Found. Comput. Math. (2002) 2:429–465

DOI: 10.1007/s102080010025

FOUNDATIONS OF COMPUTATIONAL MATHEMATICS
The Journal of the Society for the Foundations of Computational Mathematics

C^1 Lohner Algorithm

Piotr Zgliczynski

Jagiellonian University
Institute of Mathematics
Reymonta 4
30-059 Kraków, Poland
zgliczyn@im.uj.edu.pl

Abstract. We present a modification of the Lohner algorithm for the computation of rigorous bounds for solutions of ordinary differential equations together with partial derivatives with respect to initial conditions. The modified algorithm requires essentially the same computational effort as the original one. We applied the algorithm to show the existence of several periodic orbits for Rössler equations and the 14-dimensional Galerkin projection of the Kuramoto–Sivashinsky partial differential equation.

1. Introduction

In recent years we have witnessed a growing number of computer-assisted proofs in the dynamics of ordinary differential equations (ODEs), where a computer is used to rigorously check the assumptions of abstract theorems from the dynamical systems theory (see, e.g., [**2**], [**4**], [**6**], [**12**], [**19**], [**15**]). For all these proofs rigorous bounds for an appropriate Poincaré map were obtained with computer assistance.

The goal of this paper is to present a modification of the Lohner algorithm [**8**], [**9**] which, for a given autonomous ODE

$$\frac{dx}{dt} = f(x), \qquad x \in \mathbb{R}^n, \quad f \in C^\infty, \tag{1}$$

Date received: April 18, 2001. Final version received: January 10, 2002. Communicated by Arieh Iserles. Online publication: March 15, 2002.
AMS classification: 65G20, 65L05.

and a section Θ, rigorously computes the Poincaré map $P(x_0)$ and $(\partial P/\partial x)(x_0)$, where $x_0 \in \Theta$.

To this end we need to solve a system of ODEs consisting of (1) and the variational equation corresponding to it

$$\frac{dV}{dt}(t, x_0) = \frac{\partial f}{\partial x}(x(t)) \cdot V(t, x_0), \qquad V \in \mathbb{R}^{n\times n}, \tag{2}$$

with initial condition $V(0, x_0) = \mathrm{Id}$, where by Id we denote the identity matrix.

Once we have rigorous bounds for a trajectory of (1) and (2) we need to consider its intersection with Θ to obtain P and its derivatives.

By $\varphi(t, x_0)$ we will denote a solution at time t of (1) with initial condition $x(0) = x_0$. It is obvious from (1) and (2) that $V(t, x_0) = (\partial\varphi/\partial x)(t, x_0)$.

By *a C^0 algorithm* we refer to a procedure which gives rigorous estimates for $\varphi(t, x)$ (or the Poincaré map $P(x)$). By *a C^1 algorithm* we mean a rigorous procedure for the computation of $\varphi(t, x)$ and $\partial\varphi(t, x)/\partial x$ (or $P(x)$ and $\partial P/\partial x$).

Examples of C^0 algorithms: algorithms based on logarithmic norms (used in [**12**], [**4**]), the Lohner algorithm [**8**], or the Hermite–Oberschkoff method proposed by Nedialkov and Jackson (see [**13**] for a more complete list).

We can perform a C^1 computation just by a direct application a C^0 algorithm to the problem consisting of (1) and (2). This approach totally ignores the structure of the system (1) and (2) and usually leads to very poor performance and long computation times (see Section 6 for more comments). In this paper we present a modification of the Lohner algorithm, which allows us to perform the C^1 computations essentially in the same time as required for the C^0 algorithm.

