    mkv_path = joinpath(current_dir, "Mkv", "$ds")
    rm(mkv_path, recursive = true, force = true)
    mkdir(mkv_path)
end

for i in eachindex(psos)
    push!(R, [])
    trs = [trajectory(system, t_max[], u0, Δt = Δt) for u0 in psos[i]]

    Rs = [let d = distance_col_s(tr[1], ε[], Int(t_min[] / Δt))
          if d.b == true
              (r = d.r, )

```

```

        end
    end for tr in trs]

for k in eachindex(Rs)
    if !isnothing(Rs[k])
        push!(R[i], [trs[k][1][1], Rs[k].r])
    end
end
end

(all(length(R[i]) == 0 for i in eachindex(psos))) && (write(log, "0 orb
close(log);
return)

for i in eachindex(R)
    for j in eachindex(R[i])
        push!(res, let lm = lm(system, collect(R[i][j][1]), R[i][j][2] * Δt
            !isnothing(lm) && (lm)
            end)
    end
end

filter!(elt->(return !(elt == false)) , res)

all(length(res[i]) == 0 for i in eachindex(res)) && (write(log, "0 orbit
close(log);
return)

psos_res = []

for i in eachindex(res)
    write(log, "\nOrbit $i\n")

    Δ = norm(res[i].state[end] .- res[i].state[1])

    write(log, "Begin state = $(res[i].state[1])\n",
          "End state   = $(res[i].state[end])\n",
          "Δ = $Δ\n",
          "T = $(res[i].period)\n")

    Δ > ε[] && (write(log, "Distance has been increased. Skipping orbit\n"

try
    psos_ = poincaresos(res[i].state, (j, v); direction = -1, warning =
        push!(psos_res, (psos = psos_, i = i))
catch err

```

```

    write(log, "Psos = empty\n")
    continue
end

write(log, "Psos intersection number is $(length(psos_res[end].psos))"
end

write(log, "\nTotal orbits = $(length(psos_res))\n")

psos_res = group_psos(psos_res)

d = 3
if j == 1
  d = 2
elseif j == 2 || j == 3
  d = 1
end

animate = []

for i in eachindex(psos_res)
  len_i = length(psos_res[i])

  write(log, "\n$(length(psos_res[i][1].psos)) intersections have orbit"
  write(log, "[i = $(psos_res[i][1].i)]")

  for j in 2:len_i
    write(log, "\n i = $(psos_res[i][j].i)")
  end

  write(log, "]\\n")

  if len_i != 1
    psos = []

    # Find minimal component in SSSet and rotate it so
    # element with minimal component became first in SSSet
    for j in eachindex(psos_res[i])
      k = min_index(psos_res[i][j].psos, d)
      if k != 1
        push!(psos, vcat(psos_res[i][j].psos[k:end], psos_res[i][j].psos[1:k-1]))
      elseif k == 1
        push!(psos, psos_res[i][j].psos)
      end
    end

    # Group psoses by intersection number with psos
  end
end

```

```

grouped_ind = compare_psoses(psos, δ[])
if length(grouped_ind) == 1
    write(log, "All orbits were same, we'll consider first of them\n")
    save0[] && (append!(animate, psos_res[i][1].i))

elseif length(grouped_ind) != len_i
    write(log, "Orbits with indexes ")

    filtered_group      = filter(x -> (length(x) != 1), grouped_ind)
    filtered_group_len = length(filtered_group)

    for j in 1:filtered_group_len
        if j != filtered_group_len
            write(log, "$(map(x -> x.i, psos_res[i][[filtered_group[j]...]]))")
        else
            write(log, "$(map(x -> x.i, psos_res[i][[filtered_group[j]...]]))")
        end
    end

    write(log, " were same\n")

    if save0[] || saveR
        for j in eachindex(grouped_ind)
            append!(animate, psos_res[i][grouped_ind[j][1]].i)
        end
    end
end

else
    write(log, "All orbits were different\n")
    if save0[] || saveR
        for j in 1:len_i
            append!(animate, psos_res[i][j].i)
        end
    end
end
end

if save0[]
    GLMakie.activate }()
    for i in animate
        min, max = minmaxima(res[i].state)
    end
end

```

