Ампилова Наталья Борисовна,
Терентьев Сергей Валерьевич

УДК 517.9

РЕАЛИЗАЦИЯ ИНТЕРВАЛЬНОГО АЛГОРИТМА ЛОКАЛИЗАЦИИ ИНВАРИАНТНЫХ МНОЖЕСТВ ДИНАМИЧЕСКИХ СИСТЕМ

Аннотация

Работа посвящена разработке и реализации основанного на интервальной арифметике алгоритма локализации инвариантных множеств динамических систем. Используется метод аппроксимации системы с помощью символического образа, представляющего собой ориентированный граф, построенный по системе. Ячейки разбиения рассматриваются как интервальные вектора в пространстве соответствующей размерности. Оценка параметров символического образа позволяет определить точность построения. Приведены результаты численных экспериментов и сравнение с алгоритмами локализации, основанными на обычной арифметике.

Ключевые слова: динамические системы, инвариантные множества, символический образ, компьютерное моделирование, интервальная арифметика.

ВВЕДЕНИЕ

Теория динамических систем является фундаментальной математической дисциплиной, тесно связанной с большинством областей математики. Динамические системы все чаще используются во многих областях как модели реальных процессов. Они делятся на два основных класса: непрерывные и дискретные. Непрерывные динамические системы задаются автономными системами дифференциальных уравнений, а дискретные – рекуррентными соотношениями (разностными уравнениями).

Гладкие динамические системы с непрерывным временем являются основными моделями в классической механике со времен Ньютона. В XX веке они вновь привлекли к себе внимание благодаря работе А. Пуанкаре, который при решении проблемы трех тел обнаружил, что в ней возникают сложные режимы, появление которых невозможно объяснить, опираясь на принципы классической механики, а аналитическое решение получить не удается. Такие режимы характеризуются существованием апериодических траекторий, при этом решения с близкими начальными данными оказываются существенно различными (чувствительная зависимость решений от начальных данных). А. Пуанкаре разработал геометрический подход, который положил начало современной динамике, изучающей долгосрочное асимптотическое поведение системы при помощи методов, не основанных на знании ее решений в явном виде [5].

В 1963 году Е. Лоренц рассмотрел гидродинамическую модель для предсказания погоды. Рассмотренная им система дифференциальных уравнений 3 порядка ведет себя непредсказуемо (хаотично) в том смысле, что малейшие ошибки в измерении состояния атмосферы быстро растут, что делает долгосрочный прогноз погоды неправильным. Системы с хаотическим поведением траекторий были обнаружены во многих прикладных задачах, в динамике популяций и в хорошо известных уравнениях Дуффинга.

Развитие компьютерной техники привело к активному использованию компьютерного моделирования при изучении динамики систем со сложным поведени-

ем траекторий, в частности, ее инвариантных множеств. Один из основных классов компьютерных методов исследования динамических систем представляют так называемые методы, основанные на множествах (set-oriented methods), которые используют конечное разбиение фазового пространства на клетки (ячейки) и отслеживают динамику системы по попаданию точек траекторий исследуемой системы в эти клетки. Для выбранной области фазового пространства K строятся последовательные подразбиения, из которых удаляются те ячейки, образы которых заведомо не принадлежат K. При стремлении диаметра ячеек к нулю мы можем получать все более точный фазовый портрет. На этих методах основано большинство алгоритмов локализации разных видов инвариантных множеств и, в частности, аттракторов динамических систем ([1, 10, 12]).

В 1983 г. Г. Осипенко [2] предложил метод символического образа, который являлся естественным развитием описанной выше идеи и позволил применить к исследованию динамических систем методы теории графов. Символический образ есть конечная аппроксимация динамической системы, представляющая собой ориентированный граф. Он строится по заданному покрытию фазового пространства ячейками C_i, вершины графа соответствуют ячейкам, дуги – связям между ними, а именно: вершины i и j соединяются дугой, если образ ячейки C_i при действии динамической системы пересекается с ячейкой C_j. В работах [2, 3] приводятся доказательства сходимости метода при решении различных задач, например, при построении инвариантных, в частности, цепно-рекуррентных множеств динамических систем.

Наряду с развитием методов исследования динамических систем, основанных на обычной арифметике, все чаще появляются работы, использующие арифметику интервальную, основанную на проведении вычислений с получением точных границ для искомого значения, то есть получения в качестве ответа интервала, содержащего это значение.