The reader should be aware that the basic problem, with rigorous computations of solutions of (1), is our inability to obtain good rigorous bounds on the difference between solutions starting from different points (we refer to this as *the Lipschitz part of error*). A part of this inability is due to the very nature of interval computations—a phenomenon called *the wrapping effect*, which is discussed in virtually every paper in the field of rigorous computations for ODEs (see [**13**] and the references given there). This is the reason for the apparent complexity of Lohner-type algorithms—the original Lohner algorithm, the Hermite–Obreschkoff method [**13**], or the algorithm presented here—because an essential part of these algorithms is devoted to controlling the Lipschitz part of the error to avoid, or rather to suppress, the wrapping effect. This complexity results in the slowness of these algorithms compared to those based on logarithmic norms or on direct interval evaluation, when we consider the time needed to compute one initial condition. But the bounds for the Lipschitz part of the error obtained using this approach are so poor that it turns out that the Lohner-type algorithms are much faster in the task of computing the image of the Poincaré map of a cube with a desired accuracy.

There also exists another way to effectively control the Lipschitz part of the error. We call this *a division method*. It was used in [**12**], [**19**], [**20**]. To explain

the main idea let us consider the computation of $P([x])$, where $[x]$ is a box in $\mathbb{R}^n$ and P is a Poincaré map. As we compute $\varphi(t, [x])$ along the trajectory, the obtained image (denoted here by $\langle\varphi(t, [x])\rangle$) soon becomes considerably larger than the true image, $\varphi(t, [x])$, and usually starts to grow exponentially at a much faster rate than the intrinsic growth rate from the ODE under consideration. To deal with this problem we divide, at some moment t, the current value of $\langle\varphi(t, [x])\rangle$ into smaller pieces and we continue to evolve them separately. The division procedure can be applied many times until we reach the Poincaré section. It turns out that the small pieces will move away from one another at the rate given by the ODE and, since they are smaller, their growth rate due to the numerics will usually be smaller than that for the undivided set, hence resulting in much better bounds. As a result we allow the intrinsic dynamics of the ODEs to take part in the control of the Lipschitz part of the error, but this happens at the cost of managing the division process and the computation of a larger number of initial value problems.

One can expect that the optimal algorithm will be a combination of both the Lohner-type approach and the division method, but how to link them in the most efficient way will probably depend on the particular ODE under consideration.

Sections 2 and 3 contain a detailed description of the proposed C^1 Lohner algorithm. In Section 6 we discuss briefly the relative cost of the C^0 Lohner and the C^1 Lohner algorithms, we also address there the issue of a direct computation of solutions of (1) and (2) using the C^0 Lohner algorithm. In Section 5 we describe how we extract $P(x)$ and $\partial P/\partial x$ from rigorous estimates for $\varphi(t, x)$ and $\partial\varphi(t, x)/\partial x$.

In Section 7 we report on two applications of the proposed algorithm to obtain proofs of an existence of single periodic orbits for the Rössler equation and on the 14-dimensional Galerkin projection of the Kuramoto–Sivashinsky (KS) equations. The algorithm described here was also successfully applied to the Henon–Heiles Hamiltonian (see [**3**]), and to the planar restricted circular three-body problem (see [**21**]), to obtain an infinite number of geometrically distinct homo- and heteroclinic orbits to some periodic orbits.

The Appendix contains the formulas we used to generate the Taylor expansion for the applications described in Section 7.

2. C^1 Lohner Algorithm

The goal of this section is to present the C^1 Lohner algorithm for ODEs. In fact we give a description of three algorithms: the original Lohner algorithm (C^0 computation) and two C^1 algorithms called the C^1_1 and C^1_2 algorithms.

We want to solve the system consisting of (1) and (2) with the following initial conditions

$$x(0) \in [x_0] \subset \mathbb{R}^n, \qquad V(0, [x_0]) = \mathrm{Id}. \tag{3}$$

2.1. *Notation*

In the sequel, by Arabic figures we denote single-valued objects like vectors, real numbers, and matrices. Quite often in this paper we will use square brackets, e.g., $[r]$, to denote sets. Usually this will be some set constructed in the algorithm. Sets will also be denoted by single letters, e.g., S, when it is clear from the context that it represents a set. In situations when we want to stress (e.g., in the detailed description of an algorithm) that we have a set in a formula involving both single-valued objects and sets together we will use square brackets, hence we prefer to write $[S]$ instead of S to represent a set. From this point of view $[S]$ and S are different symbols in the alphabet used to name variables and, formally speaking, there is no relation between the set represented by $[S]$ and the object represented by S. Quite often in the description of the algorithm we will have a situation that both variables $[S]$ and S are used simultaneously, then usually $S \in [S]$, but this is always stated explicitly.