```

save_path = joinpath(mkv_path, "Orbit_$(i).mp4")

len   = length(res[i].state)
time  = Observable(0.0)
k     = @lift(round(Int, clamp(len * $time, 1, len)))
azim = Observable(1.275)
z     = @lift(clamp(1.0 * $azim, 1.275, 2pi + 1.275))

figi = Figure(size = (1000, 1000))

axi  = Axis3(figi[1, 1], azimuth = @lift($z * pi))
limits!(axi, min[1], max[1],
        min[2], max[2],
        min[3], max[3])

dynamic_mas = @lift(res[i].state[1:$k])
lines!(axi, dynamic_mas, color = RGBA{Float32}(0.0, 0.44705883, 0.6)

eval_range    = range(0, 1, step = 1 / len)
rotate_range  = range(1, 3, step = 10e-3)
rotate_range_ = range(3, 5, step = 10e-3)
timestamps    = vcat(eval_range, rotate_range, rotate_range_)

record(figi, save_path, timestamps) do t
    if t <= 1
        time[] = t
    elseif t < 3
        azim[] = t - 1 + 1.275
    elseif t == 3
        i1, i2 = find_element_with_i(psos_res, i)
        scatter!(axi, psos_res[i1][i2].psos, color = :red, markersize =
            ss = collect(copy(min))
            ws = collect(copy(max) .- copy(min))

            ws[j] = 0
            ss[j] = v

            o = Makie.Point3f(ss...)
            w = Makie.Point3f(ws...)

            mesh!(axi, Makie.Rect3f(o, w); color = RGBAf(0.2, 0.2, 0.25, 0.5)
    elseif t <= 5
        azim[] = t - 1 + 1.275
    end
end

```

```

    end
end

if saveR[]
    GLMakie.activate!()
    Y = [trajectory(system, total_time, u0; Dt = Δt)[1] for u0 in u0s]

    min, max, _ = total_minmaxima(Y)

    for i in animate
        save_path = joinpath(png_path, "Orbit_$(i).png")
        figi = Figure(size = (2000, 1000))
        axi1 = Axis3(figi[1, 1], azimuth = 0.5 * pi, titlesize = 40, title =
            limits!(axi1, min[1], max[1],
                    min[2], max[2],
                    min[3], max[3])
        for j in eachindex(Y)
            lines!(axi1, Y[j][:, 1], Y[j][:, 2], Y[j][:, 3]; linewidth = 1, c
        end

        axi2 = Axis3(figi[1, 2], azimuth = 0.5 * pi, titlesize = 40, title =
            limits!(axi2, min[1], max[1],
                    min[2], max[2],
                    min[3], max[3])
        lines!(axi2, res[i].state[:, 1], res[i].state[:, 2], res[i].state[:, 3]; linewidth = 1, color = :red)
        save(save_path, figi)

        save_path = joinpath(png_path, "Orbit_$(i)_1.png")
        figi = Figure(size = (1000, 1000))
        axi1 = Axis3(figi[1, 1], azimuth = 0.5 * pi, titlesize = 40, title =
            limits!(axi1, min[1], max[1],
                    min[2], max[2],
                    min[3], max[3])

        for j in eachindex(Y)
            lines!(axi1, Y[j][:, 1], Y[j][:, 2], Y[j][:, 3]; linewidth = 1, color = :blue)
        end

        lines!(axi1, res[i].state[:, 1], res[i].state[:, 2], res[i].state[:, 3]; linewidth = 1, color = :red)
        save(save_path, figi)
    end
end

close(log)
end

```

```

function create_button_next(fig, callback, args...)
    buttonNext = Button(fig[1, 2], label = "Next")
    on(buttonNext.clicks) do _
        call = @async callback(args...)
        delete_all!(fig)

        try
            wait(call)
        catch error
            showerror(stderr, error)
            println(stderr)
            @error "An error occurred" exception=(error, catch_backtrace())
        end
    end
end

function distance_col_s(x::SSSet, ε::Real, t_min::Int)
    for i in t_min:length(x)
        @inbounds if evaluate(Euclidean(), x[1], x[i]) ≤ ε
            return (b = true, r = i)
        end
    end
    return (b = false, )
end

function find_element_with_i(psos_res, I)
    for (outer_index, inner_array) in enumerate(psos_res)
        for (inner_index, element) in enumerate(inner_array)
            if element.i == I
                return (outer_index, inner_index)
            end
        end
    end
    return nothing
end

function group_by_key(arr, keyfunc)
    groups = Dict{Any, Vector{Any}}()
    for el in arr
        k = keyfunc(el)
        push!(get!(groups, k, Vector{Any}()), el)
    end
    return collect(values(groups))
end

function group_psos(psos_res)

```

```

groups_len = group_by_key(psos_res, x -> length(x.psos))