Компьютерные методы решения различных задач с помощью интервальной арифметики приведены в работах [7, 6, 8, 17]. Следует отметить работу У. Такера [18], в которой было доказано, что для классических значений параметров в системе Лоренца имеется аттрактор. У. Такер разработал алгоритм, основанный на использовании интервальной арифметики с направленным округлением, позволяющий находить точные решения большого класса обыкновенных дифференциальных уравнений. Подход Такера при исследовании системы Лоренца был основан на комбинации аналитических и компьютерных методов: строгих вычислений и теории нормальных форм. Авторы работы [16] успешно применяют методы интервальной арифметики для компьютерного доказательства существования хаоса в системе Лоренца. Статья [13] посвящена подробному исследованию отображения Икеда, оценке верхней границы инвариантного множества, локализации неблуждающего множества и периодических орбит небольшого периода. Последнее выполняется с использованием интервального оператора Кравчука. Следует отметить, что в работах, посвященнх данной тематике, методы интервальной арифметики используются в основном как доказательные вычисления.

В [9] была решена задача создания программного комплекса для компьютерного исследования динамических систем методами интервальной арифметики и символического образа. Ячейка покрытия естественным образом может быть представлена интервальным вектором, а интервальное расширение функций, описывающих систему, можно рассматривать как «отображение символического образа». При этом сам граф символического образа не строится, связь ячейка–образ(-то есть дуга на графе) хранится в специальной структуре данных. При последовательном подразбиении покрытия вычисляются параметры символического образа, позволяющие оценить точность приближения.В данной работе мы описываем различные реализации алгоритмов

локализации инвариантных множеств динамических систем. Основное внимание уделено оптимизациям в представлении данных и процессе вычислений. Приведены результаты численных экспериментов, убедительно показывающие преимущества данного алгоритма по сравнению с известными алгоритмами, основанными на обычной арифметике.

1. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Символический образ динамической системы строится по заданному покрытию фазового пространства и системе и представляет собой ориентированный граф, вершины которого соответствуют ячейкам покрытия.

Для динамической системы f, покрытия $\{M_i\}$, $i = 1, ..., n$ на символическом образе G строится дуга (i, j) если $f(M_i) \cap f(M_j) \neq \varnothing$.

Для непрерывной системы символический образ строится для отображения сдвига.

Символический образ характеризуется следующими параметрами:

– максимальный диаметр ячейки – d,

– нижняя грань r – наименьшее из расстояний между образами ячеек и теми ячейками, с которыми образы не пересекаются,

– максимальный из диаметров образов ячеек – q.

Определение 1. *Интервалом называется множество вида:*

$$x = [\underline{x}, \overline{x}] = \{\tilde{x} \in R \mid \underline{x} \le \tilde{x} \le \overline{x}\}.$$

Радиус интервала x определяется следующим образом:

$$rad(x) = \frac{\overline{x} - x}{2}.$$

Пространство всех интервалов над R обозначается IR.

Определение 2. *Пусть* $x_1, ..., x_m \in IR$. *Тогда* $\breve{x} = (x_1, ..., x_m)$ *называется интервальным вектором. Пространство интервальных векторов размерности m обозначается* IR^m.

Радиус интервального вектора определяется как

$$rad(\breve{x}) = (rad(x_1), ..., rad(x_m)),$$
$$x_1, ..., x_m \in IR.$$

Нетрудно заметить, что интервальный вектор имеет довольно естественное геометрическое представление: прямое произведение составляющих его интервалов.

Определение 3. *Пусть* $a, b \subset IR$. *Тогда функция*

$$\rho(a,b) = \max\{|\underline{a} - \underline{b}|, |\overline{a} - \overline{b}|\}$$

задает расстояние между интервалами a и b.

Так введенное расстояние является метрикой на IR. Для $a, b \in IR^m$ определим $\rho(a,b) = \max_{i \in [1,m]} \rho(a_i, b_i)$, где (a_i, b_i) – компоненты соответствующих интервальных векторов.

Пусть $f : R^m \to R^m$, $\breve{a} \in IR^m$. Определим область значений f на $\breve{a}$:

$$V(f, \breve{a}) = \{f(x), x \in \breve{a}\}.$$

Определение 4. [17] *Интервальная (то есть вычисленная по правилам интервальной арифметики) функция F называется интервальным расширением f, если для любого интервального вектора* $\breve{x}$ *выполняется включение* $V(f, \breve{x}) \subseteq F(\breve{x})$.

