Задача распознавания символического образа динамической системы

Германчук М.С., Лукьяненко В.А., Меньшиков А.О.

Крымский федеральный университет им. В. И. Вернадского, Симферополь, Россия

Всероссийская конференция ММРО-2019. Россия, г. Москва, 26–29 ноября 2019 г.

Символический образ (СО)

Символический образ H – это ориентированный граф G, построенный по покрытию C фазового пространства исследуемой динамической системы и отражающий переходы из одного элемента разбиения другой.
Пусть $C = (C_i, \ldots, C_k)$ – конечное покрытие области M замкнутыми множествами, которые назовем ячейками. Имеется область M, динамика F и построенное покрытие C, тогда можно построить такой граф G, что каждая его вершина i соответствует ячейке C_i, а ребра соответствуют узлам между ячейками, то есть направленное ребро $i \to j$ существует тогда и только тогда, когда $F(C_i) \cap C_j \neq \emptyset$. Этот ориентированный граф и будет называться символическим образом.

Алгоритм построения символического образа

1. Для исходного множества M_0, в котором лежат значения дискретной динамической системы, строим ячеечное покрытие C. Ячейки имеют форму квадрата с заранее заданным размером ребра d_0 (в общем случае ячейки могут иметь произвольную форму и произвольный размер).
2. Для покрытия C строим граф G, являющийся символическим образом.
3. Используя один из алгоритмов поиска сильно связных вершин (алгоритмы Тарьяна [7], Седжвика [8], поиска сильно связных вершин на основе поиска путей [9]), выделяем сильно связные вершины $\{i_k\}$. Если $\{i_k\} = \emptyset$, то в области M_0 нет аттрактора и локализуемое цепно-рекуррентное множество является пустым и процесс его локализации прекращается [6]. Иначе, удаляем из графа невозвратные вершины и от области M_0 переходим к новой области M_1, такой что $M_1 = \{x \in M_0^{i_k} : i_k \in G\}$.
4. Строим для нового множества M_1 покрытие C_1, так что размер ребра новой ячейки $d_1 = \frac{d_0}{2}$. Переходим к пункту 2 в том случае, если $d_1 > \varepsilon$, где ε – заранее заданный предельный размер ячейки.

Программная реализация СО

Задача программной реализации алгоритма построения символического образа основывается на теории предложенной Г. С. Осипенко. Для реализации данного проекта был выбран язык программирования Python [10]. Для отображения графики выбрана графическая среда OpenGL (библиотека GLUT).
Использованы:
- библиотека NetworkX [12], предназначенная для создания, работы и изучения структур, динамики и функций сложных сетей, а следовательно направленных графов;
- библиотека NumPy [13], предназначенная для работы с двумерными массивами.
- библиотека SciPy [14], которая позволяет решать как простые дифференциальные системы, так и системы дифференциальных уравнений с параметрами.
Среда разработки – Anaconda [15].
Все тесты проведены на MacBook Pro 2015 с процессором Intel Core i5 2,7 GHz и памятью 8 ГБ 1867 MHz DDR3, операционная система macOs High Sierra.

Реализация алгоритма построения СО

Создан класс osipGraph и методы __init__, cellDiv, fillNodes, giveDir, giveDir_ linear, deleteComponents, findPlace, findPlace_linear. Последовательность применения методов:

1. Инициализируется граф, задаются границы, размеры ячеек разбиения.
2. Применяется метод fillNodes, в котором каждой вершине графа сопоставляются ячейки покрытия.
3. Выполняется giveDir или giveDir_linear, в зависимости от выбранного метода отображения. Ячейки отображаются поточечно или линейно, к каждой вершине добавляются направленные ребра.
4. Завершается первый этап алгоритма методом deleteComponents. Алгоритм Тарьяна удаляет все несильно связные вершины, оставляя только те, соответствующие ячейки которых будут в последствии делится на четыре новых.
5. Метод cellDiv для оставшихся вершин делит соответствующие им ячейки на новые. Алгоритм переходит к третьему шагу, если размер ячейки позволяет разделить снова.