# result = Vector{Vector{Any}}()

# for group_len in groups_len
#     groups_dir = group_by_key(group_len, x -> x.direction)

#         for group_dir in groups_dir
#             groups_v = group_by_key(group_dir, x -> x.v)

#                 append!(result, groups_v)
#             end
#         end

# return groups_len #result
end

function min_index(data::SSSet, dir)
    mi = Vector(data[1])

    res = [1, 1, 1]

    for i in eachindex(data)
        if data[i][dir] < mi[dir]
            mi[dir] = data[i][dir]
            res[dir] = i
        end
    end

    return res[dir]
end

function compare_psoses(psoses, δ)
    differ = []

    len_psos = length(psoses[1])
    n = length(psoses)

    for i in 1:len_psos
        for j in 1:n
            for k in j + 1:n
                if evaluate(Euclidean(), psoses[j][i], psoses[k][i]) >= δ
                    push!(differ, (j, k))
                end
            end
        end
    end
end

```

```

unique!(differ)

    return find_optimal_groups(n, differ)
end

function find_optimal_groups(n, differ)
    forbidden = Set((min(u,v), max(u,v)) for (u,v) in differ)

    g = SimpleGraph(n)
    for u in 1:n
        for v in u+1:n
            if !( (u,v) ∈ forbidden )
                add_edge!(g, u, v)
            end
        end
    end
end

components = connected_components(g)

result = []
for comp in components
    group = tuple(comp[1], comp[2:end]...)
    push!(result, group)
end

return result
end

function create_next_file(dir_path, file_prefix, file_extension)
    files = readdir(dir_path)

    filtered_files = filter(file -> startswith(file, file_prefix) && ends with(file, file_extension))

    numbers = Int[]
    for file in filtered_files
        suffix = file[length(file_prefix)+1:end-length(file_extension)]
        try
            num = parse(Int, suffix)
            push!(numbers, num)
        catch end
    end

    max_number = isempty(numbers) ? 0 : maximum(numbers)

    new_file_number = max_number + 1
    new_file_name = "$(file_prefix)$(new_file_number)$(file_extension)"

```

```

new_file_path = joinpath(dir_path, new_file_name)

touch(new_file_path)

return new_file_path
end

function to_line(tuple)
    field_names = fieldnames(typeof(tuple))

    line_parts = String[]
    for name in field_names
        value = getproperty(tuple, name)
        push!(line_parts, "$(name) = $(value)")
    end

    line = "(" * join(line_parts, ", ") * ")"
    return line
end

function initFourthPage(screen, fig, system, final_psos, i_orbits, ds, u0s)
    current_size = Makie.size(fig.scene)
    figure = Figure(size = current_size)
    display(screen, figure)

    psos = [if i_orbits[] [i] == true
            final_psos[] .psos[i]
            else
            nothing
            end for i in eachindex(i_orbits[])]

    u0s = [if i_orbits[] [i] == true
           u0s[i]
           else
           nothing
           end for i in eachindex(i_orbits[])]
    filter!(isnothing, psos)
    filter!(isnothing, u0s)

    v      = final_psos[] .v
    j      = final_psos[] .j
    dir   = final_psos[] .d

    saveR = Observable(false)
    save0 = Observable(false)
    t_max = Observable(30.0)

```

```

t_min = Observable(1.0)
ε      = Observable(0.01)
δ      = Observable(0.05)
Δt    = 0.01

ds == "Chua" && (ε[] == 0.005)

label  = Label(figure[1, 1], L"\text{Enter}\\", \epsilon, \, \text{maximum a")
label1 = Label(figure[2, 1], L"t_{max}=")
tb1    = Textbox(figure[2, 2], placeholder = string(t_max[]), validator = Float64)

# TODO: make denominator for tb2 (0.01 <= t_min)
label2 = Label(figure[3, 1], L"t_{min}=")
tb2    = Textbox(figure[3, 2], placeholder = string(t_min[]), validator = Float64)

# TODO: make denominator for tb3 (0.05 <= ε <= 1)
label3 = Label(figure[4, 1], L"\epsilon=")
tb3    = Textbox(figure[4, 2], placeholder = string(ε[]), validator = Float64)

