CAPD::DynSys: a flexible C++ toolbox for rigorous numerical analysis of dynamical systems

Tomasz Kapela[1], Marian Mrozek[1], Daniel Wilczak[1,*], Piotr Zgliczyński[2,]

Faculty of Mathematics and Computer Science, Jagiellonian University, Łojasiewicza 6, 30-348 Kraków, Poland.

Abstract

We present the CAPD::DynSys library for rigorous numerical analysis of dynamical systems. The basic interface is described together with several interesting case studies illustrating how it can be used for computer-assisted proofs in dynamics of ODEs.

Keywords: rigorous numerical analysis, C++ library, computer-assisted proof
2010 MSC: 65G20,
2010 MSC: 37C27

1. Introduction

In the study of nonlinear ODEs, there is a huge gap between what we can observe in numerical simulations and what we can prove rigorously. It is possible to overcome this problem by means of computer-assisted poofs. For its realization it is desirable to have a library for rigorous integration of ODEs and computation of Poincaré maps derived from ODEs. There are several libraries designed for rigorous integration of ODEs. Some of them are open source, just to mention [1, 2, 3], and some are not [4]. To the best of our knowledge none of them directly supports computation of Poincaré maps, which is a powerful tool for studying dynamics of ODEs.

In the present paper we describe the CAPD::DynSys library [5] which is well suited for this task. What this library offers may be described as follows.

Consider an initial value problem for an ODE

$$x' = f(\lambda, t, x), \tag{1}$$

$$x(t_0) = x_0, \tag{2}$$

where $x \in \mathbb{R}^n$, t is a time variable, $\lambda \in \Lambda \subset \mathbb{R}^k$ is a fixed parameter and $f : \Lambda \times \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is a smooth, 'programmable' function. Let $\varphi(t, t_0, \lambda, x_0)$ be a solution of (1)–(2) for some fixed λ. Given a set of initial conditions $Z \subset \mathbb{R}^n$, parameters $\Delta \subset \Lambda$ and a $t > t_0$ we want to:

*Corresponding author.
Email addresses: Tomasz.Kapela@uj.edu.pl (Tomasz Kapela), Marian.Mrozek@ii.uj.edu.pl (Marian Mrozek), Daniel.Wilczak@ii.uj.edu.pl (Daniel Wilczak), Piotr.Zgliczynski@ii.uj.edu.pl (Piotr Zgliczyński)

[1]This research is partially supported by the Polish National Science Center under Maestro Grant No. 2014/14/A/ST1/00453 and under Grant No. 2015/19/B/ST1/01454.

[2]This research is partially supported by the Polish National Science Center under Grant No. 2019/35/B/ST1/00655.

- establish that for all $x_0 \in Z$ and $\lambda \in \Delta$ the solution $\varphi(t, t_0, \lambda, x_0)$ is defined,
- give a rigorous bound for $D_a\varphi(t, t_0, \lambda, x_0)$ valid for all $x_0 \in Z$ and $\lambda \in \Delta$, where D_a is the partial derivative operator with respect x_0 and/or λ of order r. The case $r = 0$ means that we compute rigorous bounds for $\varphi(t, t_0, \Delta, Z)$.

Analogous questions can be asked for Poincaré maps for ODEs.

The CAPD::DynSys library can accomplish these tasks for many interesting systems for n not too large (an example of a 'large' value of n up to which the library performed well is $n = 80$, which was used for choreografies in N-body problem) and the order of the partial derivatives is also not too large, say $r = 2, 3$ (the library handled $r = 5$ in validation of KAM tori [6]) if the sizes of initial conditions are not too big and the integration time $t - t_0$ is not very large.

The fact that we can compute (enclose) $\varphi(t, t_0, \Delta, Z)$ in single computation is essential for the computer assisted proofs, as very often abstract theorems in dynamics involve assumptions on the behavior of solutions on sets, while the single trajectory computations usually do not lead to interesting rigorous statements about the dynamics of the underlining ODE.

In the following discussion we will say that we performed C^r computations if the partial derivatives up to order r have been computed.

1.1. Short history

CAPD is an acronym for "Computer Assisted Proofs in Dynamics". The library was initiated in early 1990's by Marian Mrozek as the tool for the computer assisted proof of chaotic dynamics in the Lorenz system [7, 8, 9]. The library is split into CAPD::DynSys devoted to the dynamics and discussed in this paper and CAPD::RedHom devoted to topology [10]. The DynSys part was developed by Mrozek's Ph. D. students and their descendants at the Jagiellonian University in Krakow, Poland. The most important contributors are (listed more or less chronologically): P. Zgliczyński, P. Pilarczyk, D. Wilczak, T. Kapela.

Papers of Mischaikow, Mrozek and Szymczak on the Lorenz attractor [7, 8, 9] used C^0 computations, only. Other results using C^0 computations in CAPD library from these early stages of development are

- symbolic dynamics for the Rössler system [11],
- connecting orbits in the Michelson system [12],
- periodic orbits in the Rössler system [13].

The early version of C^0-integrator was based on the logarithmic norms and it was slow and inefficient, when compared to the algorithms currently used by the library.

Around 2000 the Lohner algorithm [14] (for C^0-computations) and C^1-Lohner type algorithm [15] for efficient C^1-computations were implemented in the CAPD library. The initial implementation was for second order polynomial vector fields, only. Around that time Daniel Wilczak joined the project and implemented these algorithms for general 'programmable' vector fields using the automatic differentiation. The CAPD library was published online for the first time in 2004. Around the year 2008 the C^r-Lohner algorithm [16] was added to the library, and soon after this a rigorous solver of differential inclusions [17] and support for computation in high precision were written by Tomasz Kapela.

1.2. Some computer assisted proofs using CAPD library

The quality of the bounds provided by the CAPD::DynSys library can be judged by looking at the list of computer-assisted proofs in dynamics of ODEs in which it was used. The list below is incomplete, we focus only on a number of selected applications. The results using C^0- and C^1-computations include the questions of the existence of periodic orbits and their local uniqueness, the existence of symbolic dynamics, the existence of hyperbolic invariants sets, the existence of homo- and heteroclinic orbits. Here are some examples

- symbolic dynamics in the Hénon-Heiles Hamiltonian [18],
- symbolic dynamics and symmetric periodic orbits in Michelson system [19],
- homoclinic and heteroclinic connections between Lyapunov orbits and symbolic dynamics in the planar circular restricted three body problem [20, 21],
- Shilnikov orbits and Bykov cycles in the Michelson system [22],
- existence of choreographies in Newtonian N-body problem [23, 24],
- hyperbolic Smale-Williams attractor for Kuznetsov System [25],
- invariant manifolds in the restricted three body problem by Capiński and his coworkers [26, 27],
- Birkhoff regions of instability in the three body problem, using Aubry-Mather theory, by Galante and Kaloshin [28],
- existence of double spiral attractor in the Chua's circuits by Galias and Tucker [29],
- counting of periodic orbits of flows by Galias and Tucker [30, 31],
- stability of N-body motions forming platonic polyhedra by Fenucci and Gronchi [32],
- study of periodic by orbits by Miyaji and Okamoto [33],
- existence of unimodal solutions in the Proudman–Johnson equation by Miyaji and Okamoto [34],
- study of singularities in dynamical systems by Matsue [35],
- applications to rigorous estimates of reachable sets in the context of control theory and robotic by Jaulin and his coauthors [36, 37] and Cyranka et al. [38],
- heteroclinic connections in Ohta–Kawasaki Model by Cyranka and Wanner [39].

To address other phenomena, such as bifurcations of periodic orbits, invariant tori through the KAM theory, nonlinear stability of elliptic periodic orbits, KAM stability etc. one needs the knowledge of partial derivatives with respect to the initial conditions of the higher order. Using algorithms from CAPD::DynSys library for C^r-computations the following results has been obtained

- global and local bifurcations of periodic orbits and invariant manifolds [40, 41, 42, 43],
- Arnold diffusion in the restricted three body problem by Capiński and Gidea [44],
- non-linear stability of elliptic periodic solutions [45, 6, 46],
- normally hyperbolic invariant manifolds and computer assisted Melnikov method [47, 48].

1.3. Outline of the paper

In Section 2 we describe the basic interface to CAPD::DynSys library. In Sections 3, 4 and 5 we present a list of case studies on how the CAPD::DynSys library can be used in various contexts. Examples are grouped by the maximal order of space derivatives involved – we refer to them as C^0, C^1 and C^r computations. The examples selected here are on the one hand very short (so that it is possible to write out the full C++ code) but on the other side are non-trivial and present diverse spectrum of mathematical problems, where the CAPD::DynSys library may be helpful.

2. The CAPD library: interface and basic usage

The CAPD::DynSys library provides data structures and algorithms designed for analysis of discrete and continuous dynamical systems in finite and infinite dimension. They are written in the spirit of generic programming with high level of abstraction allowing the user to tune or adapt some subroutines for specific problems.

The CAPD::DynSys library provides algorithms for both non-rigorous and rigorous computation. The non-rigorous ones are mainly used for simulation, prototyping or finding approximations of objects we are interested in. They are based on double precision floating point numbers supported by hardware so they are fast but prone to errors coming from rounding, significant bits cancellation, inaccuracy of numerical method etc. On the other hand, rigorous methods are group of algorithms, which compute an outer bounds of the objects we are interested in (like values and derivatives of maps, solutions to IVPs).

Most often used interface of the CAPD::DynSys library is available via the following two header files

```
#include "capd/capdlib.h"    // CAPD library header
#include "capd/mpcapdlib.h" // Multi-precision CAPD header
using namespace capd;
```

All types and algorithms are defined in the main namespace `capd`.