В работе проведен рефакторинг данного метода с целью добавить возможность использовать не квадратную область для исследования. Также был изменен метод определения номера узла. Теперь, область разбивается на ячейки, каждой из которых дается номер строки и номер столбца, где отсчет строк идет сверху вниз, а отсчет столбцов слева направо. Затем, используя формулу $nodeNumber = col + row * self.lengthOfSide[0]$ вычисляется номер узла. Это нужно для того, чтобы в дальнейшем поиск номера ячейки, в которое попадает отображение точки, вычислялся всего за три действия: вычисление номера строки, столбца и далее самого номера. Это значительно ускоряет работу программы.

Для точечного метода, который был незначительно переделан, выбираем ячейки начиная с нулевой, и отображаем координаты определенного количества равномерно заданных точек, принадлежащих ячейке. От количества данных точек будет сильно зависеть время и точность работы. По результатам проведенных подсчетов, оптимальное количество – девять точек. После отображения ищется в какую ячейку попали отображенные точки с помощью функции findPlace().

Для линейного метода последовательность действий:

1. Отображаются верхняя левая и нижняя правая координаты ячейки.
2. С помощью метода findPlace_linear находятся ячейки, в которые попадают отображения.
3. Строятся ребра во все ячейки, лежащие в прямоугольнике, покрывающим найденные ячейки.

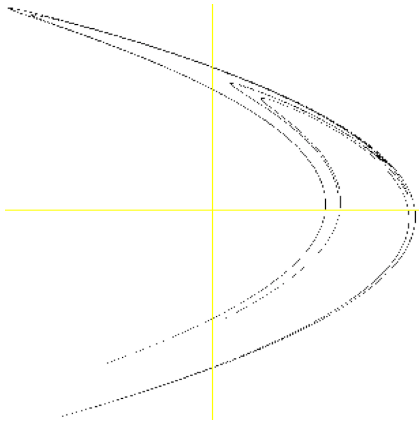
Завершает первый этап метод deleteComponents().
В начале добавляются все ячейки, которые отображаются сами в себя в отдельный список. Затем с помощью метода strongly_connected_components из библиотеки NetworkX находятся все сложные компоненты. Наконец, удаляются все вершины, не являющиеся сложными и не входящие в список тех, что отображаются в себя. Теперь граф полностью готов к делению.
Деление ячеек на четыре новых реализовано в методе cellDiv. Для каждой ячейки, что вошла в граф после удаления сложных компонент, в новом инициализованном графе строится четыре ячейки по такому принципу: первая ячейка имеет те же координаты вершины, что и изначальная, вторая ячейка справа от первой, третья снизу и четвертая в нижнем правом углу.
После получения нового графа, можно передавать его значения старому для сохранения в памяти и проделать те же операции снова.
Что же касается прорисовки графа, то она тривиальна и реализована в функции drawgraph.

Анализ реализации алгоритмов построения СО

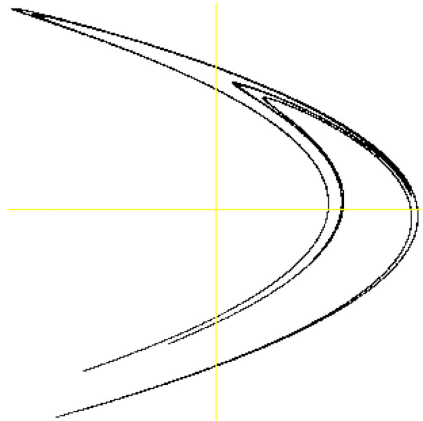
1. Отображение Хенона [17] – это дискретная динамическая система с параметрами a, b, в зависимости от которых будут получаться различные цепно-рекуррентные множества:

$$\begin{cases} x_{n+1} = 1 - ax_n^2 + by_n \\ y_{n+1} = x_n \end{cases},$$

Параметры $a = 1.4$, $b = 0.3$, при которых отображение называется классическим и известно, что оно имеет цепно-рекуррентное множество в области $[-1.5, 1.5] \times [-\frac{10}{3}, \frac{-10}{3}]$.