For a set $[S]$ by $[S]_I$ we denote the interval hull of $[S]$, i.e., the smallest product of intervals containing $[S]$. The symbol $\mathrm{hull}(x_1, \ldots, x_k)$ will denote the interval hull of intervals $x_1, \ldots, x_k$. For any interval set $[S] = [S]_I$ by $\mathrm{m}([S])$ we will denote a center point of $[S]_I$. For any interval $[a, b]$ we define a diameter by $\mathrm{diam}([a, b]) = b - a$. For an interval vector or an interval matrix $[S] = [S]_I$ by $\mathrm{diam}([S])$ we will denote the maximum of diameters of its components. For an interval $[x^-, x^+]$ we set $right([x^-, x^+]) = x^+$ and $left([x^-, x^+]) = x^-$.

For a set $X \subset \mathbb{R}^d$ by $\mathrm{int}X$ we denote an interior of X.

2.2. *Taylor Coefficients for $x(t, x_0)$ and $V(t, x_0)$*

Let $\Phi(h, x, p)$ denote the Taylor method of order p for (1). For a description of a procedure for the generation of Φ, for an arbitrary order for a wide class of functions, the reader is referred to [**10**], [**16**]. The basic idea of this procedure can be explained as follows. Let us set $x(t, x_0) = \varphi(t, x)$. By differentiation with respect to t of (1) and (2) we obtain

$$\begin{aligned}
\frac{d^2}{dt^2}x(t, x_0) &= \frac{d}{dt}f(x(t, x_0)) = df(x(t, x_0)) \cdot f(x(t, x_0)), \\
\frac{d^2}{dt^2}V(t, x_0) &= \frac{d}{dt}\left(df(x(t, x_0)) \cdot V(t, x_0)\right) \\
&= df^2(x(t, x_0))(f(x(t, x_0), f(x(t, x_0)) \\
&\quad + df(x(t, x_0)) \cdot df(x(t, x_0)) \cdot V(t, x_0).
\end{aligned}$$

We see that the second derivatives with respect to t of $x(t, x_0)$ and $V(t, x_0)$ are functions of $x(t, x_0)$ and $V(t, x_0)$ only. Hence there exist functions $a_1 \colon \mathbb{R}^n \to \mathbb{R}^n$

and $b_1\colon \mathbb{R}^n \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ such that

$$\frac{d^2}{dt^2}x(t,x_0) = \frac{d}{dt}f(x(t,x_0)) = a_1(x(t,x_0)), \tag{4}$$

$$\frac{d^2}{dt^2}V(t,x_0) = \frac{d}{dt}(df(x(t,x_0))\cdot V(t,x_0)) = b_1(x(t,x_0),V(t,x_0)). \tag{5}$$

An easy induction shows that, in general for $i \geq 2$, we have

$$\frac{d^i}{dt^i}x(t,x_0) = \frac{d^{i-1}}{dt^{i-1}}f(x(t,x_0)) = a_{i-1}(x(t,x_0)), \tag{6}$$

$$\begin{aligned}\frac{d^i}{dt^i}V(t,x_0) &= \frac{d^{i-1}}{dt^{i-1}}(df(x(t,x_0))\cdot V(t,x_0))\\ &= b_{i-1}(x(t,x_0),V(t,x_0)),\end{aligned} \tag{7}$$

for some functions $a_i\colon \mathbb{R}^n \to \mathbb{R}^n$ and $b_i\colon \mathbb{R}^n \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$. In the sequel we will use the following notation

$$\frac{d^i}{dt^i}f = a_i, \tag{8}$$

$$\frac{d^i}{dt^i}\left(\frac{\partial f}{\partial x}V\right) = b_i. \tag{9}$$

In this notation the symbols $(d^i/dt^i)f$ and $(d^i/dt^i)((\partial f/\partial x)V)$ are the functions of x and V.