Поскольку для вещественнозначной функции f существует много интервальных расширений ([15]), удобно выбрать то, которое получается, если вычислять f по правилам интервальной арифметики (естественное расширение). Обозначим такое расширение $f(\breve{a})$. По известному свойству монотонности по включению $V(f, \breve{a}) \subseteq f(\breve{a})$. Разобьем $\breve{a}$ на более мелкие части $\breve{a}^1, ..., \breve{a}^n$. Тогда из указанного свойства следует

$$V(f, \breve{a}) \subseteq \bigcup_{i=0}^{n} f(\breve{a}^i) \subseteq f(\breve{a}).$$

Поскольку при стремлении диаметра подразбиения к нулю $\rho(V(f, \breve{a}), f(\breve{a}))$ оценивается через диаметр $\breve{a}$ (как множества), а для дифференцируемой функции f через квадрат диаметра [18], то при последовательном подразбиении интервальных векторов мы получаем оценку для $V(f, \breve{a})$ с любой заданной точностью.

Таким образом, интерпретируя ячейки покрытия как интервальные вектора в пространстве соответствующей размерности, можно рассматривать интервальное расширение исходной функции f как «отображение символического образа» и строить образ ячейки, используя правила и функции интервальной арифметики.

2. ОПИСАНИЕ АЛГОРИТМА

Пусть $\{M_i\}$ есть покрытие фазового пространства M динамической системы f, где $i = 1, ..., n$. Обозначим за $\breve{M}_i$ интервальный вектор, соответствующий ячейке M_i, тогда $\breve{M}_i = a_1^i \times ... \times a_j^i$, где a_j^i – j-ая компонента интервального вектора. Таким образом, ячейка покрытия является многомерным параллелепипедом. Введем также обозначения

$$C_i = \{\breve{M}_j \,|\, \breve{M}_j \cap f(\breve{M}_i) \neq \varnothing\},$$

$$c_i = \{j \,|\, \breve{M}_j \cap f(\breve{M}_i) \neq \varnothing\} \text{ и } \breve{R}_i = \bigcup_{j \in c_i} \breve{M}_j .$$

Определим диаметр ячейки $diam(\breve{M}_i) = \max_{k \in [1,m]} \{2 * rad(\breve{M}_i)\}_k$ и обозначим через d наибольший из таких диаметров. Пусть $r = \min_{i,k} dist(f(\breve{M}_i), \breve{M}_k)$, где $f(\breve{M}_i) \cap \breve{M}_k = \varnothing$ и $dist$ обозначает хаусдорфово расстояние между множествами. (Отметим, что в данном случае удобнее использовать именно это расстояние, а не интервальное, так как последнее дает большее значение для непересекающихся интервалов.) Таким образом, для оценки параметров символического образа нужно вычислять d и r.

Пусть $D_1 = \{\breve{M}_1, ..., \breve{M}_n\}$ – начальное покрытие компакта $\breve{M}$ ячейками $\breve{M}_i \subseteq IR^m$; $i = 1, ..., n$. Для каждой ячейки $\breve{M}_i$ вычислим множества $\breve{R}_i$. Построим множество-представление графа G_1, соответствующее символическому образу G_1 отображения f относительно покрытия D_1, а именно $G_{D_1} = \bigcup_{i \in [1,n]} \breve{R}_i$. Это множество содержит все ячейки покрытия, которые участвуют в построении графа G_1. Вычисляем значения d_1 и r_1. На следующем шаге разбиение применяется к элементам множества G_{D_1}, в результате повторения описанной процедуры получается множество-представление G_{D_2} для графа G_2. Вычисляем d_2, r_2. Множества локализуются с точностью $\varepsilon > d_2$.

На k-ом шаге мы рассматриваем разбиение вида

$$D_k = \{\breve{M}_{j1}, ..., \breve{M}_{jm} \,|\, \breve{M}_{jl} \subset G_{D_{k-1}}\},$$

строим множество-представление G_{D_k} и вычисляем соответствующие d_k и r_k. Согласно [3], при стремлении диаметра разбиения к нулю мы локализуем инвариантные множества с точностью $\varepsilon > d_k$.

3. ПРЕДСТАВЛЕНИЕ СИСТЕМЫ И ДИНАМИЧЕСКАЯ ГЕНЕРАЦИЯ КОДА

Вычисление арифметических выражений, описывающих систему, может быть оптимизировано следующим образом: программный модуль, создаваемый динамически на основе специального описания системы с помощью специально разработанной библиотеки. Такой подход позволяет избежать многократного обхода дерева разбора при вычислении арифметического выражения [9]. При реализации алгоритма были созданы 2 библиотеки динамической генерации кода:

– для Microsoft .Net платформы [9];

– с поддержкой POSIX стандартов.