2.1. Basic arithmetic types and naming convention.

In the CAPD::DynSys library the special type `interval` provides interval arithmetic (see [49]) and is a base for all data types used in rigorous computations. The precision provided by built-in floating point types is sometimes not sufficient and causes huge overestimation in rigorous computations. The CAPD::DynSys library defines MpFloat and MpInterval types that provide floating point numbers and intervals, respectively, of arbitrary precision. The implementation is based on the MPFR library [50]. The following example shows the basic usage of the above four arithmetic types.

```
#include <iostream>          // C++ standard output libary
#include "capd/capdlib.h"    // CAPD library header
#include "capd/mpcapdlib.h" // Multi-precision CAPD header
using namespace std;
using namespace capd;

template <typename T>
T f(T x){
  return sqr(sin(x))*exp(x) + 2*x*(cos(x));
}
```

```
int main(){
  cout.precision(17);
    // Non-rigorous computations
  double x = 0.51342;
  cout << "f=" << f(x) << endl;
    // Rigorous computations using
    // interval arithmetics with hardware support (53 mantissa bits)
  interval iy = f(interval(x));
  cout << "f=" << iy << ", width = " <<  width(iy) << endl;
    // Non-rigorous computations using multiprecision arithmetics
  cout.precision(60);
  MpFloat::setDefaultPrecision(200);  // with 200 mantissa bits
  cout << "f=" << f(MpFloat(x)) << endl;
    // Rigorous computations using multiprecision interval arithmetics
  MpInterval mpfx = f(MpInterval(x));
  cout << "f=" << mpfx << ",\nwidth = " <<  width(mpfx) << endl;
}
/* Output:
f=1.2975560311330947
f=[1.2975560311330918, 1.2975560311330983], width = 6.4392935428259079e-15
f=1.2975560311330945234794709162008584476628110771288848512335 9
f=[1.2975560311330945234794709162008584476628110771288848512335 8
    ,1.2975560311330945234794709162008584476628110771288848512335 9 ],
width = 3.7338091667166850242864384322680745443541513012327002798606 7e-60
*/
```

On top of these four basic arithmetic types the CAPD::DynSys library builds data structures such as vectors, matrices, hessians, jets (truncated Taylor series) and algorithms for manipulating them. Other data structures represent functions, solutions to ODEs or Poincaré maps, etc. Most of defined types use the following naming convention pattern

```
[Prefix]ClassName
```

for example

```
  DVector,   DMatrix,   DJet,   DMap,   DOdeSolver,   DPoincareMap, ...
 MpVector,  MpMatrix,  MpJet,  MpMap,  MpOdeSolver,  MpPoincareMap, ...
  IVector,   IMatrix,   IJet,   IMap,   IOdeSolver,   IPoincareMap, ...
MpIVector, MpIMatrix, MpIJet, MpIMap, MpIOdeSolver, MpIPoincareMap, ...
```

Prefixes `D` and `Mp` mean that the class is designed for non-rigorous computation based on `double` and `MpFloat` arithmetic types, respectively. Similarly, classes with prefixes `I` and `MpI` provide data structures and rigorous algorithms based respectively on `interval` and `MpInterval`. Whenever possible, we try to provide common interface for all kinds of data types and algorithms so that it is possible to switch between them if needed.

2.2. Maps and their Taylor coefficients.

One of the most important types is the class `[Prefix]Map` which represents a (possibly parameter dependent) map

$$f_a : \mathbb{R} \times \mathbb{R}^n \ni (t, x_1, x_2, \ldots, x_n) \to (f_1, f_2, \ldots, f_m) \in \mathbb{R}^m,$$

where $a = (a_1, a_2, \ldots, a_k)$ for some $k \geq 0$ is a vector of parameters. This class is usually used to define a generator of a discrete dynamical system or a vector field. The special time variable can be used to define non-autonomous vector fields.

This class provides also an easy to use interface for computation of higher order Taylor coefficients of the underlying map by means of automatic differentiation [51]. An instance of `Map` can be created by means of two constructors. If the map is given by a short formula it is convenient to parse the expression from a string with the following syntax

```
IMap f("par:a1,a2,...,ak;time:t;var:x1,x2,...,xn;fun:f1,f2,...,fm;");
```

The sections `par` and `time` are optional. More complicated expressions can be defined as C++ functions with the signature

```
void f(capd::autodiff::Node t,               // time variable
  capd::autodiff::Node in[], int dimIn,      // input variables x1,...,xn
  capd::autodiff::Node out[], int dimOut,    // output: function values
  capd::autodiff::Node params[], int noParam // parameters
);
```

and then sent to the constructor of `Map`. Below we present a short example illustrating the usage of both constructors and how the class can be used to compute values and derivatives of represented function.

```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;
/** Ikeda map is given by
 X = 1+u*(x*cos(r)-y*sin(r))
 Y =   u*(x*sin(r)+x*cos(r))
 where
 r = p - 6/(1+x^2+y^2)
 and p,u are parameters */
void ikeda(capd::autodiff::Node /*t*/,       // unused time variable
  capd::autodiff::Node in[], int dimIn,      // input variables x,
  capd::autodiff::Node out[], int dimOut,    // output: function values X,Y
  capd::autodiff::Node params[], int noParam // parameters: p,u
){
  capd::autodiff::Node r = params[0]-6./(1.+sqr(in[0])+sqr(in[1]));
  capd::autodiff::Node s = sin(r), c = cos(r);
  out[0] = 1+params[1]*(in[0]*c - in[1]*s);
  out[1] = params[1]*(in[0]*s + in[1]*c);
}

int main(){
  IMap Henon("par:a,b;var:x,y;fun:y+1-a*x^2,b*x;");
  // Set parameter a=1.4. Note: it is not representable.
  Henon.setParameter("a",interval(14)/10.);
  // Set b\in [0.2,0.4], which contains the standard one b=0.3
  Henon.setParameter("b",interval(2, 4)/10.);
  IVector ix { {-1,2}, {0,1} };  // ix =[-1,2] x [0,1]
  IVector hx = Henon(ix);        // enclose Henon_{a,b}(ix)
  // Enclose derivative D Henon_{a,b}(ix)
  // for each point x in ix and for parameters a,b as above
  IMatrix Dhx = Henon.derivative(ix);

  int dimIn=2, dimOut=2, noParams=2, highestDerivative=5;
  IMap Ikeda(ikeda,dimIn,dimOut,noParams,highestDerivative);
  // Set parameters p=0.4, u=0.75
  Ikeda.setParameters({interval(4)/10,interval(0.75)});
  IVector x {{1.,1.},{1.5,1.5}};
  // Container for Taylor coefficient of 2-dimensional map up to order 4
```

```
  IJet jet(2,4);
  Ikeda(ix, jet);   // Compute Taylor expansion of Ikeda map at point ix
  cout << jet.toString() << endl;    // and print in human readable form
  // Access to first order derivatives
  cout << jet(0,0) << ", " << jet(0,1) << endl;
  cout << jet(1,0) << ", " << jet(1,1) << endl;
  // jet(i,j,c) gives access to normalized derivative ∂²f_i/∂x_j∂x_c
  cout << jet(0,0,1) << ", " << jet(1,0,0) << endl; // ∂²f_0/∂x_0∂x_1 and (1/2!) ∂²f_1/∂x_0²
  // jet(i,j,c,k) gives access to normalized derivative ∂³f_i/∂x_j∂x_c∂x_k
  cout << jet(1,0,0,1) << endl; // (1/2!) ∂³f_1/∂x_0²∂x_1
  // For higher order Taylor coefficients use Multiindex notation
  cout << jet( Multiindex({2,2})) << endl; // Access to vector (1/2!2!) ∂⁴f/∂x_0²∂x_1²
}
```

2.3. Solving initial value problems.

Algorithms which solve initial value problems (IVPs) are split between three groups of classes.

One-step solvers. The first group consists of

`[Prefix]OdeSolver, [Prefix]CnOdeSolver`

The above classes provide algorithms for one-step integration of ODEs. The class `OdeSolver` is optimized for C^0 and C^1 integration, that is solutions to IVPs and/or associated first order variational equations. The second class `CnOdeSolver` can integrate higher order variational equations as well. An example of its usage will be given in Section 5.

Long-time integration. The above one-step methods are in general not recommended for direct usage. The second group of classes is built on top of `[Cn]OdeSolver`, that is

```
[Prefix]TimeMap, [Prefix]CnTimeMap,
[Prefix]PoincareMap, [Prefix]CnPoincareMap.
```

Class `[Cn]TimeMap` combines a one-step solver with automatic step control strategies to compute trajectory segment over (usually) large time range. If integration time is not given explicitly but is determined by reaching certain Poincaré section, then one should use `[Cn]PoincareMap`. This class provides algorithms for computation of Poincaré maps and their derivatives. In the CAPD::DynSys library a Poincaré section is always defined as the set of zeroes of a smooth scalar-valued function $S : \mathbb{R}^m \to \mathbb{R}$ and realized by classes

`[Prefix]NonlinearSection, [Prefix]AffineSection, [Prefix]CoordinateSection.`

The most general nonlinear case is covered by `NonlinearSection`. The library provides also computationally more efficient `AffineSection`, where the section is a hyperplane given by the normal vector $n \in \mathbb{R}^m$ and translation $c \in \mathbb{R}^m$, that is $S(x) = \langle n, x - c\rangle$. The last, and very often used type of section is `CoordinateSection`, where $S(x) = x_i - c$ for some $i \in 1, \ldots, m$ and a constant $c \in \mathbb{R}$.

Sets and their propagation. The third group consists of classes which specify different ways of representation of initial conditions and their propagation along trajectories. In rigorous computations special care should be taken on how intermediate results are represented. When a set of initial condition is propagated by a dynamical system and on each step the image is bounded

by an interval vector (product of intervals), then typically we observe the wrapping effect that leads to huge overestimation. On the other hand when the image is bounded by some non-linear shape, e.g given by multidimensional polynomials (like in the case of Taylor models [4]), then the result is more accurate but the computational cost increases rapidly with the dimension and degree of the polynomial.