Explicit formulas for Taylor coefficients for polynomials of degree 2 are given in the Appendix.

2.3. *An Outline of Algorithms*

Before we present the Lohner algorithm and our C^1 modification of it we would like to stress the main points in the original Lohner algorithm [**8**]:

- Even to perform C^0 computations one needs some kind of a C^1 data, because a direct evaluation of $\Phi(h, x_0, p)$ using an interval arithmetic leads to *the wrapping effect*, which allows us to integrate (1) for a very short time, only.
- The C^1 information about φ is obtained by computing a partial derivative of an explicitly given $\Phi(h, x, p)$ instead of estimating $\partial\varphi(h, x)/\partial x$.
- The rearrangement computations together with the knowledge of $\partial\Phi(h, x, p)/\partial x$ reduces considerably the wrapping effect.

Obviously to have a rigorous procedure one also needs to take care of the errors introduced by the finite computer arithmetic (round-off errors) and the discretization error of the numerical method used (the Taylor method in the Lohner algorithm).

The round-off errors are taken care of by the interval arithmetic (see [**8**], [**11**] and the references given there).

The basic idea of modification of an original Lohner algorithm, which leads to a C^1 Lohner algorithm presented here, is the realization that $\partial\Phi(h, x, p)/\partial x$ is also a Taylor expansion for $\partial\varphi(h, x)/\partial x$, hence with a little additional computational effort we can turn a C^0 algorithm into a C^1 algorithm. More precisely, we have the following:

Lemma 1. *Consider the problem* (1)–(3). *Let* $h > 0$. *Assume that* $[W_1] \subset \mathbb{R}^n$ *is a compact and convex set such that*

$$\varphi([0, h], [x_0]) \subset [W_1], \tag{10}$$

and $[W_3] \subset \mathbb{R}^{n\times n}$ *is a compact and convex set such that*

$$V([0, h], [x_0]) \subset [W_3]. \tag{11}$$

Then:

- $\dfrac{\partial\varphi}{\partial x}(t, x) = V(t, x)$ *for* $t \in [0, h]$ *and* $x \in [x_0]$; *and*
- $V(h, [x_0]) \subset \dfrac{\partial\Phi(h, [x_0], p)}{\partial x} + \dfrac{h^{p+1}}{(p+1)!}\left(\dfrac{d^p}{dt^p}\left(\dfrac{\partial f}{\partial x}V\right)([W_1], \mathrm{Id})\right)\cdot[W_3]$.

Let us remind the reader (see Section 2.2), that the symbol $(d^p/dt^p)((\partial f/\partial x)V)$ is a function of x and V, hence it makes sense to plug in $[W_1]$ for x and Id for V.

Proof. The first assertion is obvious.

To prove the second assertion we show that it is a Taylor expansion for V of order p plus a remainder term.

Observe that by the definition of Φ it follows that

$$\Phi(h, x, p) = \sum_{i=0}^{p} \frac{\partial^i\varphi(0, x)}{\partial t^i}\frac{h^i}{i!}. \tag{12}$$

We differentiate the above formula with respect to x to obtain

$$\frac{\partial\Phi(h, x, p)}{\partial x} = \sum_{i=0}^{p} \frac{\partial^i}{\partial t^i}\frac{\partial\varphi(0, x)}{\partial x}\frac{h^i}{i!} = \sum_{i=0}^{p} \frac{d^i}{dt^i}V(0, x)\frac{h^i}{i!}. \tag{13}$$

It remains to compute an enclosure for the remainder term $[R] \subset \mathbb{R}^{n\times n}$ given by

$$[R]_{ij} = \frac{h^{p+1}}{(p+1)!}\frac{d^{p+1}}{dt^{p+1}}V_{ij}(\theta_{ij}h, x) \tag{14}$$

for $i, j = 1, \ldots, n$ and $x \in [x_0]$, where $\theta_{ij} \in [0, 1]$ depends on x.