При проектировании данных учитывались следующие требования:

1) быстродействие,

2) инкапсуляция данных,

3) поддержка вещественной и интервальной арифметик,

4) простота и гибкость в использовании.

Вычисление арифметических выражений, а также доступ к описанию системы должны осуществляться максимально эффективно. Описание системы должно инкапсулироваться в одном объекте на протяжении всего жизненного цикла программы. Такой подход позволяет достичь не только предельной локализации изменений (при их необходимости), но и прогнозируемости последствий изменений. Опи-

сания системы для двух разных арифметик выглядят одинаково, арифметические выражения вычисляются однообразно. Поэтому для достижения этой тождественности в реализации методы создания класса–системы должны опираться на использование принципов обобщенного программирования, то есть объект–системы можно применять к различным арифметикам, не меняя внутреннюю архитектуру этой структуры данных. При использовании модуля динамической генерации кода необходимо ознакомиться только с форматом входных данных, способом инициализации и с набором исключительных ситуаций, возникающих при некорректной работе с библиотекой, не вникая в подробности внутреннего устройства библиотеки.

Платформа Microsoft .Net. В качестве промежуточного языка используется C# [22]. Благодаря технологии .Net Code Document Object Model [20], компилятор и компоновщик представлены в виде некоторого объекта в адресном пространстве процесса. С помощью технологии .Net reflection [19] сборка (Assembly [22]) производится сразу в адресном пространстве запущенного инструмента без создания физического DLL-файла (библиотеки). Взаимодействие с объектом-системой осуществляется через специальный параметризованный интерфейс:

```
internal interface ISystem<T>
{
  IInterval [] Bounds { get; }
  int Dimension { get; }
  IFunction<T> Function { get; }
  string InputCoveragePath { get; }
  string OutputCoveragePath { get; }
}
```

Многомерная функция представлена параметризованным интерфейсом, который имеет следующий вид:

```
public interface IFunction<T>
{
  T[] In { set; }
  T[] Out { get; }
  Evaluate();
  Node Root { get; }
}
```

Аргументы функции задаются через свойство *In*, а возвращаемое значение хранится в *Out*.

Свойство `Root` является корневым узлом дерева разбора заданного арифметического выражения. Это дерево используется при генерации кода класса-функции, а также при решении ряда специальных задач. Например, при вычислении константы Липшица (ее формула рекурсивна, поэтому нужна рекурсивная структура данных).

Поддержка POSIX стандартов. Эта библиотека реализована исключительно на *C++* , с поддержкой POSIX-платформ. В качестве компилятора была использована утилита *g++.exe*, а компоновщика – *dllwrap.exe*. Кроме того, для обработки сценариев компиляции применяется утилита *make.exe*. Указанные программы являются частью набора инструментов *MinGW* [26].

В этой реализации система также является параметризованным классом. Параметр – операнд в арифметике(тип *double* или *Interval*). Класс–система имеет следующий интерфейс доступа:

```
template<typename T> class System
 {
 public:
  SystemBounds::const_iterator boundsBegin()const;
  SystemBounds::const_iterator boundsEnd()const;
  T* Evaluate( T* );
  std::string GetInputCoveragePath()const;
  std::string GetOutputCoveragePath()const;
 };
```

4. ПРЕДСТАВЛЕНИЕ ЯЧЕЙКИ И РАЗЛИЧНЫЕ РЕАЛИЗАЦИИ АЛГОРИТМА

Основной задачей при локализации инвариантных множеств является реализация алгоритма построения образа ячейки. Существует несколько таких алгоритмов, которые представляют ячейку некоторым произвольным набором точек (например вершинами и центром), а в качестве образа рассматривают объединение ячеек, в которые эти образы попадают. При этом алгоритмы, которые являются наиболее предпочтительными с теорети-

ческой точки зрения, оказываются наиболее трудоемкими, поскольку значительно увеличивается количество вычислений функций, описывающих систему.

Таким образом, использование интервальной арифметики является хорошей альтернативой арифметики обычной. Мы вычисляем образ ячейки как образ множества, то есть вычисляем значение интервальной функции. Для заданной системы время выполнения операции вычисления образа ячейки есть некоторая константа.