In the CAPD::DynSys library the sets are represented (see [52]) as parallelepipeds, doubletons and tripletons. These strategies provide good compromise between speed and accuracy as shown in [53]. To choose appropriate set representation there are several factors to consider:

- What set geometry will bring good compromise between speed and accuracy? The two that are usually the most efficient are
  - `Rect2` - doubleton representation of the form $x + C * r0 + Q * q$ where $x, q, r0$ are interval vectors (x is a point interval vector) and C, Q are interval matrices, with Q close to orthogonal.
  - `Tripleton` - a subset of $\mathbb{R}^m$ in the form $x + C * r0 + \text{intersection}(B * r, Q * q)$ where $x, q, r, r0$ are interval vectors (x is a point interval vector) and C, B, Q are interval matrices, with Q close to orthogonal.
- What order of derivatives with respect to initial condition do we need? It is indicated by the prefix: `C0` sets enclose only the trajectory, `C1` sets enclose also first order derivatives with respect to initial conditions and `Cn` sets are used to store jets of flow up to given order, which has to be specified at the set construction.
- What numerical method should be used to propagate the set? There are two main groups of methods implemented in CAPD::DynSys: one based on the Taylor method and second based on the Hermite-Obreshkov (explicit-implicit) formula [54]. The infix `HO` indicates that the Hermite-Obreshkov method is requested.
- Is default double precision enough? If not add `Mp` prefix to compute with arbitrary precision (paying appropriate computational cost).

Summarizing, the names of data structures which represent initial conditions for ODEs follow the pattern

```
[Mp]Cx[HO]GeometrySet
```

Not all combinations of components are implemented (please consult documentation for the full list of supported set representations), but the above pattern helps to encode a set representation type. For example

- `C0HOTripletonSet` stores C^0 information only using tripleton representation and is propagated by the Hermite-Obershkov method [54],
- `C1HORect2Set` stores points on the trajectory and first order derivatives with respect to initial conditions both in the form of doubletons and uses Hermite-Obreshkov method [55] for their propagation.
- `MpCnRect2Set` stores values and all derivatives up to given order in form of doubletons with `MpInterval` coefficients and propagates them by the Taylor method.

The following short code illustrates basic usage of the above three groups of classes.

```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;

int main(){
  cout.precision(11);
  // define vector field, solver and classes for long time integration
  IMap pendulum("par:w,pi;time:t;var:x,dx;fun:dx,cos(pi*t)-w*dx-sin(x);");
  pendulum.setParameters({interval(1)/100,interval::pi()});
  IOdeSolver solver(pendulum, 20);
  ITimeMap tm(solver);
  ICoordinateSection section(2,0); // two variables, index of x=0 is 0
  // alternatively, we could here use:
  // INonlinearSection section("var:x,y;fun:x;");
  // IAffineSection section(IVector(2),IVector({1.,0.})); //(origin,normal)
  IPoincareMap pm(solver,section);

  // integrate set [1,1]\times [0.25,0.5] over T=1
  C0HOTripletonSet s0( IVector({{1.,1.}, {0.25,0.5}}) );
  cout << "y=" << tm(1.,s0) << endl;

  // integrate point (1,0.5) and variational equations over T=100 and then
  // print image and solution to variational equation (monodromy matrix)
  C1Rect2Set s1( IVector({{1.,1.}, {0.5,0.5}}) );
  cout << "u=" << tm(100.,s1) << endl;
  cout << "D=" << (IMatrix)s1 << endl;

  // continue integration of s1 until it reaches the section x=0
  IMatrix DP(2,2);
  IVector P = pm(s1,DP);
  cout << "P=" << P << endl;
  cout << "DP=" << DP << endl;
  // After computation P = pm(s1,DP) the set s1 is often far from section.
  cout << "s1=" << (IVector)s1 << endl;
  cout << "Ds1=" << (IMatrix)s1 << endl;
}
/* Output (rounded to 11 digits):
y={[1.0115905491, 1.2463432271],[-0.6386533516, -0.42846233913]}
u={[0.38323812527, 0.38323812528],[0.50996214875, 0.50996214877]}
D={
{[-4.3182170195, -4.3182170176],[-3.3700059498, -3.3700059482]},
{[4.3928041834, 4.3928041856],[3.3430225316, 3.3430225334]}
}
P={[-1.8354535396e-11, 1.8354535396e-11],[-0.34402409982, -0.34402409981]}
DP={
{[6.2681785123, 6.2681785151],[4.8172609505, 4.8172609528]},
{[-0.43855824706, -0.43855824587],[-0.27979305979, -0.27979305885]}
}
s1={[-0.063583452514, -0.063583452497],[-0.37811336848, -0.37811336848]}
Ds1={
{[6.0890412578, 6.0890412606],[4.6898120176, 4.6898120199]},
{[-1.5492512746, -1.5492512733],[-1.1344125491, -1.1344125481]}
} */
```

2.4. The role of coordinate systems in integration of ODEs and computation of Poincaré maps

The data structures, which represent initial conditions for ODEs provide constructors, that allow to set $x0, C, r0$ in both doubleton and tripleton representation. A proper usage of them can significantly improve obtained bounds. We will illustrate this issue with two suggestive examples.

Consider the pendulum equation $x'' = -\sin(x)$ and the following IVP: $x(0) \in [2,3]$ and $x'(0) = 5 - x(0)$, that is a line segment joining points $(2,3)$ and $(3,2)$ in the phase space. The following short code shows huge difference between two bounds on $(x(2), x'(2))$, depending on how this line segment is initially represented. The set `s1` represents initial conditions as

$$x0 + C * r0 = \begin{bmatrix} 2.5 \\ 2.5 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} [-0.5, 0.5] \\ [0,0] \end{bmatrix},$$

while `s2` wraps it to the smallest interval vector, which contains this line segment, that is $(x(0), x'(0)) \in [2,3]^2$.

```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;

int main(){
  IMap vf("var:x,dx;fun:dx,-sin(x);");
  IOdeSolver solver(vf,10);
  ITimeMap tm(solver);
  IVector x0({2.5,2.5}), r0({{-0.5,0.5},{0.,0.}});
  IMatrix C({{1.,1.},{-1.,1.}});
  C0HOTripletonSet s1(x0,C,r0), s2(x0+C*r0);
  std::cout << "s1(t=2): " << tm(2.,s1) << std::endl;
  std::cout << "s2(t=2): " << tm(2.,s2) << std::endl;
}
/** Output (rounded to 6 digits):
s1(t=2): {[7.821, 8.33752],[2.36664, 3.07598]}
s2(t=2): {[4.4177, 11.7408],[-0.930075, 6.3727]} */
```

Setting coordinate system on the Poincaré section also matters. Let us consider the Lorenz system [56]

$$x' = 10(y - x), \quad y' = x(28 - z), \quad z' = xy - \frac{8}{3}z.$$

Let us fix the Poincaré section $\Pi = \{(x, y, z) : z = 27\}$ and denote by $\mathcal{P} : \Pi \to \Pi$ the corresponding Poincaré map. We will use coordinates (x, y) to describe points on Π. Define $\alpha = 7\pi/18$ and

$$Q_\alpha = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}.$$

A computer-assisted proof of the existence of chaotic dynamics in the Lorenz system given by Galias and Zgliczyński [57] required in particular, that the following inequality holds true:

$$\left|\pi_y Q_\alpha^{-1} \mathcal{P}^2(u)\right| < 3.6 \quad \text{for} \quad u = Q_\alpha \cdot (s, 0),\ s \in [0.625, 0.675]. \tag{3}$$

The following program illustrates the difference between multiplication by Q_α^{-1} after computation of Poincaré map, that is $Q_\alpha^{-1}\left(\mathcal{P}^2(u)\right)$ and computation of $\left(Q_\alpha^{-1}\mathcal{P}^2\right)(u)$ in a single routine. We see that the second estimate is much tighter and, in particular, the requested inequality (3) is validated.

```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;

int main(){
  interval alpha = 7.*interval::pi()/18;
  interval c = cos(alpha), s = sin(alpha), z = 0.;
  IMatrix Q ({{c,-s,z},{s,c,z},{z,z,interval(1.)}});
  IMatrix invQ = transpose(Q);

  IMap lorenz("par:s;var:x,y,z;fun:10*(y-x),x*(28-z)-y,x*y-s*z;");
  lorenz.setParameter(0,interval(8)/3);
  IOdeSolver solver(lorenz,30);         // ODE integrator
  ICoordinateSection section(3,2,27.); // section is z=27
  IPoincareMap pm(solver,section);

  IVector r0(interval(625,675)/1000,-interval(36)/10,27);
  C0HOTripletonSet s1(IVector(3),Q,r0), s2 = s1;
  interval returnTime;
  // here we print  \pi_y Q^{-1}\left(\mathcal{P}^2(x0 + C * r0)\right)
  std::cout << "y1: " << (invQ*pm(s1,2))[1] << std::endl;
  // here we print  \pi_y \left(Q^{-1}\mathcal{P}^2\right)(x0 + C * r0)
  std::cout << "y2: " << pm(s2,IVector(3),invQ,returnTime,2)[1];
}
/** Output (rounded to 6 digits):
y1: [-4.16179, -0.129486]
y2: [-3.18066, -1.19211] */
```

3. C^0 solver and its applications

Topological methods are powerful and inexpensive in comparison to methods requiring estimates on derivatives. In this section we present two case studies:

- the existence of symmetric periodic orbits in the Michelson system and
- the existence of an attractor for the Rössler system.

3.1. Periodic orbits in the Michelson system

The Michelson system [58] is a 3D system

$$x' = y, \qquad y' = z, \qquad z' = c^2 - y - x^2/2. \tag{4}$$

reversible with respect to the involution

$$\mathcal{R} : (x, y, z) \to (-x, y, -z).$$

The above symmetry maps trajectories onto trajectories of the system but reverses the time, that is $\mathcal{R}(\phi(t, u)) = \phi(-t, \mathcal{R}(u))$ whenever $\phi(t, u)$ exists. A trajectory of (4) is said to be $\mathcal{R}$-symmetric if it is invariant under this symmetry.