We will derive a different expression for the remainder term. Observe that, for $t, s \geq 0$, we have

$$V(t+s, x) = V(s, \varphi(t, x)) \cdot V(t, x). \tag{15}$$

Hence, after taking m derivatives with respect to s, we obtain for $s = 0$ the following identity

$$\frac{d^m}{dt^m} V(t, x) = \left(\frac{d^m}{dt^m} V(0, \varphi(t, x)) \right) \cdot V(t, x). \tag{16}$$

Hence, for $m = p+1$, taking into account that $\varphi(\theta_{ij} h, x) \subset [W_1]$, $V(t, x) \subset [W_3]$ for $x \in [x_0]$, and (7) and (9) we obtain

$$[R] \subset \frac{h^{p+1}}{(p+1)!} \left(\frac{d^p}{dt^p} \left(\frac{\partial f}{\partial x} V \right) ([W_1], \mathrm{Id}) \right) \cdot [W_3]. \tag{17}$$

□

Remark 2. Observe that we can enclose the remainder term for $V(h, [x_0])$ in the second assertion of Lemma 1 by

$$[R] = \frac{d^p}{dt^p} \left(\frac{\partial f}{\partial x} V \right) ([W_1], [W_3]) \frac{h^{(p+1)}}{(p+1)!}, \tag{18}$$

but the formula given there is cheaper to compute and gives better bounds.

Let us fix p_e and p_v being, respectively, the orders for the Taylor methods used to solve (1) and (2), respectively. For the C_2^1 algorithm we require that $p_v \geq p_e$.

In the description below the objects with an index k refer to the current values and those with an index $k+1$ are the values after the next time step.

One step of the Lohner algorithms is a shift along the trajectory of system (1) and (2) with the following input and output data:

Input data:

- t_k is the current time;
- h_k is a time step;
- $[x_k] \subset \mathbb{R}^n$, such that $\varphi(t_k, [x_0]) \subset [x_k]$; and
- (for C^1 algorithms, only) $[V_k] \subset \mathbb{R}^{n \times n}$, such that $(\partial \varphi / \partial x)(t_k, [x_0]) \subset [V_k]$.

Output data:

- $t_{k+1} = t_k + h_k$ is a new current time;
- $[x_{k+1}] \subset \mathbb{R}^n$, such that $\varphi(t_{k+1}, [x_0]) \subset [x_{k+1}]$; and
- (for C^1 algorithms, only) $[V_{k+1}] \subset \mathbb{R}^{n \times n}$, such that $(\partial \varphi / \partial x)(t_{k+1}, [x_0]) \subset [V_{k+1}]$.

We do not specify here a form (a representation) of sets $[x_k]$ and $[V_k]$. They can be interval sets, balls, doubletons, etc. (see [**11**]). This issue is very important in handling the wrapping effect and will be discussed in detail in Section 3.

One step of the algorithm consists of the following parts:

Part 1(a) A computation of a rough enclosure $[W_1]$ for (1).
$[W_1]$ is a compact and convex set such that

$$\varphi([0, h_k], [x_k]) \subset [W_1]. \tag{19}$$

Part 1(b) (Required for the C_1^1 Algorithm). A computation of a rough enclosure $[W_2]$ for (1).
$[W_2]$ is a compact and convex set, such that

$$\varphi([0, h_k], \mathrm{m}([x_k])) \subset [W_2]. \tag{20}$$

Part 2 (Required for C^1 Algorithms). A computation of a rough enclosure $[W_3]$ for (2).
$[W_3]$ is a compact and convex set, such that

$$V([0, h_k], [x_k]) \subset [W_3]. \tag{21}$$

Part 3 A computation of $\partial\Phi/\partial x$ and (for C^1 algorithms) $[V_{k+1}]$.
Part 4 A computation of $[x_{k+1}]$.