Оптимизация при выборе структуры данных позволяет значительно сократить время вычислений по сравнению с алгоритмами локализации, основанными на обычной арифметике. Анализ алгоритмов локализации показывает, что наибольшее время занимает поиск ячейки в покрытии. Мы рассматриваем ячейки одного размера. В описываемых далее реализациях использовались представления ячейки целым числом, многомерным целочисленным вектором и элементом *R*-дерева. Проводились сравнения быстродействия. При решении задачи о локализации инвариантных множеств было реализовано 4 версии алгоритма:

1) базовая (I_0) [11],

2) модификация с распределенными вычислениями и использованием модуля динамической генерации кода (*I*) [9],

3) модификация *I* с поддержкой *R*-деревьев (*II*) [9],

4) версия *III* с поддержкой оптимального хранения ячеек.

Все приложения реализованы на языке C++. При реализации алгоритма была использована библиотека BOOST интервальной арифметики [24]. Текущая реализация этой библиотеки работает на платформах POSIX, Win32 и Macintosh Carbon.

Визуализация инвариантных множеств осуществляется с помощью технологии GNUPLOT [25].

4.1. ПРИЛОЖЕНИЕ *I*

В I_0 была реализована поддержка дискретных систем размерности 2. Вычисление арифметических выражений выполнялось традиционно – путем обхода дерева разбора. В реализацию *I* была добавлена поддержка непрерывных систем размерности 2, а также дискретных и непрерывных систем размерности 3. При организации процесса обработки непрерывных систем в *I* был реализован интервальный метод Рунге-Кутта 4-го порядка, разработанный Ю. Шокиным [7], и использованы приведенные оценки ширины интервала-решения, полученного данным методом.

Использование модуля динамической генерации кода позволило существенно оптимизировать процесс локализации инвариантных множеств.

В *I*, так же как и в I_0 для хранения ячеек покрытия используется структура данных множество(SET), а операции поиска, вставки и удаления производятся за время пропорциональное log *N*, где *N* – количество элементов в множестве. Моделью данных, соответствующей SET, является двунаправленный список, все элементы которого различны.

Так как операциями, существенно влияющими на скорость работы I_0, являются вычисление образа ячейки (Img) и пересечение образа ячейки с покрытием (Intersect), то на их выполнение тратится основная часть общего времени работы программы.

При реализации распределенных алгоритмов решались следующие задачи:

1) распараллеливание операций Img и Intersect;

2) синхронизация доступа к ячейкам текущего покрытия (каждая ячейка должна обрабатываться однократно, например, если операция Img для одной ячейки будет выполняться в нескольких потоках одновременно, тогда быстродействие обработки ДС может существенно снизиться);

3) синхронизация доступа к новому покрытию (для корректной модификации нового покрытия поток-обработчик должен единолично владеть этим ресурсом).

4.2. ПРИЛОЖЕНИЕ *II*

Так как операция *Intersect* сводится к перебору ячеек покрытия, то предлагае-

мая оптимизация ориентирована на выявление способов организации представления покрытия таким образом, чтобы поиск ячейки становился наиболее эффективным. Мы используем подход, предложенный в работе [21]. Он основан на понятии *R*-деревьев. В этом случае фазовое пространство логически представляется как последовательность иерархически вложенных многомерных кубов, а процесс поиска ячейки упрощается за счет выявления и отсечения тех областей покрытия, в которых ячейки быть не может.

В отличие от задач, связанных с использованием баз данных, нам достаточно хранить только координаты ячейки, поэтому потребность в хранении индексов многомерных объектов отпадает. Так как во всех реализациях этого алгоритма рассматриваются ячейки одного размера и используется целочисленная система координат, единица длины в которой есть размер ячейки, нам кажется естественным представление индексной записи в виде массива целых чисел – многомерного индекса ячейки, а листового узла – в виде упорядоченного списка таких кортежей, элементы которых различны. Не листовой узел в нашем случае представлен в виде списка пар: указатель на дочернюю вершину и целочисленный интервальный вектор. Если дочерняя вершина – листовой узел, тогда ограничивающий многомерный куб равен минимальному интервальному вектору, содержащему все номера ячеек этого листа. Иначе ограничивающий интервальный вектор равен минимальному интервальному вектору, содержащему все интервальные вектора из списка пар дочернего узла.