Let us define a Poincaré section

$$\Pi = \{(0, y, z) : y, z \in \mathbb{R}\}$$

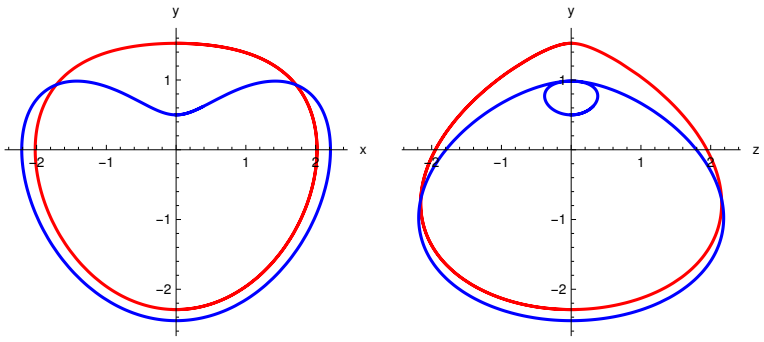

Figure 1: Projections of two observed $\mathcal{R}$-symmetric periodic orbits for the system (4) with the parameter $c = 1$.

and denote by

$$\mathcal{P}_c \colon \Pi \to \Pi \tag{5}$$

a family of Poincaré maps parametrized by $c > 0$. Note that we allow intersection of trajectories with the section Π in both directions. It is easy to see that $\mathcal{P}_c$ is reversible with respect to the involution $\mathcal{R}(y, z) = (y, -z)$, that is $\mathcal{R} \circ \mathcal{P} = \mathcal{P}^{-1} \circ \mathcal{R}$ — see [19]. In [59] an analytic proof of the existence of two R-symmetric periodic orbits for the system (4) is given — see Figure 1. Here we extend this result to a range of parameters.

Theorem 1. *For all parameter values $c \in C := [1 - 1/128, 1 + 1/128]$ the system (4) has at least two R-symmetric periodic solutions.*

Proof: In order to prove the existence of a family of symmetric period-two points for $\mathcal{P}_c$ in the set $\{0\} \times Y \times \{0\}$ parametrized by $c \in C$ it suffices to show that

- $\mathcal{P}_c$ is defined and continuous on $\{0\} \times Y \times \{0\}$ for $c \in C$ and
- $\pi_z \mathcal{P}_c(0, \min Y, 0)$ and $\pi_z \mathcal{P}_c(0, \max Y, 0)$ have opposite signs for all $c \in C$.

Then for each $c \in C$ there is a $y_c^* \in Y$ such that $\mathcal{P}_c(0, y_c^*, 0) = (0, \tilde{y}_c, 0)$ for some $\tilde{y}_c \in \mathbb{R}$ and the result follows from the reversibility of $\mathcal{P}_c$. The following program checks the above set of inequalities for two disjoint subintervals Y_1, Y_2 of positive semi-axis. Additionally, the program checks that $\pi_y \mathcal{P}_c(0 \times y \times 0) < 0$ for all $c \in C$ and $y \in Y_1 \cup Y_2$, which implies

$$\Big(\pi_y \mathcal{P}_c(\{0\} \times Y_1 \times \{0\}) \cup \pi_y \mathcal{P}_c(\{0\} \times Y_2 \times \{0\})\Big) \cap (Y_1 \cup Y_2) = \emptyset$$

and thus the two families of periodic points in Y_1 and Y_2 are different. The program executes within less than 1 second on a laptop-type computer. □

```
/* Proof of symmetric periodic orbits in the Michelson system. */
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
/**
 * @param c - parameter range
 * @param Y - interval on the y-axis
```

```
 * @param n - numer of grid elements in Y
 */
void validateSymPO(interval c, interval Y, int n){
  IMap vf("par:c;var:x,y,z;fun:y,z,c^2-y-0.5*x^2;");
  IOdeSolver solver(vf, 15);         // order of the solver
  ICoordinateSection section(3, 0); // Poincare section x=0
  IPoincareMap pm(solver, section);
  vf.setParameter("c",c);
  // We split Y into n subintervals and check that
  // the Poincare map is defined on (0,Y ,0). What the result is
  // is not crucial, as long as the image is in the y<0 halfplane.
  interval g = (Y.right()-Y.left())/n;
  for(int i=0;i<n;++i){
    C0HOTripletonSet s({0.,Y.left() + interval(i,i+1)*g,0.});
    assert( pm(s)[1] < 0.);
  }
  // Compute P(0,min(Y),0) and P(0,max(Y),0)
  C0HOTripletonSet s1(IVector({0.,Y.left(),0.}));
  C0HOTripletonSet s2(IVector({0.,Y.right(),0.}));
  IVector r1 = pm(s1), r2 = pm(s2);
  // Check that z-coordinate changes the sign
  std::cout << "validated? " << ( r1[2]*r2[2]<0 ) << ": "
            << r1[2] << ", " << r2[2] << std::endl;
}
int main(){
  std::cout.precision(4);
  interval I(-1,1);
  // Call the algorithm with two approximate periodic points.
  // Parameter c=1, validate orbits with accuracy 1e-13.
  validateSymPO(1., 1.5259617305036892  + 1e-13*I, 1);
  validateSymPO(1., 0.50002564853520548 + 1e-13*I, 1);
  // Repeat for parameter range c \in [1-1/128,1+1/128].
  validateSymPO(1+I/128, 1.5259617305036892  + 2e-1*I, 2);
  validateSymPO(1+I/128, 0.50002564853520548 + 2e-1*I, 4);
  return 0;
}
/* Ouptut (rounded to 4 digits, change of the signs is relevant):
validated? 1: [1.414e-13, 2.466e-13], [-2.48e-13, -1.444e-13]
validated? 1: [-4.212e-13, -2.477e-13], [2.331e-13, 4.036e-13]
validated? 1: [0.1046, 0.6294], [-0.6001, -0.1967]
validated? 1: [-1.314, -0.09417], [0.03929, 1.153] */
```

3.2. *Attractor in the Rössler system*

Simulation shows that for a wide range of parameter values the Rössler system [60]

$$x' = -(y+z), \qquad y' = x + by, \qquad z' = b + z(x-a) \tag{6}$$

possesses a chaotic attractor — see Fig. 2. To the best of our knowledge, the first proof that for classical parameter values there is a trapping region for the attractor was given in [42]. To present the details of the computer assisted proof, let us define a Poincaré section and the corresponding Poincaré map by

$$\begin{array}{lcl} \Pi & = & \{(0,y,z) : y,z \in \mathbb{R}, x' > 0\}, \\ \mathcal{P}_{a,b} & : & \Pi \to \Pi. \end{array} \tag{7}$$

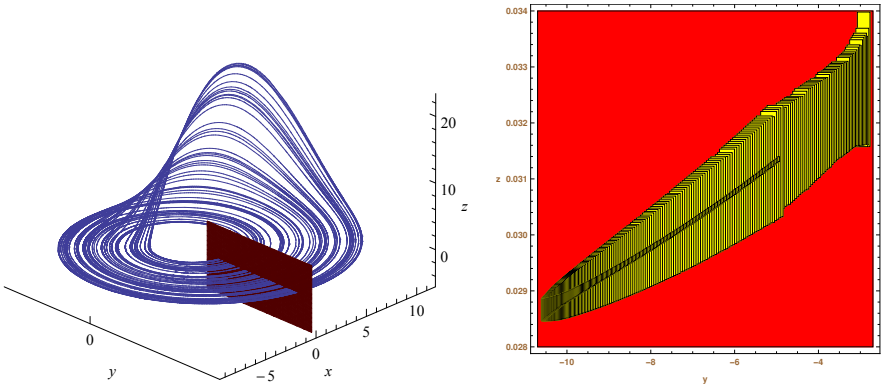

Figure 2: (Left) Observed chaotic attractor of the system (6) with $b = 0.2$ and $a = 5.7$. (Right) Plot of trapping region W and rigorous enclosure of $\mathcal{P}_{a,b}(W)$ for the system with parameters $b = 0.2$ and $a = 5.7$.

Theorem 2 ([55]). *For $a = 5.7$ and $b = 0.2$ the set*

$$W = Y \times Z := [-10.7, -2.7] \times [0.028, 0.034] \tag{8}$$

is forward invariant for $\mathcal{P}_{a,b}$. In particular, it contains compact, connected invariant set $\mathcal{A} = \bigcap_{n>0} \mathcal{P}^n_{a,b}(W)$.

Proof: First observe, that $x' = -(y+z) \geq 2.7 - 0.034 > 0$ for all $(y,z) \in W$ and hence $W \subset \Pi$. Inclusion $\mathcal{P}_{a,b}(W) \subset W$ can be checked in direct computation. The set W is subdivided uniformly (for simplicity of the program) $W \subset \bigcup_{i=1}^{N} Y_i \times Z$ and then inclusion $\mathcal{P}_{a,b}(Y_i \times Z) \subset W$ is checked for all $i = 1, \ldots, N$. The program executes within less than 1 second on a laptop-type computer. □

```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
int main(){
  IMap vf("par:a,b;var:x,y,z;fun:-(y+z),x+b*y,b+z*(x-a);");
  vf.setParameter("a",interval(57)/10);
  vf.setParameter("b",interval(2)/10);
  IOdeSolver solver(vf, 20);
  ICoordinateSection section(3, 0);   // section is given by x = 0 and
  IPoincareMap pm(solver, section, poincare::MinusPlus);  // x'> 0
  // Coordinates of the trapping region W = Y\times Z
  interval Y = interval(-107,-27)/10; // Y=[-10.7,-2.7]
  interval Z = interval(28,34)/1000;  // Z=[0.028,0.034]
  // Subdivide uniformly Y onto N subintervals
  const int N = 150;
  bool result = true;
  for (int i = 0; i < N and result; ++i) {
    IVector Wi ({0., Y.left() + diam(Y)*interval(i,i+1)/N, Z});
    C0HOTripletonSet s(Wi);
    IVector u = pm(s);
    result = result and subset(u[1],Y) and subset (u[2],Z);
  }
  std::cout << "Trapping region validated? : " << result << std::endl;
  return 0;
}
```

In the next section we will show that the attractor $\mathcal{A} = \bigcap_{n>0} \mathcal{P}^n_{a,b}(W)$ contains a chaotic and uniformly hyperbolic invariant set.