2.4. *Part* 1*—A Computation of a Rough Enclosure for Equation* (1)

Let $[Y] = [Y]_I$ be an interval set, such that $[x_k] \subset [Y]$. It is easy to see that if

$$[[x_k] + [0, h_k] f([Y])]_I \subset \mathrm{int}[Y] \tag{22}$$

holds, then

$$\varphi([0, h_k], [x_k]) \subset [W_1] = [[x_k] + [0, h_k] f([Y])]_I\,, \tag{23}$$

where by $[[x_k] + [0, h_k] f([Y])]_I$ we denote the interval enclosure of the set $[x_k] + [0, h_k] f([Y])$.

Equation (22) suggests an iterative procedure. We can start with some $[Y_0]$ such that $[x_k] \subset [Y_0]$ and then set $[Y_{i+1}] = [[x_k] + [0, h_k] f([Y_i])]_I$, till (22) holds. This procedure does not always work, for example, usually h_k cannot be too large for this procedure to succeed.

Sometimes, if some a-priori bounds, B, for the solutions of (1) are known we can take these bounds as $[W_1]$. To tighten these bounds we can refine $[W_1]$ as follows:

$$[W_1] = B \cap [[x_k] + [0, h_k] f(B)]_I. \tag{24}$$

2.5. *Part 2—A Computation of a Rough Enclosure for the Variational Part*

In this section by $\|x\|$ we denote an arbitrary norm.

In order to present and justify the procedure for producing a rough enclosure for the variational part we need to recall a notion of the logarithmic norm of a square matrix.

Definition 1 [**5**, Definition I.10.4]. Let Q be a square matrix, then we call

$$\mu(Q) = \lim_{h>0, h\to 0} \frac{\|\mathrm{Id} + hQ\| - 1}{h}$$

the *logarithmic norm* of Q.

Theorem 3 [**5**, Definition I.10.5]. *The logarithmic norm is obtained by the following formulas*:

- *for the Euclidean norm*

$$\mu(Q) = \textit{the largest eigenvalue of } 1/2(Q + Q^T);$$

- *for the max norm* $\|x\| = \max_k |x_k|$:

$$\mu(Q) = \max_k \left(q_{kk} + \sum_{i\neq k} |q_{ki}| \right);$$

- *for the norm* $\|x\| = \sum_k |x_k|$:

$$\mu(Q) = \max_i \left(q_{ii} + \sum_{k\neq i} |q_{ki}| \right).$$

The following theorem was proved in [**5**, Theorem I.10.6].

Theorem 4. *Consider the differential equation*

$$\frac{dx}{dt} = f(t, x), \qquad f \in \mathcal{C}^1, \quad x \in \mathbb{R}^n. \tag{25}$$

Let $t_0 < t_1 < \cdots < t_k = t_N$. *Let* $x(t)$ *be any solution of* (25) *on the interval* $[t_0, t_N]$. *Let* $v(t)$ *denote the Euler polygon, so that*

$$v'(t) = f(t_i, x_i) \qquad \textit{for } t_i < t < t_{i+1}, \quad i = 0, \ldots, N-1. \tag{26}$$

Suppose that we have the estimates for $t_0 \le t \le t_N$:

$$\mu\left(\frac{\partial f}{\partial x}(t, \eta)\right) \le l(t) \qquad \textit{for} \quad \eta \in [x(t), v(t)], \tag{27}$$

$$\|v'(t) - f(t, v(t))\| \le \delta(t), \tag{28}$$

$$\|v(t_0) - x(t_0)\| \le \rho. \tag{29}$$

Then, for $t_N \geq t > t_0$, we have

$$\|x(t) - v(t)\| \leq e^{L(t)} \left(\rho + \int_{t_0}^{t} e^{-L(s)} \delta(s)\, ds \right), \tag{30}$$

where $L(t) = \int_{t_0}^{t} l(s)\, ds$.