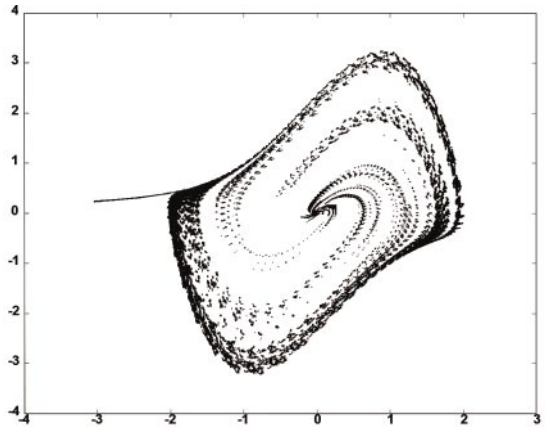

Рис. 1. Инвариантное множество отображения Ван-Дер-Поля

4.3. ПРИЛОЖЕНИЕ *III* С ПОДДЕРЖКОЙ ОПТИМАЛЬНОГО ХРАНЕНИЯ ЯЧЕЕК

Каждой ячейке сопоставляется уникальное натуральное число – индекс. Ячейки можно упорядочить по возрастанию согласно их порядковому номеру. Совокупность индексов ячеек представлена в виде одномерного массива машинных слов, длина которого вычисляется по следующей формуле: $\lceil count \div 8 \rceil$, где *count* – количество ячеек в покрытии.

При таком представлении ячейка покрытия может быть представлена 1 битом : значение 1 – ячейку нужно обработать. Операции пересечения и добавления осуществляются за время пропорциональное $O(1)$. В обоих случаях производится операция обращения к элементу массива по индексу (то есть адрес элемента с индексом *index* равен адресу массива + *index* * размер элемента массива (в байтах)). Для того, чтобы определить, принадлежит ли ячейка покрытию, необходимо вычислить 2 величины: индекс слова, в котором ячейка могла бы располагаться, а также бит в слове, который соответствует этой ячейке. Обозначим за *index* порядковый номер некоторой ячейки. Тогда индекс слова вычисляется по формуле: $word\_index = \lfloor index \div 8 \rfloor$. Позиция в слове: $pos\_index = index \bmod 8$, проверка вхождения ячейки в покрытие осуществляется следующим образом:
(1 << *pos_index*) & *coverage_array*[*word_index*].
Операция сдвига (<<) позволяет эффективно возводить 2 в степень *pos_index*. У полученного числа разряды, не стоящие на позиции *pos_index*, имеют нулевое значение. Результат операции & равен нулю, если у *coverage_array*[*word_index*] на позиции *pos_index* нулевой бит.

В этой реализации был использован упрощенный вариант распределенного

алгоритма локализации инвариантных множеств. Мютех используется только при выборе ячейки для обработки. Стек обработанных ячеек и поток-наполнитель отсутствуют. При таком подходе удалось избежать снижения производительности, связанного с наличием взаимных блокировок при наполнении нового покрытия. При потребности хранения всех ячеек покрытия (даже тех, которые не участвуют в процессе локализации) адресное пространство процесса существенно сократилось за счет оптимального представления ячейки в памяти (1 бит). При выборе ячейки был использован алгоритм с отсечением, основанный на специфике представления хранилища индексов ячеек: если машинное слово равно нулю, значит, данный блок не хранит ячеек и, значит, перебирать его разряды не нужно. Осуществляется переход к следующему слову.

4.4. ЧИСЛЕННЫЕ ЭКСПЕРИМЕНТЫ

Все вычисления использовали следующую программную и аппаратную конфигурацию:

– CPU: core 2 DUO: E6420 2.13 Ghz,
– Memory 2 Gb,
– OS: Microsoft Windows XP SP 2.

4.4.1. Система Ван-дер-Поля

Рассмотрим следующую систему:

$$\dot{x} = y$$
$$\dot{y} = 1.5(1 - x^2)y - x.$$

Инвариантное множество отображения Ван-Дер-Поля содержит точку покоя (0, 0) и периодическую орбиту. Вычисления проводились в области

$M = [-3.5, 3.5] \times [-3.5, 3.5]$.

Функция системы для построения представления символического образа задается при помощи 100 итераций метода Рунге-Кутта с шагом 0.001 (оператор сдвига на 0.1) (рис. 1).

4.4.2. Система О. Юнге

$$x_1 = y - \mu x,$$
$$y_1 = \lambda y(1 - x),$$
$$z_1 = x - \gamma z,$$

где $\lambda = 2.35$, $\mu = 0.5$, $\gamma = 0.1$.

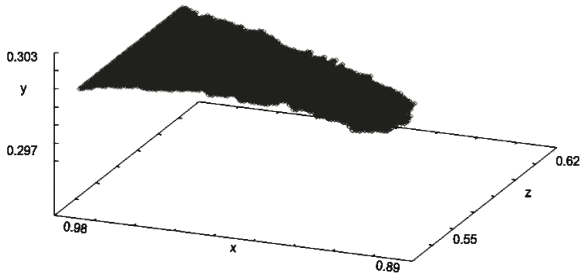

Рис. 2. Инвариантное множество системы О. Юнге

Начальная область

$\breve{M} = [-0.38, 0.98] \times [0.05, 1.45] \times [-0.38, 0.98]$.