4. C^1 solver and its applications

By a rigorous C^1-algorithm we mean an algorithm capable of computing bounds for the following system of ODEs

$$x'(t) = f(t, x(t)), \qquad V'(t) = D_x f(t, x(t)) \cdot V(t)$$

for some initial conditions $x(0) \in [x_0] \subset \mathbb{R}^n$ and $V(0) \in [V_0] \subset \mathbb{R}^{n\times n}$. In principle, any C^0 solver is capable of doing this task. Taking into account special structure of this system of equations, one can design an algorithm of complexity $O(n^3)$ which is much faster than the direct application of the C^0 solver which has complexity $O(n^6)$. Such an algorithm was proposed in [15] and later improved in [55]. Both versions are very powerful tools for studying hyperbolic-like properties of dynamical systems, such as verification of periodic orbits and their stability [23, 24, 61], connecting orbits [20, 62] and hyperbolic attractors [25]. Here we present a few short, yet non-trivial examples:

- verification of the existence of a solution to a boundary value problem,
- verification of the existence of hyperbolic periodic solutions with very high localization accuracy,
- verification of the existence of a hyperbolic chaotic set.

4.1. Boundary value problem.

In this section we show that C^1 algorithms can be used to solve boundary value problems. As an example, we reproduce the result by Nakao [63].

Theorem 3 ([63]). *The equation*

$$x'' = -0.1x - 0.1x^3 - 0.4464\cos t \tag{9}$$

has a solution satisfying $x'(0) = x'(2\pi) = 0$.

Proof: The proof in [63] is also computer-assisted but relies on solving a zero-finding problem in some infinite-dimensional functional space. Here we propose a direct approach, as we have tools capable to compute derivatives of ODEs with respect to initial conditions. Denote by $\varphi(t, t_0, x_0, x_0') = (\varphi_x(t, t_0, x_0, x_0'), \varphi_{\dot{x}}(t, t_0, x_0, x_0'))$ a solution of the initial value problem $x(t_0) = x_0, x'(t_0) = x_0'$ for (9). It is easy to see that the zeroes of

$$F(x) = \varphi_{\dot{x}}(2\pi, 0, x, 0) = 0$$

correspond to the solutions of the boundary value problem we are looking for. The following program checks, by means of the interval Newton operator (10), that the function F has a zero at some x_* with $|x_* + 0.5072| \leq 10^{-4}$. The program executes within less than 1 second on a laptop-type computer. □

```
/** Example of solving BVP: x'(0)=x'(2pi)=0 **/
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;
int main(){
  IMap f("par:a;time:t;var:x,dx;fun:dx,-x*(1+x^2)/10 + a*cos(t);");
  f.setParameter("a",interval(4464)/interval(10000));
  IOdeSolver solver(f,20);  // ODE integrator
  ITimeMap tm(solver);      // class for long time integration

  IVector u0({-0.5072,0.});
  IVector r({interval(-1e-4,1e-4),0.});
  C0HOTripletonSet s0(u0);
  C1HORect2Set s(u0,r);  // represent set s = u0 + Id*r
  // integrate set and variational equation until T=2*pi
  IVector y = tm(2.*interval::pi(),s0); // C0 computation
  tm(2.*interval::pi(),s);              // C1 computation
  // solve equation F(r1) := proj_{x'}(phi(2pi,u0+(r1,0))) = 0
  interval N = - y[1]/((IMatrix)s)[1][0];
  cout << "(N,r1)=(" << N << "," << r[0] << ")"<< endl;
  cout << "subset(N,r)? = " << boolalpha << subset(N,r[0]) << endl;
  return 0;
}
/* Output (rounded to six digits):
(N,r1)=([-2.68037e-05, -2.67903e-05],[-0.0001, 0.0001])
subset(N,r)? = true
*/
```

4.2. High localization accuracy bounds for periodic orbits in the Rössler system.

In Section 3.2 we have proved that system (6) has an attractor. Here we will prove the existence of three periodic orbits on this attractor with very high localization accuracy.

Theorem 4. *Put*

$$\begin{aligned} u_1 &= (-8.380941742829876287348763043 1, 0.029590060630667102951494027735),\\ u_2 &= (-5.424073822665204351567302546 3, 0.031081210807876445187367377796),\\ u_3 &= (-6.233158628537974951507647941 1, 0.030640111658160569478006226700). \end{aligned}$$

There exist three hyperbolic periodic points $u_m^* = u_m + r_m$, $m = 1,2,3$ for the Poincaré map (7) of period $1,2,3$, respectively satisfying $\|r_m\|_1 \leq 10^{-28}$ and the coordinates of r_m are known with accuracy 10^{-54}.

Proof: The proof relies on properties of the interval Newton operator [64]

$$N(f, x_0, X) = x_0 - [Df(X)]_I^{-1} f(x_0), \tag{10}$$

where by $[A]_I$ we mean an interval hull of the matrix A. It is well known [64], that if X is a convex set, $x_0 \in X$ and $N(f, x_0, X) \subset X$ then the mapping f has a unique zero in the set X. Moreover, this zero belongs to $N(f, x_0, X)$. The following program validates the existence of three periodic solutions by means of the interval Newton operator applied to the function $f = \mathcal{P}_{a,b}^m - \mathrm{Id}$ for $m = 1,2,3$, depending on the orbit. It also prints the largest diameter of the components of $N(\mathcal{P}_{a,b}^m - \mathrm{Id}, u_m, u_m + R_m) - u_m$, where $R_m = 10^{-28} \cdot [-1,1]^2$, which in each case $m = 1,2,3$ is

less than 10^{-54}. This shows, that coordinates of $r_m \in N(\mathcal{P}^m_{a,b} - \mathrm{Id}, u_m, u_m + R_m) - u_m$ are known with accuracy 10^{-54}. We would like to emphasize, that it is very easy to obtain much higher localization accuracy by either providing more digits for initial points or by iterating the interval Newton operator. Finally, the program computes bounds on the eigenvalues of the Poincaré map at periodic points which proves that they are all of saddle type. From these bounds it is also clear, that the three orbits are different.

The program uses high-precision version of the C^1 ODE solver from the CAPD::DynSys library to obtain tiny bounds on $\mathcal{P}_{a,b}$ and its derivative. The program executes within 20 seconds on a laptop-type computer. □

```
#include <iostream>
#include "capd/mpcapdlib.h"
using namespace capd;
using namespace std;
/**
 * @param (y,z) - approximate periodic point
 * @param e - radius of the set centred at (y,z)
 * @param n - period
 */
void po(MpFloat y, MpFloat z, int n, double e=1e-28){
 MpIMap vf("par:a,b;var:x,y,z;fun:-(y+z),x+b*y,b+z*(x-a);");
 vf.setParameter("a",MpInterval(57)/10);
 vf.setParameter("b",MpInterval(2)/10);
 MpIOdeSolver solver(vf,80);           // ODE integrator of order 80
 MpICoordinateSection section(3,0.);   // the section is x=0
 MpIPoincareMap pm(solver,section, poincare::MinusPlus);
 // Approximate periodic point and a ball around it.
 MpIVector u0({MpInterval(0.),y,z});
 MpIVector r({MpInterval(0.),MpInterval(-e,e),MpInterval(-e,e)});
 MpC0TripletonSet s0(u0);
 // Compute P^n(u0)-u0 and project it onto (y,z)
 MpIVector fu0( 2, (pm(s0,n) - u0).begin() + 1 );
 MpC1Rect2Set s1(u0,r); // represent s1 =  u0 + Id*r
 MpIMatrix Dphi(3,3);
 // compute DP^n(u0+r)
 MpIVector u = pm(s1,Dphi,n);
 MpIMatrix DP = pm.computeDP(u,Dphi);
 // projection of DP^n(u0+r)-Id onto 2D subspace
 MpIMatrix M({{DP[1][1]-1.,DP[1][2]},{DP[2][1],DP[2][2]-1.}});
 // enclose -(DP^n(u0+r)-Id)^{-1}*(P^n(u0)-u0)
 MpIVector N = - matrixAlgorithms::gauss(M,fu0);
 cout << boolalpha << "(validated?, accuracy) = ("
      << subset(N,MpIVector(2,r.begin()+1)) << ", " << maxWidth(N) << ")";
 // compute bound on eigenvalues using an explicit formula
 MpInterval t = sqrt(4*DP[2][1]*DP[1][2] + sqr(DP[1][1]-DP[2][2]));
 cout << "\neigenvalues=(" << 0.5*(DP[1][1]+DP[2][2]-t)
      << "," << 0.5*(DP[1][1]+DP[2][2]+t) << ")" << endl;
}
int main(){
 MpFloat::setDefaultPrecision(200); // 200 manitissa bits
 po("-8.380941742829876287348763043 1","-.029590060630667102951494027735",1);
 po("-5.424073822665204351567302546 3","-.031081210807876445187367377796",2);
 po("-6.233158628537974951507647941 1","-.030640111658160569478006226700",3);
}
/* Output (rounded to 6 digits) of the program:
(validated?, accuracy) = (true, 8.10612e-56 )
eigenvalues=([-2.40396 ,-2.40395 ],[-1.28211e-14 ,-1.28210e-14 ])
```

```
(validated?, accuracy) = (true, 2.03877e-55 )
eigenvalues=([-3.51201 ,-3.51200 ],[-1.24264e-26 ,1.20278e-26 ])
(validated?, accuracy) = (true, 6.49223e-55 )
eigenvalues=([-2.34193 ,-2.34192 ],[-2.73947e-26 ,2.73947e-26 ])*/
```

4.3. Uniformly hyperbolic chaotic invariant set

In the last example of this section we would like to recall the result from [55] about the existence of a uniformly hyperbolic and chaotic invariant set in the Rössler system (6). From Theorem 2 we know that the set $W = Y \times Z$ defined by (8) is a trapping region for the Poincaré map (7) of the Rössler system (6) for parameters values $a = 5.7, b = 0.2$. This implies the existence of a connected, compact invariant set $\mathcal{A} = \bigcap_{n>0} \mathcal{P}^n_{a,b}(W)$. Here we present a proof that this attractor is non-trivial.