# TODO: make denominator for tb4 (10e-10 <= δ <= 10e-3)
label4 = Label(figure[5, 1], L"\delta=")
tb4    = Textbox(figure[5, 2], placeholder = string(δ[]), validator = Float64)

label5 = Label(figure[6, 1], "Save orbits picture? ")
cb1    = Checkbox(figure[6, 2], checked = false)

label6 = Label(figure[7, 1], "Save orbits animation? ")
cb2    = Checkbox(figure[7, 2], checked = false)

on(tb1.stored_string) do t t_max[] = parse(Float64, t) end
on(tb2.stored_string) do t t_min[] = parse(Float64, t) end
on(tb3.stored_string) do ε ε[]      = parse(Float64, ε) end
on(tb4.stored_string) do δ δ[]      = parse(Float64, δ) end
on(cb1.checked)        do s saveR[] = s end
on(cb2.checked)        do s saveO[] = s end

create_button_next(figure, callback, system[], psos, v, j, dir, u0s, t_
end

```

#### 7.4.6 Исходный код файла poincare.jl

```

using DynamicalSystems, OrdinaryDiffEq
using GLMakie, WGLMakie, Makie.Colors

```

```

using GridLayoutBase

include("util.jl")

function convert_to_svector(points::Vector{Point3f})
    return [SVector(Float64(p[1]), Float64(p[2]), Float64(p[3])) for p in points]
end

function scan_poincaresos(
    As::Vector{Tuple{S, T}} where {S<:AbstractStateSpaceSet, T<:StepRange}
    args...; kwargs...
)
    return scan_poincaresos([As[i][1] for i in eachindex(As)], args...; kwargs...)
end

# Correct dir when final_psos shown and change it
function scan_poincaresos(
    As::Vector{<:AbstractStateSpaceSet},
    screen, current_size, best_psos, final_psos, i_orbits, total_time, pa,
    linekw = (), scatterkw = (), j = 2,
    colors = [:red, :orange, :green, :yellow, :pink, :brown,
              :cyan, :magenta, :purple, :coral, :gold, :teal,
              :salmon, :lavender, :gray, :violet, :silver]
)
    size_ = size(As, 1)
    @assert size(colors, 1) > size_

    each_i = eachindex(As)
    was_changed = false
    orbit_number = nothing
    button_change_psos = nothing

    mi, ma, minmax = total_minmaxima(As)

    otheridxs = [setdiff(1:3, j)...]

    figure = Figure(size = current_size)

    display(screen, figure)

    ax = Axis3(figure[1, 1])
    axp = Axis(figure[1, 2])

    GLMakie.limits!(ax, mi[1], ma[1],
                    mi[2], ma[2],
                    mi[3], ma[3])
    GLMakie.limits!(axp, mi[otheridxs[1]], ma[otheridxs[1]],
                    mi[otheridxs[2]], ma[otheridxs[2]])

```

```

                mi[otheridxs[2]], ma[otheridxs[2]])

sg = SliderGrid(figure[2, :], 
    label      = "$(('x':'z')[j]) =",
    range      = range(mi[j], ma[j], length = total_time),
    startvalue = (ma[j] + mi[j]) / 2)
)
slider = sg.sliders[1]

if size_ == 1
    lines!(ax, As[1].data; color = RGBA{Float32}(0.0, 0.44705883, 0.6)
else
    line = []
@inbounds for i in each_i
    push!(line, lines!(ax, As[i].data; color = colors[i], transparency = 0.6))
end
end

direction          = Observable(-1)
scatter_plots_ax  = Observable(Point3f[])
scatter_plots_axp = Observable(Point2f[])
point_color        = Observable(Symbol[])
curr_plane         = Observable{Any}()

final_psos[] = (psos = [best_psos[k][j].s for k in eachindex(best_psos)]
for j in each_j)

axp_scatter = Makie.scatter!(axp, scatter_plots_axp; color = point_color[])
ax_scatter  = Makie.scatter!(ax, scatter_plots_ax; color = point_color[])

orbits_range = each_i

# Main animation
on(slider.value) do j_val
    all_psos3d = Point3f[]
    all_psos2d = Point2f[]
    all_colors = Symbol[]

    curr_plane[] = (j, j_val)

    @inbounds for i in orbits_range
        psos = nothing

        try
            psos = poincaresos(As[i], curr_plane[]; direction = direction[])
        catch err
            continue
        end
    end
end

```