Начальное разбиение – [100,100]. Количество подразбиений –10. Результат показан на рис. 2.

4.5. СРАВНИТЕЛЬНЫЕ ХАРАКТЕРИСТИКИ ПРОИЗВОДИТЕЛЬНОСТИ ПРИЛОЖЕНИЙ *I*, *II*, *III*

Были рассмотрены следующие системы.

Отображение с задержкой

$$x_1 = y,$$
$$y_1 = 2.27\ ay(1 - x).$$

Отображение Икеда

$$x_1 = 2 - 0.9(x\cos\tau(x, y) - y\sin\tau(x, y)),$$
$$y_1 = 0.9(y\cos\tau(x, y) + x\sin\tau(x, y)),$$
$$\tau(x, y) = 0.4 - \frac{6}{1 + x^2 + y^2}.$$

Система Дуффинга

$$\dot{x} = y,$$
$$\dot{y} = x - x^3 - 0.15y.$$

Для системы Дуффинга вычисление образа ячейки производилось по методу Рунге-Кутта с шагом t, равным 0.03. Количество итераций – 100.

В приведенных далее табл. 1 и диаграмме (рис. 3) используются следующие обозначения:

Табл. 1. Сравнительные характеристики работы различных реализаций алгоритма

Система	*I*	*II*	*III*
A	61.875 sec	48.045 sec	18.047 sec
B	237.192 sec	198.437 sec	71.609 sec
C	135.501 sec	107.782 sec	43.641 sec
D	540.328 sec	429.593 sec	166.767 sec
E	386.615 sec	350.890 sec	191.822 sec
F	1417.016 sec	1263.472 sec	593.032 sec

A – отображение с задержкой (в разбиении 4 миллиона ячеек),

B – отображение с задержкой (в разбиении 16 миллионов ячеек),

C – отображение Икеда (в разбиении 4 миллиона ячеек),

D – отображение Икеда (в разбиении 16 миллионов ячеек),

E – система Дуффинга (в разбиении 4 миллиона ячеек),

F – система Дуффинга (в разбиении 16 миллионов ячеек).

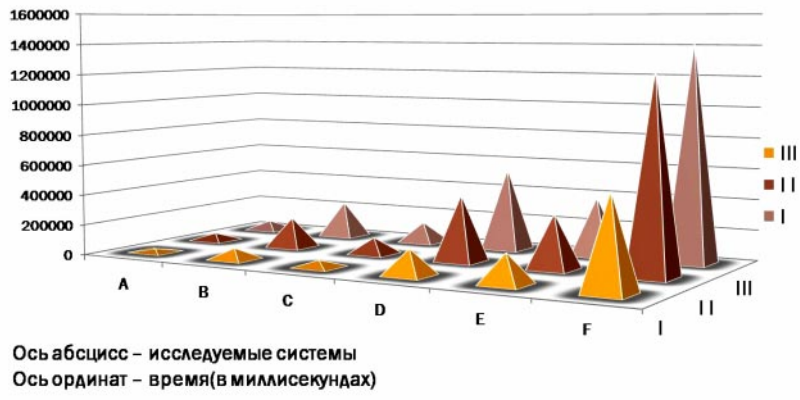

Рис. 3. Диаграмма сравнительных характеристик работы различных реализаций алгоритма

5. ЗАКЛЮЧЕНИЕ

В работе описаны реализации алгоритмов локализации инвариантных множеств динамических систем, основанные на методах интервальной арифметики и символического образа. Основное внимание уделено способам оптимизации, а именно представлению данных и оптимизации вычислений. Реализованы версии алгоритма с различными оптимизациями и проведено сравнение их эффективности. Результаты численных экспериментов показали, что оптимальным является представление ячейки одним битом, означающим ее присутствие в текущем покрытии. При оптимизации вычислений используется динамическая генерация кода. Это позволяет отказаться от использования дерева разбора арифметического выражения при вычислениях. Описанные алгоритмы могут быть применены как в учебном процессе при моделировании динамических систем, так и в исследовательской работе.

Литература

1. *Dellnitz M., Junge O.* Set oriented numerical methods for dynamical systems. Handbook of dynamical systems. Vol. 2. Ed. B. Fiedler, 2002.

2. *Осипенко Г.С.* О символическом образе динамической системы. Сб. Граничные задачи. Пермь, 1983. С. 101–105.