Theorem 5 ([55]). *Let $l_M = -8.4$, $r_M = -7.6$, $l_N = -5.7$, $r_N = -4.6$ and define two subsets of W,*

$$M = [l_M, r_M] \times Z \quad \text{and} \quad N = [l_N, r_N] \times Z.$$

Fix $a = 5.7$, $b = 0.2$ and denote $\mathcal{P} = \mathcal{P}_{a,b}$. Then the maximal invariant set for $\mathcal{P}^2$ in $N \cup M$, denoted by $\mathcal{H} = \mathrm{inv}(\mathcal{P}^2, N \cup M) \subset \mathcal{A}$, is uniformly hyperbolic; in particular it is robust under perturbations of the system. The dynamics of $\mathcal{P}^2$ on $\mathcal{H}$ is chaotic in the sense that $\mathcal{P}^2|_{\mathcal{H}}$ is conjugated to the Bernoulli shift on two symbols.

Proof: The proof relies on some partial results from [11, 40, 25]. Semiconjugacy of $\mathcal{P}^2|_{\mathcal{H}}$ to the Bernoulli shift is proved by means of the method of covering relations [11]. It is sufficient to check the following geometric conditions

$$\begin{array}{lcll} \pi_y \mathcal{P}^2_{a,b}(y,z) & < & l_M & \text{for } (y,z) \in \{l_M\} \times Z, \\ \pi_y \mathcal{P}^2(y,z) & > & r_N & \text{for } (y,z) \in \{r_M\} \times Z, \\ \pi_y \mathcal{P}^2(y,z) & < & l_M & \text{for } (y,z) \in \{r_N\} \times Z, \\ \pi_y \mathcal{P}^2(y,z) & > & r_N & \text{for } (y,z) \in \{l_N\} \times Z, \end{array} \tag{11}$$

where π_y denotes the projection onto y coordinate. Rigorous bounds on $\mathcal{P}^2(\{l_M\}\times Z)$, $\mathcal{P}^2(\{r_M\}\times Z)$, $\mathcal{P}^2(\{l_N\} \times Z)$ and $\mathcal{P}^2(\{r_N\} \times Z)$, returned by our routine, are shown in Fig. 3.

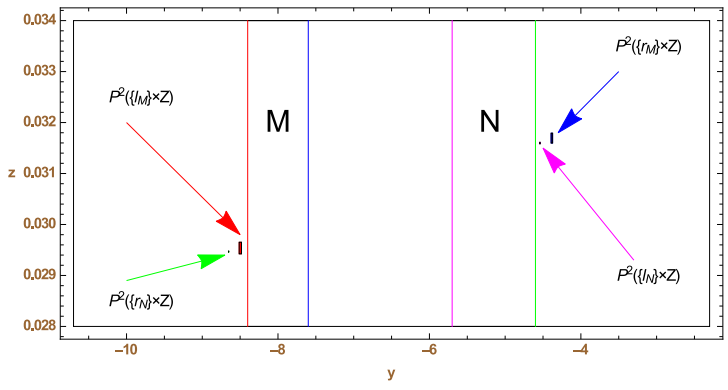

Figure 3: The sets M and N and rigorous enclosures of the images of their exit edges — see (11).

Hyperbolicity of $\mathcal{H}$ is proved by means of the cone condition introduced in [40]. Let Q be a diagonal matrix $Q = \mathrm{Diag}(\lambda, \mu)$ with arbitrary coefficients satisfying $\lambda > 0$ and $\mu < 0$. It was shown in [25] that if for all $(y, z) \in N \cup M$ the matrix

$$DP^2(y,z)^T \cdot Q \cdot DP^2(y,z) - Q \tag{12}$$

is positive definite, then the maximal invariant set for P^2 in $N \cup M$ is uniformly hyperbolic. The following program checks all necessary conditions with constants $\lambda = 1$ and $\mu = -100$. Some subdivision of sets was necessary to obtain sharp bounds on the derivative of $\mathcal{P}^2$. The program executes within less than 2 seconds on a laptop-type computer. □

```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;

// z-coordinate of the trapping region
interval Z = interval(28,34)/1000;  // Z=[0.028,0.034]
// y-coordinates of sets M and N
interval My=interval(-84,-76)/10, Ny=interval(-57,-46)/10;

/// This routine checks cone-condition on the set Y\times Z
/// The interval Y is split into g pieces
bool checkCC(IPoincareMap& pm, interval Y, int g) {
  bool res = true;
  interval p = (Y.right()-Y.left())/g;
  IMatrix Dphi(3,3);
  // define quadratic form Q
  IMatrix Q({{0.,0.,0.},{0.,1,0.},{0.,0.,-100}});
  for (int i = 0; i < g and res; ++i) {
    // compute derivative of P^2 on a grid element
    C1Rect2Set s({0.,Y.left()+interval(i,i+1)*p,Z});
    interval returnTime; // not used, by required by syntax below
    IVector y = pm(s, Dphi, returnTime, 2);
    IMatrix DP = pm.computeDP(y,Dphi);
    // check positive definitness by Sylvester criterion
    DP = transpose(DP)*Q*DP - Q;
    res = res and DP[1][1]>0 and (DP[1][1]*DP[2][2]-sqr(DP[1][2]))>0;
  }
  return res;
}
int main(){
  IMap vf("par:a,b;var:x,y,z;fun:-(y+z),x+b*y,b+z*(x-a);");
  vf.setParameter("a",interval(57)/10);
  vf.setParameter("b",interval(2)/10);
  IOdeSolver solver(vf, 20);
  ICoordinateSection section(3, 0); // section x=0, x'>0
  IPoincareMap pm(solver, section, poincare::MinusPlus);

  // Inequalities for the covering relations -- see (11).
  cout << boolalpha;
  C0HOTripletonSet LM( IVector({0.,My.left(), Z}) );
  C0HOTripletonSet RM( IVector({0.,My.right(),Z}) );
  C0HOTripletonSet LN( IVector({0.,Ny.left(), Z}) );
  C0HOTripletonSet RN( IVector({0.,Ny.right(),Z}) );
  cout << "pi_y P^2(LM) < lM? " << ( pm(LM,2)[1] < My ) << endl;
  cout << "pi_y P^2(RM) > rN? " << ( pm(RM,2)[1] > Ny ) << endl;
  cout << "pi_y P^2(LN) > rN? " << ( pm(LN,2)[1] > Ny ) << endl;
```

```
  cout << "pi_y P^2(RN) < lM? " << ( pm(RN,2)[1] < My ) << endl;
  // Check cone conditions.
  cout << "Cone condition on M? " << checkCC(pm,My,80) << endl;
  cout << "Cone condition on N? " << checkCC(pm,Ny,40) << endl;
}
```

5. C^r solver and its application

The CAPD::DynSys library offers a rigorous solver for higher order variational equations, that is

$$\begin{aligned}
\frac{d}{dt}\phi(t,x) &= f(t,x(t)),\\
\frac{d}{dt}D_x\phi(t,x) &= D_x f(t,x(t))\cdot D_x\phi(t,x),\\
\frac{d}{dt}D_a\phi(t,x) &= D_x f(t,x(t))D_a\phi(t,x) + h.o.t,
\end{aligned}$$

where D_a is the partial derivative operator with respect to a multiindex a and $h.o.t.$ stands for higher order terms not written explicitly. Higher order derivatives are extremely useful in studying global and local bifurcations [40, 41, 42, 43] as well as non-linear stability of elliptic periodic solutions [45, 6]. Here we present one short example of application of C^r solver to study KAM tori near an elliptic periodic orbit in the Michelson system (4). The existence of a wide branch of such orbits parametrized by c was proved in [6]. Here we give a proof for just one parameter value close to 1 : 4 resonance – see Fig. 4. The following theorem is a special case of the result from [6].

Theorem 6. *For the parameter value $c = 0.226$ the Poincaré map (5) has a symmetric period-two point $u_* = (0, y_*, 0)$, $|y_* - 0.43407644067709| \leq 2\cdot 10^{-14}$, which is stable. That is, any neighbourhood $U \subset \mathbb{R}^3$ of the periodic trajectory $O(u_*)$ contains a 2D invariant torus surrounding the orbit $O(u_*)$ and separating the phase space.*

Proof: The proof utilizes the classical result by Siegel and Moser [65] with computational tools. First, we have to check that the periodic orbit indeed exists. The interval Newton operator (10) applied to the scalar equation

$$F(y) = \pi_z(\mathcal{P}_c(0,y,0)) = 0$$

validates the existence of a zero y_* of F satisfying $|y_* - 0.43407644067709| \leq 2\cdot 10^{-14}$.