Точечный метод



Линейный метод

Рис.: Отображение Хенона

2. Отображение Заславского [18] имеет вид:

$$\begin{cases} x_{n+1} = x_n + \nu(1 + \mu y_n) + \epsilon\nu\mu cos(2\pi x_n)) mod 1 \\ y_{n+1} = e^{-r}(y_n + \epsilon cos(2\pi x_n)) \end{cases},$$

Выбраны параметры: $\epsilon = 5, \nu = 0.2, r = 2, \mu = 1 - \frac{e^{-r}}{r}$. Область поиска: $[-1, 1][-1, 1]$.

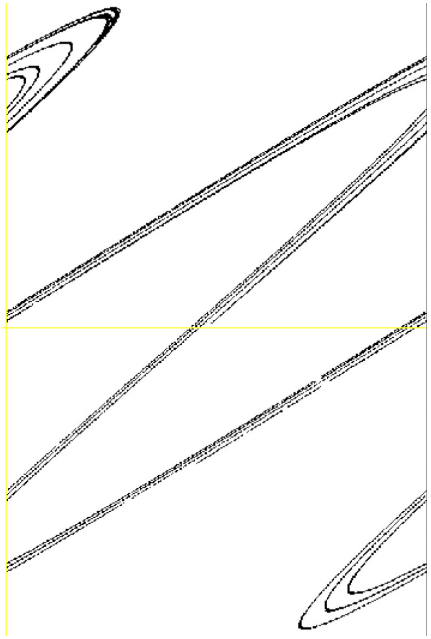


Рис.: Отображение Заславского

3. Аттрактор Питера де Йона [19] задается:

$$\begin{cases} x_{n+1} = sin(ay_n) - cos(bx_n) \\ y_{n+1} = sin(cx_n) - cos(dy_n) \end{cases},$$

Выбраны параметры $a = -2.7, b = -0.09, c = -0.86, d = -2.2$.

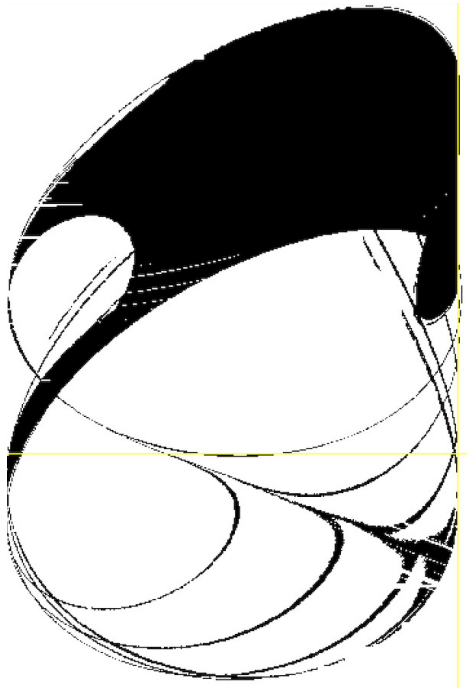


Рис.: Аттрактор Питера де Йона

Заключение

В работе систематизированы теоретические и прикладные результаты решения задачи построения символического образа динамической системы и выделения цепно-рекуррентных множеств, выбраны и модернизированы алгоритмы, пригодные для реализации задачи, реализованы алгоритмы построения траекторий динамической системы с использованием точечного метода и линейного метода реализована визуализаци графического отображения данных траекторий.
Точечный алгоритм оказался достаточно быстрым и точным для построения символических образов. Возможность изменять количество точек позволяет контролировать скорость и точность работы программы.
Выбранная среда разработки Anaconda и язык Python, оправдали ожидания и обеспечили приемлемую скорость, точность и стабильность. В перспективе предполагается реализация других методов построения символического образа, основанных на комбинации адаптивных и линейных алгоритмов.