3. *Осипенко Г.С., Ампилова Н.Б.* Введение в символический анализ динамических систем. СПб.: Изд. СПбГУ, 2005.

4. *Ахо А., Сети Р., Ульман Дж.* Компиляторы: принципы, технологии, инструменты. М.: Издательский дом «Вильямс», 2003.

5. *Каток А.Б. и Хассельблат Б.* Введение в современную теорию динамических систем // М.: Факториал, 1999.

6. *Меньшиков Г.Г.* Интервальный анализ и методы вычислений. СПб.: НИИХ СПбГУ, 1998–2001. Вып. 1. Введение в интервальную организацию вычислений // www.apmath.spbu.ru/ru/education/courses/elective/menshikov.html.

7. *Шокин Ю.И.* Интервальный анализ. Новосибирск: Сибирское отделение изд-ва «Наука», 1981. С. 64–68 // http://www.nsc.ru/interval/index.php?j=Library/InteBooks/index.

8. *Шарый С.П.* Конечномерный интервальный анализ, «XYZ», 2009.

9. *Терентьев С.В.* Разработка и реализация основанных на интервальной арифметике алгоритмов компьютерного исследования динамических систем / Дисс. на соиск. к.ф.м.н, СПб.: СПбГУ, 2010.

10. *Dellnitz M. and Hohmann A.* A Subdivision Algorithm for the Computation of Unstable Manifolds and Global Attractors // Numerische Mathematik, 1997. Vol. 75, № 3. P. 293–317.

11. *Ампилова Н.Б. Терентьев С.В.* Применение методов интервальной арифметики к задаче построения символического образа // Электронный журнал «Дифференциальные уравнения и процессы управления», 2006. № 4. http://www.neva.ru/journal/j/index.html.

12. *Dellnitz M., Junge O.* An adaptive subdivision technique for the approximation of attractors and invariant measures // Comput. Visual. Sci., 1998. № 1. P. 63–68.

13. *Galias Z.* Rigorous investigation of the Ikeda map by means of interval arithmetic // Nonlinearity 15(2002). P. 1759–1779.

14. *Ikeda K.* Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system, Opt. Commun. 30 (1979). P. 257–261.

15. *Jaulin L., Kieffer M., Didrit O., Walter E.* Applied interval analysis. Berlin: Springer, 2001.

16. *Mischaikow K. and Mrozek M.* Chaos in the Lorenz equations: A computer assisted proof. Part II: Details // Mathematics of Computation, 1998, July. Vol. 67, № 223. P. 1023–1046.

17. *Neumaier A.* Interval Methods for systems of equations, Cambridge University Press, 1990.

18. *Tucker W.* A rigorous ODE solver and Smale's 14th problem. Found. Comput. Math., 2002. № 2. P. 53–117.

19. .NET Framework Developer's Guide. Reflection. Microsoft Developers Network // http://msdn.microsoft.com/en-us/library/cxz4wk15.aspx.

20. *Kathleen Dollard.* Code Generation in Microsoft .NET. Apress, 2004. P. 760.

21. *Guttman A.* R-Trees: A Dynamic Index Structure for Spatial Searching. Proc. 1984 ACM SIGMOD International Conference on Management of Data. P. 47–57 // http://www.sai.msu.su/~megera/postgres/gist/papers/gutman-rtree.pdf.

22. MSDN (http://msdn.microsoft.com/library/).

23. Standard Template Library (http://msdn2.microsoft.com/en-us/library/c191tb28(VS.80).aspx).

24. BOOST (http://www.boost.org).

25. Standard Template Library (http://gnuplot.sourceforge.net/).

26. MinGW (http://www.mingw.org/).

27. Antlr (http://www.antlr.org/).

Abstract

The paper is devoted to the design and implementation of the algorithm for localizing invariant sets of dynamical systems. The algorithm is based on interval arithmetics methods. The system is approximated by using a symbolic image which is an oriented graph constructed in accordance with the system. The cells of a covering are supposed to be interval vectors. The degree of approximation to the invariant set is estimated through the symbolic image parameters. The results of numerical experiments and the comparison with the algorithms based on real arithmetics are given.

Keywords: dynamical systems, invariant sets, symbolic image, computer simulation, interval arithmetics.

Ампилова Наталья Борисовна,
кандидат физико-математических наук, доцент кафедры информатики математико-механического факультета СПбГУ,
nataly@is1483.spb.edu,

Терентьев Сергей Валерьевич,
кандидат физико-математических наук, инженер-программист Peter-Service,
waterq@yandex.ru.