```

        psos2d = [p[otheridxs] for p in psos]
        append!(all_psos2d, psos2d)

        psos3d = [p for p in psos]
        append!(all_psos3d, psos3d)

@inbounds for _ in psos
    push!(all_colors, colors[i])
end
end

scatter_plots_axp[] = all_psos2d
scatter_plots_ax[] = all_psos3d
point_color[] = all_colors

if was_changed
    axp_scatter = Makie.scatter!(axp, scatter_plots_axp; color =
        ax_scatter = Makie.scatter!(ax, scatter_plots_ax;
color = point_color, scatterkw..., markersize = 5)
    was_changed = false
end
end

# Surface animation
ss = copy(mi)
ws = copy(ma) .- copy(mi)
ws[j] = 0

p = map(slider.value) do val
    ss[j] = val
    o = Makie.Point3f(ss...)
    w = Makie.Point3f(ws...)
    Makie.Rect3f(o, w)
end

mesh!(ax, p; color = RGBAf(0.2, 0.2, 0.25, 0.5), transparency = true)

# Buttons
buttons_grid = GridLayout(figure[3, :])

if size_ != 1
    # Button to reset everything
    button_reset = Button(buttons_grid[1, 1], label = "Reset all")
    on(button_reset.clicks) do _
        GLMakie.limits!(ax, mi[1], ma[1],
                        mi[2], ma[2],

```

```

                mi[3], ma[3])
GLMakie.limits!(axp, mi[otheridxs[1]], ma[otheridxs[1]],
                mi[otheridxs[2]], ma[otheridxs[2]])

for k in eachindex(line)
    delete!(screen, figure.scene, line[k])
end
empty!(line)

delete!(screen, figure.scene, ax_scatter)
delete!(screen, figure.scene, axp_scatter)

@inbounds for i in each_i
    push!(line, lines!(ax, As[i].data; color = colors[i], tra
end

was_changed == true && (was_changed = false)

slider.range = range(mi[j], ma[j]; length = total_time)
set_close_to!(slider, (mi[j] + ma[j]) / 2)

was_changed = true

scatter_plots_ax = Observable(Point3f[])
scatter_plots_axp = Observable(Point2f[])
point_color       = Observable(Symbol[])

orbits_range = each_i
orbit_number = nothing

ss = copy(mi)
ws = copy(ma) .- copy(mi)
ws[j] = 0
end

# Buttons to change orbits
vertical_grids = [vbox!(Button(figure, label = "Change to $i"),
                        Checkbox(figure, checked = true))
                  for i in each_i]

@inbounds for i in each_i
    buttons_grid[1, 1 + i] = vertical_grids[i]
    on(vertical_grids[i].content[1].content.clicks) do _
        GLMakie.limits!(ax, minmax[i][1][1], minmax[i][2][1], mi
                        minmax[i][2][2], minmax[i][1][3], mi
        GLMakie.limits!(axp, minmax[i][1][otheridxs[1]], minmax[i
                        minmax[i][1][otheridxs[2]], minmax[i

```

```

        for k in eachindex(line)
            delete!(screen, figure.scene, line[k])
        end
        empty!(line)

        delete!(screen, figure.scene, ax_scatter)
        delete!(screen, figure.scene, axp_scatter)

        push!(line, lines!(ax, As[i].data; color = RGBA{Float32}(
            ss = copy(minmax[i][1])
            ws = copy(minmax[i][2]) .- copy(minmax[i][1])
            ws[j] = 0

            was_changed == true && (was_changed = false)

            slider.range = range(minmax[i][1][j], minmax[i][2][j]; length = 100)
            set_close_to!(slider, (minmax[i][1][j] + minmax[i][2][j]) / 2)
            was_changed = true

            scatter_plots_ax = Observable(Point3f[])
            scatter_plots_axp = Observable(Point2f[])
            point_color = Observable(Symbol[])

            orbits_range = range(i, i)
            orbit_number = i
        end

        # Checkboxes
        on(vertical_grids[i].content[2].content.checked) do b
            i_orbits[] [i] = b
        end
    end

    # Button to change final psos
    button_change_psos = Button(buttons_grid[1, size_ + 2], label = "Change")
    on(button_change_psos.clicks) do _
        if length(scatter_plots_ax[]) != 0
            psos_tmp = []
            for i in eachindex(As)
                try
                    push!(psos_tmp, poincaresos(As[i], curr_plane[]))
                catch err end
            end
            final_psos[] = (psos = psos_tmp, v = slider.value[], j = i)
        end
    end
end

```