In fact the proof of the above theorem, given in [**5**], is valid for any function v, which is piecewise C^1. Hence we can take $v(t)$ to be also the solution of (25) in the above theorem. In this situation $\delta(t) = 0$ and we obtain the following:

Theorem 5. *Let Z be a convex set and let $\mu((\partial f/\partial x)(x)) \leq l$ for $x \in Z$. Suppose that $\varphi(t, x) \in Z$ and $\varphi(t, y) \in Z$ for all $t \in [0, h]$. Then*

$$\|\varphi(h, y) - \varphi(h, x)\| \leq e^{hl} \|y - x\|,$$
$$\left\| \frac{\partial \varphi(t, x)}{\partial x} \right\| \leq e^{tl} \qquad \text{for} \quad t \in [0, h].$$

The enclosure procedure can be formulated as follows:

Input parameters:

- h_k is a time step;
- $[x_k] \subset \mathbb{R}^n$ and $[V_k] \subset \mathbb{R}^{n \times n}$; and
- $[W_1] \subset \mathbb{R}^n$ is compact and convex, such that $\varphi([0, h_k], [x_k]) \subset [W_1]$.

On **output** we compute $[W_3]$ as follows:

(1) $l = \mu((\partial f/\partial x)([W_1]))$;
(2) we define an interval matrix $[W] \subset \mathbb{R}^{n \times n}$ by $[W_{ij}] = [\pm \max(e^{l[0,h_k]})]$, $i, j = 1, \ldots, n$; and
(3) $[W_3] = (\mathrm{Id} + [[0, h_k] \cdot (\partial f/\partial x)([W_1])]_I \cdot [W]) \cap [W]$.

From Theorem 5 it follows that $V([0, h_k], [x_k]) \subset [W]$, where $[W]$ is defined in the second step of the above procedure. The next step is an attempt to refine this enclosure by using (2).

2.6. *Part 3—A Computation of the Variational Part*

Input parameters:

- h_k is a time step;
- $[x_k] \subset \mathbb{R}^n$ and $[V_k] \subset \mathbb{R}^{n \times n}$;
- $[W_1] \subset \mathbb{R}^n$ is compact and convex, such that $\varphi([0, h_k], [x_k]) \subset [W_1]$; and
- (for C^1 algorithms) $[W_3] \subset \mathbb{R}^{n \times n}$ such that $V([0, h_k], [x_k]) \subset [W_3]$.

From the linearity of (2), with respect to V, it follows that

$$V(t_k + h_k, [x_0]) \subset V(h_k, [x_k]) \cdot [V_k],$$

hence to compute $[V_{k+1}]$ it is enough to compute bounds for $V(h_k, [x_k])$, which we will denote $[J_k]$. To control the Lipschitz part of the error for the x-variable we will use the matrix $[A_k]$, which for the C_1^1 algorithm is equal to $[J_k]$ (the partial derivative of the flow with respect to initial conditions) and for the C^0 and C_2^1 algorithms we use instead the partial derivative of the Taylor expansion (this is an original Lohner approach) .

To be more specific, we proceed as follows:

C_1^1 algorithm. We set

$$\begin{aligned}[A_k] = [J_k] \;=\; \mathrm{Id} + \sum_{i=1}^{p_v} \frac{d^{(i-1)}}{dt^{(i-1)}} \left(\frac{\partial f}{\partial x} V\right) ([x_k], \mathrm{Id}) \frac{h_k^i}{i!} \\ + \frac{h_k^{p_v+1}}{(p_v+1)!} \left(\frac{d^{p_v}}{dt^{p_v}} \left(\frac{\partial f}{\partial x} V\right) ([W_1], \mathrm{Id})\right) \cdot [W_3]. \qquad (31)\end{aligned}$$

From Lemma 1 it follows that we have

$$\frac{\partial \varphi}{\partial x}(h_k, [x_k]) = V(h_k, [x_k]) \subset [J_k] = [A_k]. \qquad (32)$$