Then we compute truncated Birkhoff normal form at the periodic orbit and we check the twist condition. If a certain coefficient in the normal form does not vanish then the existence of KAM tori and stability of periodic orbit follow from the theorem by Siegel and Moser. The following program executes within less than 1 second on a laptop-type computer. □

```
/** Existence of invariant curves around an elliptic PO **/
#include <iostream>
#include "capd/capdlib.h"
#include "capd/normalForms/planarMaps.hpp"
using namespace capd;
using namespace std;
int main(){
  cout.precision(17);
```

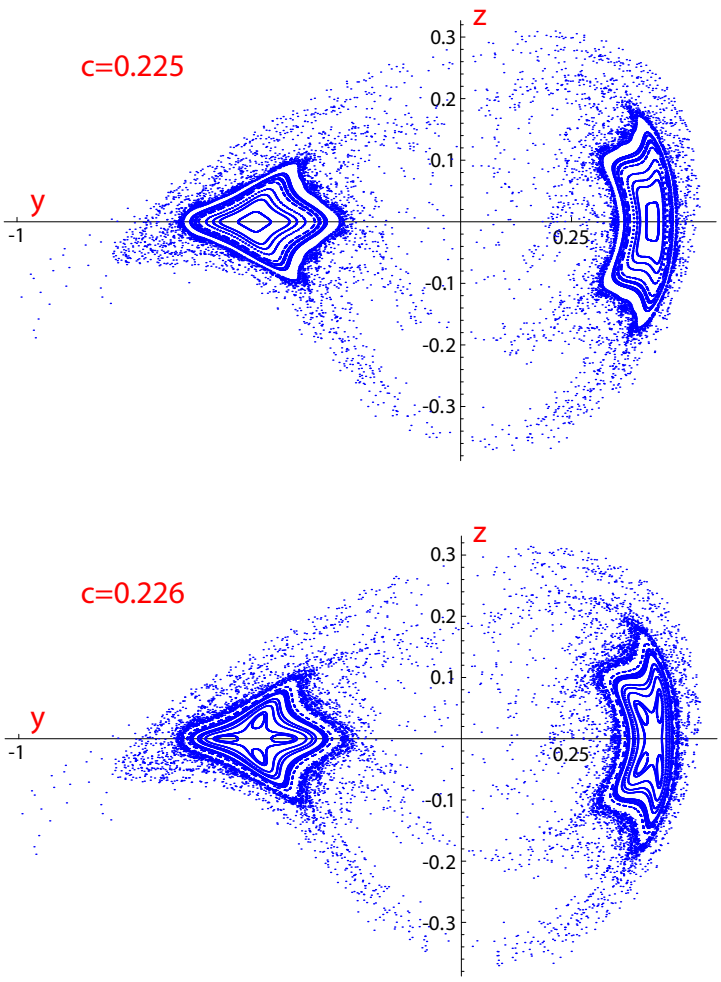

Figure 4: Phase portrait of the Poincaré map (5) of the Michelson system (4). It is an evidence that a family of elliptic periodic orbits crosses $1:4$ resonance when the parameter varies in the interval $c \in [0.225, 0.226]$. The existence of such period quadrupling bifurcation was proved in [42].

```
  IMap vf("par:c;var:x,y,z;fun:y,z,c^2-y-0.5*x^2;",3);
  vf.setParameter("c",interval(226)/1000);
  ICnOdeSolver solver(vf,20);
  ICoordinateSection section(3,0); // x=0
  ICnPoincareMap pm(solver,section);
  IMatrix DP(3,3);

  // Validate existence of a periodic point by the Newton method
  IVector u0({0,.43407644067709,0.});
  IVector r = 1e-12*interval(-1,1)*IVector{0.,1,0.};
  C0TripletonSet s0(u0);
  IVector y0 =  pm(s0);
  C1Rect2Set s(u0,r); // represent s = u0 + Id*r
  IVector y = pm(s,DP);
  DP = pm.computeDP(y,DP);
  interval N = - y0[2]/DP[2][1];
  cout << "subset(N,r)? = " << boolalpha << subset(N,r[1])
       << ", N = " << N << endl;
  // Integrate 3rd order variational equations over the full period 2
  CnRect2Set S(u0+N,3);
  IJet jet(3,3,3);
  y = pm(S,jet,2);
  jet = pm.computeDP(jet);
  // project onto 2-dim section
  IJet P(2,2,3);
  for(int j=0;j<=3;++j)
    for(int c=0;c<=j;++c){
      Multiindex m1({c,j-c}), m2({0,c,j-c});
      for(int i=0;i<2;++i)
        P(i,m1) = jet(i+1,m2);
    }
  // Coefficient of the Birkhoff normal form should not vanish - see [65]
  std::cout << "twist? " <<
    normalForms::computePlanarEllipticNormalForm(P)[1].real();
}
/* Output:
subset(N,r)? = true, N = [-1.859388640094055e-15, 1.0194263423055196e-14]
twist? [15.406918970206604, 15.406919005718061]
*/
```

6. Summary

In this article we described a basic interface of the CAPD::DynSys library. The C++ source code of the CAPD::DynSys library consists of over 120 000 lines and thus it is clear that presenting all implemented features and details of algorithms in one article is impossible. We showed, however, that the library is a powerful tool for rigorous numerical analysis of dynamical systems by examining several non-trivial examples.

We would like to mention, that the library provides support for integration of differential inclusions [17] and algorithms for rigorous integration of dissipative PDEs [66]. In the nearest future a C^1 algorithm for PDEs, constrained $C^0 - C^1$ algorithms for ODEs (that is taking into account constraints of the system, like Hamiltonians, measure preservation) and for delay differential equations should be added.

References

References

[1] N. S. Nedialkov, VNODE-LP: A validated solver for initial value problems in ordinary differential equations, Tech. Rep. Technical Report CAS-06-06-NN (2006).

[2] M. Kashiwagi, kv -a c++ library for verified numerical computation (2019).
URL `http://verifiedby.me/kv/`

[3] A. Rauh, M. Brill, C. Günther, A novel interval arithmetic approach for solving differential-algebraic equations with ValEncIA-IVP, Int. J. Appl. Math. Comput. Sci. 19 (3) (2009) 381–397. doi:10.2478/v10006-009-0032-4.

[4] M. Berz, K. Makino, New methods for high-dimensional verified quadrature, Reliable Computing 5 (1) (1999) 13–22. doi:10.1023/A:1026437523641.

[5] CAPD, Computer Assisted Proofs in Dynamics, a package for rigorous numerics, `http://capd.ii.uj.edu.pl`.

[6] D. Wilczak, R. Barrio, Systematic computer-assisted proof of branches of stable elliptic periodic orbits and surrounding invariant tori, SIAM Journal on Applied Dynamical Systems 16 (3) (2017) 1618–1649.

[7] K. Mischaikow, M. Mrozek, Chaos in the Lorenz equations: a computer-assisted proof, Bull. Amer. Math. Soc. (N.S.) 32 (1) (1995) 66–72.

[8] K. Mischaikow, M. Mrozek, Chaos in the Lorenz equations: a computer-assisted proof. Part II: Details, Mathematics of Computations (67) (1998) 1023–1046.

[9] K. Mischaikow, M. Mrozek, A. Szymczak, Chaos in the Lorenz equations: a computer-assisted proof. Part III: the classical parameter values, Journal of Differential Equations (169) (2001) 17–56.

[10] M. Juda, M. Mrozek, CAPD::RedHom v2 - homology software based on reduction algorithms, in: Mathematical Software – ICMS 2014, Hong, Hoon, Yap, Chee K. (Eds.), Vol. 8592 of Lecture Notes in Computer Science, 2014, pp. 160–166.

[11] P. Zgliczyński, Computer assisted proof of chaos in the Rössler equations and in the Hénon map, Nonlinearity 10 (1) (1997) 243–252.

[12] M. Żelawski, Rigorous numerical approach to isolation in dynamical systems on the example of the Kuramoto-Sivashinsky equation, Reliable Computing 5 (2) (1999) 113–129.

[13] P. Pilarczyk, Computer assisted method for proving existence of periodic orbits, Topol. Methods Nonlinear Anal. 13 (2) (1999) 365–377.

[14] R. J. Lohner, Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems, in: Computational ordinary differential equations (London, 1989), Vol. 39 of Inst. Math. Appl. Conf. Ser. New Ser., Oxford Univ. Press, New York, 1992, pp. 425–435.

[15] P. Zgliczyński, C^1-Lohner algorithm, Foundations of Computational Mathematics 2 (4) (2002) 429–465. doi:10.1007/s102080010025.

[16] D. Wilczak, P. Zgliczyński, C^r-Lohner algorithm, Schedae Informaticae 20 (2011) 9–46.

[17] T. Kapela, P. Zgliczyński, A Lohner-type algorithm for control systems and ordinary differential inclusions, Discrete & Continuous Dynamical Systems - B 11 (2009) 365–385. doi:10.3934/dcdsb.2009.11.365.

[18] G. Arioli, P. Zgliczyński, Symbolic dynamics for the Hénon–Heiles hamiltonian on the critical energy level, J. Diff. Eq 171 (2001) 173–202.

[19] D. Wilczak, Chaos in the Kuramoto-Sivashinsky equations–a computer-assisted proof, Journal of Differential Equations 194 (2) (2003) 433–459. doi:http://dx.doi.org/10.1016/S0022-0396(03)00104-9.

[20] D. Wilczak, P. Zgliczyński, Heteroclinic connections between periodic orbits in planar restricted circular three-body problem – a computer assisted proof, Comm. Math. Phys. 234(1) 37–75.

[21] D. Wilczak, P. Zgliczyński, Heteroclinic connections between periodic orbits in planar restricted circular three body problem. part II, Comm. Math. Phys. 259 (2005) 561–576.

[22] D. Wilczak, The existence of Shilnikov homoclinic orbits in the Michelson system: A computer assisted proof, Foundations of Computational Mathematics 6 (4) (2006) 495–535. doi:10.1007/s10208-005-0201-2.

[23] T. Kapela, P. Zgliczyński, The existence of simple choreographies for the N-body problem – a computer-assisted proof, Nonlinearity 16 (6) (2003) 1899.