```

        end
    end

    # Button to show final psos
    button_change_psos = Button(buttons_grid[1, size_ + 3], label = "")
    on(button_change_psos.clicks) do _
        j != final_psos[].j && (button_change_hyperplane.clicks[] = b
        j != final_psos[].j && (button_change_hyperplane.clicks[] = b
        set_close_to!(slider, final_psos[].v)
    end
else
    # Button to change final psos
    button_change_psos = Button(buttons_grid[1, 1], label = "Change f
    on(button_change_psos.clicks) do _
        length(scatter_plots_ax[]) != 0 &&
            (final_psos[] = (psos = scatter_plots_ax[], v = slider.va
    end

    # Button to show final psos
    button_change_psos = Button(buttons_grid[1, 2], label = "Show fin
    on(button_change_psos.clicks) do _
        j != final_psos[].j && (button_change_hyperplane.clicks[] = b
        j != final_psos[].j && (button_change_hyperplane.clicks[] = b
        set_close_to!(slider, final_psos[].v)
    end
end

# Button to change hyperplane direction
button_change_hyperplane = Button(buttons_grid[1, size_ + 4], label = "")
on(button_change_hyperplane.clicks) do _
    j = (j + 1 == 4) ? 1 : j + 1
    otheridxs = [setdiff(1:3, j)...]
    if isnothing(orbit_number)
        GLMakie.limits!(axp, mi[otheridxs[1]], ma[otheridxs[1]], mi[otheridxs[2]], ma[otheridxs[2]])
    end
    ss = copy(mi)
    ws = copy(ma) .- copy(mi)
    ws[j] = 0

    delete!(screen, figure.scene, ax_scatter)
    delete!(screen, figure.scene, axp_scatter)

    was_changed == true && (was_changed = false)

```

```

        slider.range = range(mi[j], ma[j]; length = total_time)
        set_close_to!(slider, (mi[j] + ma[j]) / 2)

        scatter_plots_ax = Observable(Point3f[])
        scatter_plots_axp = Observable(Point2f[])
        point_color       = Observable(Symbol[])

        was_changed = true
    else
        ss = copy(minmax[orbit_number][1])
        ws = copy(minmax[orbit_number][2]) .- copy(minmax[orbit_number][1])
        ws[j] = 0

        GLMakie.limits!(axp, minmax[orbit_number][1][otheridxs[1]],
                        minmax[orbit_number][2][otheridxs[1]],
                        minmax[orbit_number][1][otheridxs[2]],
                        minmax[orbit_number][2][otheridxs[2]])

        delete!(screen, figure.scene, ax_scatter)
        delete!(screen, figure.scene, axp_scatter)

        was_changed == true && (was_changed = false)

        slider.range = range(minmax[orbit_number][1][j], minmax[orbit_number][2][j])
        set_close_to!(slider, (minmax[orbit_number][1][j] + minmax[orbit_number][2][j]) / 2)

        scatter_plots_ax = Observable(Point3f[])
        scatter_plots_axp = Observable(Point2f[])
        point_color       = Observable(Symbol[])

        was_changed = true
    end
end

# Button to change psos direction
button_change_direction = Button(buttons_grid[1, size_ + 5], label =
on(button_change_direction.clicks) do _
    if direction[] == 1
        direction[] = -1
    elseif direction[] == -1
        direction[] = 1
    end

    if length(scatter_plots_ax[]) != 0
        tmp = slider.value[]
        slider.value[] = 0
        set_close_to!(slider, tmp)
    end
end)
```

```

        end
    end

    # Button to go to the next page
    button_next = Button(buttons_grid[1, size_ + 6], label = "Next")
    on(button_next.clicks) do _
        delete_all!(figure)
        pageClosedT[] = true
    end

    return best_psos
end

function Base.filter(f, t::StateSpaceSet)
    filtered_data = [point for point in t if f(point)]
    return StateSpaceSet(filtered_data)
end

function total_minmaxima(As::Vector{<:AbstractStateSpaceSet})
    minmax = Vector{()}()
    mi, ma = DynamicalSystems.minmaxima(As[1])

    mi = Vector(mi); ma = Vector(ma)

    push!(minmax, [mi, ma])

    @inbounds for j in 2:length(As)
        mi2, ma2 = DynamicalSystems.minmaxima(As[j])
        mi2 = Vector(mi2)
        ma2 = Vector(ma2)

        mi = min.(mi, mi2)
        ma = max.(ma, ma2)

        push!(minmax, [mi2, ma2])
    end

    return mi, ma, minmax
end

function find_best_poincaresos(As::Vector{Tuple{S, T}}) where {S<:Abstract}
    return find_best_poincaresos([As[i][1] for i in eachindex(As)], D, to
end

function find_best_poincaresos(As::Vector{<:AbstractStateSpaceSet}, D::In

```

```

best_psos = []
for i in eachindex(As)
    psos_j = Vector{Any}(undef, D)

    min, max = minmaxima(As[i])

    for j in 1:D
        psos_j[j] = (s = AbstractStateSpaceSet{D, Float64}[], v = Flo
            for j_val in range(min[j], max[j], length = total_time)
                section = nothing

                try
                    section = poincaresos(As[i], (j, j_val)); direction =
                catch err
                    continue
                end

                if isempty(psos_j[j].s) || length(section) > length(psos_
                    psos_j[j] = (s = section, v = j_val)
                end
            end
        end
        push!(best_psos, psos_j)
    end

    return best_psos
end

function filter_orbits(trs::Vector{Tuple{S, T}}) where {S<:AbstractStateSpa
    return filter_orbits([trs[i][1] for i in eachindex(trs)])
end

function filter_orbits(trs::Vector{<:AbstractStateSpaceSet})
    filtered_trs = Vector{StateSpaceSet{}}()
    for tr in trs
        invalid_index = findfirst(tr) do state
            !(all(-10e6 .< state .< 10e6) && all(!isinf, state) && all(!i
        end

        if invalid_index !== nothing
            valid_points = tr[1:invalid_index-1]
        else
            valid_points = tr
        end

        push!(filtered_trs, valid_points)
    end
end

```

```

    end
    return filtered_trs
end

```

#### 7.4.7 Исходный код файла LM.jl

```

#using Pkg; Pkg.add(url = "https://github.com/JuliaDynamics/PeriodicOrbits")
using PeriodicOrbits
using NonlinearSolve
using SciMLBase

"""
This function is almost full copy of PeriodicOrbits.periodic_orbit()
Reason why it's here is because the original one doesn't support Δt changing
while creating PO trajectory.
"""

function periodic_orbit_(ds::CoupledODEs, alg::OptimizedShooting, ig::Init)
    D = dimension(ds)

    f = (err, v, p) -> begin
        if isinplace(ds)
            u0 = @view v[1:D]
        else
            u0 = SVector{D}(v[1:D])
        end
        T = v[end]

        bounds = zeros(eltype(v), alg.n * 2)
        for i in 0:alg.n-1
            bounds[i + 1] = i * alg.Δt
            bounds[i + alg.n + 1] = T + i * alg.Δt
        end
        tspan = (0.0, T + alg.n * alg.Δt)

        sol = solve(SciMLBase.remake(ds.integ.sol.prob; u0 = u0, tspan = tspan,
                                      DynamicalSystemsBase.DEFAULT_DIFFEQ..., ds.diffeq..., saveat = 0))
        if (length(sol.u) == alg.n * 2)
            for i in 1:alg.n
                err[D * i - (D - 1):D * i] = (sol.u[i] - sol.u[i + alg.n])
            end
        else
            fill!(err, Inf)
        end
    end
end

```

```

prob = NonlinearLeastSquaresProblem(
    NonlinearFunction(f, resid_prototype = zeros(alg.n*dimension(ds))), 

sol = solve(prob, NonlinearSolve.LevengbergMarquardt(); alg.nonlinear_so

u0 = sol.u[1:end - 1]
T = sol.u[end]

T < t_min && (return nothing)

return (state = trajectory(ds, T - Δt, u0; Δt = Δt)[1], period = T)
end

function lm(system::CoupledODEs, u0::U where {U<:AbstractArray{<:Real}}, 
    ig = InitialGuess(u0, T)
    po = periodic_orbit_(system, OptimizedShooting(), ig, Δt, t_min)
    return po
end

```

#### 7.4.8 Исходный код файла util.jl

```

using GLMakie

function delete_all!(fig::Figure)
    while !isempty(fig.content)
        for child in fig.content delete!(fig.content[1]) end
        trim!(fig.layout)
    end
end

```