[24] T. Kapela, C. Simó, Computer assisted proofs for nonsymmetric planar choreographies and for stability of the eight, Nonlinearity 20 (5) (2007) 1241.

[25] D. Wilczak, Uniformly hyperbolic attractor of the Smale–Williams type for a Poincarḿap in the Kuznetsov system, SIAM Journal on Applied Dynamical Systems 9 (4) (2010) 1263–1283. doi:10.1137/100795176.

[26] M. J. Capiński, Computer assisted existence proofs of Lyapunov orbits at L2 and transversal intersections of invariant manifolds in the Jupiter-Sun PCR3BP, SIAM J. Applied Dynamical Systems 11 (4) (2012) 1723–1753.

[27] M. J. Capiński, A. Wasieczko-Zając, Geometric proof of strong stable/unstable manifolds with application to the restricted three body problem, Top. Meth. Non. Anal. 46 (1) (2015) 363—-399.

[28] J. Galante, V. Kaloshin, Destruction of invariant curves in the restricted circular planar three body problem using comparison of action, Duke Math. J. 159 (2) (2011) 275–327.
[29] Z. Galias, W. Tucker, Rigorous integration of smooth vector fields around spiral saddles with an application to the cubic Chua's attractor, Journal of Differential Equations 266 (5) (2019) 2408–2434. doi:10.1016/j.jde.2018.08.035.
[30] Z. Galias, W. Tucker, Rigorous study of short periodic orbits for the Lorenz system, in: Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on, 2008, pp. 764–767. doi:10.1109/ISCAS.2008.4541530.
[31] Z. Galias, W. Tucker, Validated study of the existence of short cycles for chaotic systems using symbolic dynamics and interval tools, Int. J. Bifurcation Chaos 21 (2) (2011) 551–563. doi:10.1142/S021812741102857X.
[32] M. Fenucci, G. F. Gronchi, On the stability of periodic n-body motions with the symmetry of platonic polyhedra, Nonlinearity 31 (11) (2018) 4935–4954. doi:10.1088/1361-6544/aad644.
[33] T. Miyaji, H. Okamoto, A computer-assisted proof of existence of a periodic solution, Proc. Japan Acad. Ser. A Math. Sci. 90 (10) (2014) 139–144.
[34] T. Miyaji, H. Okamoto, Existence proof of unimodal solutions of the Proudman–Johnson equation via interval analysis, Japan Journal of Industrial and Applied Mathematics 36 (1) (2019) 287–298. doi:10.1007/s13160-018-00339-x.
URL https://doi.org/10.1007/s13160-018-00339-x
[35] K. Matsue, Rigorous numerics of finite-time singularities in dynamical systems - methodology and applications, prepint.
[36] S. Rohou, L. Jaulin, L. Mihaylova, F. Le Bars, S. M. Veres, Reliable non-linear state estimation involving time uncertainties, Automatica 93 (2018) 379 – 388. doi:https://doi.org/10.1016/j.automatica.2018.03.074.
URL http://www.sciencedirect.com/science/article/pii/S0005109818301699
[37] S. Rohou, L. Jaulin, L. Mihaylova, F. Le Bars, S. M. Veres, Guaranteed computation of robot trajectories, Robotics and Autonomous Systems 93 (2017) 76 – 84. doi:https://doi.org/10.1016/j.robot.2017.03.020.
URL http://www.sciencedirect.com/science/article/pii/S0921889016304006
[38] J. Cyranka, M. A. Islam, G. Byrne, P. Jones, S. A. Smolka, R. Grosu, Lagrangian reachabililty, in: International Conference on Computer Aided Verification, Springer, 2017, pp. 379–400.
[39] J. Cyranka, T. Wanner, Computer-assisted proof of heteroclinic connections in the one-dimensional Ohta–Kawasaki model, SIAM Journal on Applied Dynamical Systems 17 (1) (2018) 694–731. doi:10.1137/17M111938X.
[40] H. Kokubu, D. Wilczak, P. Zgliczyński, Rigorous verification of cocoon bifurcations in the Michelson system, Nonlinearity 20 (9) (2007) 2147–2174.
[41] D. Wilczak, P. Zgliczyński, Period doubling in the Rössler system – a computer assisted proof, Foundations of Computational Mathematics 9 (5) (2009) 611–649. doi:10.1007/s10208-009-9040-x.
[42] I. Walawska, D. Wilczak, Validated numerics for period-tupling and touch-and-go bifurcations of symmetric periodic orbits in reversible systems, Communications in Nonlinear Science and Numerical Simulation 74 (2019) 30 – 54. doi:https://doi.org/10.1016/j.cnsns.2019.03.005.
URL http://www.sciencedirect.com/science/article/pii/S1007570419300735
[43] D. Wilczak, P. Zgliczyński, Computer assisted proof of the existence of homoclinic tangency for the Hénon map and for the forced damped pendulum, SIAM Journal on Applied Dynamical Systems 8 (4) (2009) 1632–1663. doi:10.1137/090759975.
[44] M. J. Capiński, M. Gidea, Arnold diffusion, quantitative estimates and stochastic behavior in the three-body problem, preprint.
[45] T. Kapela, C. Simó, Rigorous KAM results around arbitrary periodic orbits for hamiltonian systems, Nonlinearity 30 (3) (2017) 965–986.
[46] D. Wilczak, R. Barrio, Distribution of stable islands within chaotic areas in the non-hyperbolic and hyperbolic regimes in the Hénon-Heiles system, Nonlinear Dynamics (to appear).
[47] M. J. Capiński, P. Zgliczyński, Beyond the Melnikov method: a computer assisted approach, J. Differential Equations 262 (1) (2017) 365–417. doi:10.1016/j.jde.2016.09.032.
URL https://doi.org/10.1016/j.jde.2016.09.032
[48] M. J. Capiński, P. Zgliczyński, Beyond the Melnikov method II: Multidimensional setting, Journal of Differential Equations 265 (9) (2018) 3988 – 4015. doi:https://doi.org/10.1016/j.jde.2018.05.028.
URL http://www.sciencedirect.com/science/article/pii/S0022039618303097
[49] R. E. Moore, Interval analysis, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1966.
[50] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, P. Zimmermann, MPFR: A Multiple-Precision binary Floating-point library with correct Rounding, ACM Trans. Math. Softw. 33 (2) (2007) 13–es. doi:10.1145/1236463.1236468.
URL https://doi.org/10.1145/1236463.1236468
[51] L. B. Rall, G. F. Corliss, An introduction to automatic differentiation, in: Computational differentiation (Santa Fe, NM, 1996), SIAM, Philadelphia, PA, 1996, pp. 1–18.
[52] M. Mrozek, P. Zgliczyński, Set arithmetic and the enclosing problem in dynamics, Ann. Polon. Math. 74 (2000) 237–259.

[53] T. Miyaji, P. Pilarczyk, M. Gameiro, H. Kokubu, K. Mischaikow, A study of rigorous ode integrators for multi-scale set-oriented computations, Applied Numerical Mathematics 107 (2016) 34 – 47. doi:https://doi.org/10.1016/j.apnum.2016.04.005.
URL http://www.sciencedirect.com/science/article/pii/S0168927416300435

[54] N. S. Nedialkov, K. R. Jackson, An interval Hermite–Obreschkoff method for computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation, Developments in Reliable Computing 5 (1998) 289–310.

[55] I. Walawska, D. Wilczak, An implicit algorithm for validated enclosures of the solutions to variational equations for ODEs, Applied Mathematics and Computation 291 (2016) 303–322.

[56] E. Lorenz, Deterministic nonperiodic flow, J. Atmospheric Sci. 20 (1963) 130–141.

[57] Z. Galias, P. Zgliczyński, Computer assisted proof of chaos in the Lorenz equations, Phys. D 115 (3-4) (1998) 165–188.

[58] D. Michelson, Steady solutions of the Kuramoto-Sivashinsky equation, Physica D: Nonlinear Phenomena 19 (1) (1986) 89–111. doi:http://dx.doi.org/10.1016/0167-2789(86)90055-2.

[59] W. C. Troy, The existence of steady solutions of the Kuramoto–Sivashinsky equation, Journal of Differential Equations 82 (2) (1989) 269 – 313. doi:https://doi.org/10.1016/0022-0396(89)90134-4.
URL http://www.sciencedirect.com/science/article/pii/0022039689901344

[60] O. E. Rössler, An equation for continuous chaos, Phys. Lett. A 57 (5) (1976) 397–398.

[61] R. Barrio, M. Rodríguez, F. Blesa, Computer-assisted proof of skeletons of periodic orbits, Computer Physics Communications 183 (1) (2012) 80 – 85. doi:http://dx.doi.org/10.1016/j.cpc.2011.09.001.

[62] D. Wilczak, Abundance of heteroclinic and homoclinic orbits for the hyperchaotic Rössler system, Discrete Contin. Dyn. Syst. Ser. B 11 (4) (2009) 1039–1055.

[63] M. T. Nakao, A numerical verification method for the existence of weak solutions for nonlinear boundary value problems, Journal of Mathematical Analysis and Applications 164 (2) (1992) 489 – 507. doi:https://doi.org/10.1016/0022-247X(92)90129-2.
URL http://www.sciencedirect.com/science/article/pii/0022247X92901292

[64] A. Neumaier, Interval methods for systems of equations, Vol. 37 of Encyclopedia of Mathematics and its Applications, Cambridge University Press, Cambridge, 1990.

[65] C. L. Siegel, J. K. Moser, Lectures on Celestial Mechanics, Springer-Verlag, 1971.

[66] D. Wilczak, P. Zgliczyński, A geometric method for infinite-dimensional chaos: Symbolic dynamics for the Kuramoto-Sivashinsky PDE on the line, Journal of Differential Equations 269 (10) (2020) 8509 – 8548. doi:https://doi.org/10.1016/j.jde.2020.06.020.
URL http://www.sciencedirect.com/science/article/pii/S002203962030334X