C_2^1 Algorithm. We set

$$[A_k] = \mathrm{Id} + \sum_{i=1}^{p_e} \frac{d^{(i-1)}}{dt^{(i-1)}} \left(\frac{\partial f}{\partial x} V\right) ([x_k], \mathrm{Id}) \frac{h_k^i}{i!}. \qquad (33)$$

The remainder of the Taylor expansion of the order p_v, plus the error term, are given by

$$\begin{aligned}[\Delta A_k] \;=\; \sum_{i=p_e+1}^{p_v} \frac{d^{(i-1)}}{dt^{(i-1)}} \left(\frac{\partial f}{\partial x} V\right) ([x_k], Id) \frac{h_k^i}{i!} \\ + \frac{h_k^{p_v+1}}{(p_v+1)!} \left(\frac{d^{p_v}}{dt^{p_v}} \left(\frac{\partial f}{\partial x} V\right) ([W_1], \mathrm{Id})\right) \cdot [W_3]. \qquad (34)\end{aligned}$$

Finally, we set

$$[J_k] = [A_k] + [\Delta A_k]. \qquad (35)$$

Observe that, by Lemma 1, we have

$$\begin{aligned}\frac{\partial \Phi}{\partial x}(h, [x_k], p_e) \;&\subset\; [A_k], \\ \frac{\partial \varphi}{\partial x}(h_k, [x_k]) \;&=\; V(h_k, [x_k]) \subset [J_k].\end{aligned}$$

On output:

- the interval matrix $[A_k]$, such that for the C^0 and C^1_2 algorithms we have

$$\frac{\partial \Phi}{\partial x}(h_k, [x_k], p_e) \subset [A_k], \tag{36}$$

  and for the C^1_1 algorithm we have

$$\frac{\partial \varphi}{\partial x}(h_k, [x_k]) \subset [A_k]; \tag{37}$$

- (for C^1 algorithms, only) the interval matrix $[V_{k+1}]$, such that

$$\frac{\partial}{\partial x}\varphi(h_k, [x_k]) \subset [J_k], \tag{38}$$

$$V(t_k + h_k, [x_0]) \subset [V_{k+1}] = [J_k] \cdot [V_k]. \tag{39}$$

We do not specify here how we evaluate the product $[J_k] \cdot [V_k]$. This depends on the representation of $[V_k]$ and is discussed in Section 3.

Let us stress here that what we add in the C^1 algorithms, when compared to the C^0 algorithm, is only the computation of the error term for $[J_k]$ plus an evaluation of $[J_k] \cdot [V_k]$, because the Taylor expansion for V is already computed in $[A_k]$, which also is present in the C^0 algorithm.

2.7. *Part* 4*—A Moving Forward with x*

Input parameters for the C^0 and C^1_2 algorithms:

- h_k is a time step;
- $[x_k] \subset \mathbb{R}^n$ is a current estimate for $\varphi(t_k, [x_0])$;
- $[W] \subset \mathbb{R}^n$ is a compact and convex set, such that $\varphi([0, h_k], [x_k]) \subset [W]$ (this is the set $[W_1]$ obtained in Part 1(a)); and
- $[A_k] \subset \mathbb{R}^{n\times n}$, such that $(\partial\Phi/\partial x)(h_k, [x_k], p_e) \subset [A_k]$.

Input parameters for the C^1_1 algorithm:

- h_k is a time step;
- $[x_k] \subset \mathbb{R}^n$ is a current estimate for $\varphi(t_k, [x_0])$;
- $[W] \subset \mathbb{R}^n$ is a compact and convex set, such that $\varphi([0, h_k], \mathrm{m}([x_k])) \subset [W]$ (this is the set $[W_2]$ obtained in Part 1(b)); and
- $[A_k] \subset \mathbb{R}^{n\times n}$, such that $(\partial\varphi/\partial x)(h_k, [x_k]) \subset [A_k]$.

On **output** we set

$$[x_{k+1}] = \Phi(h_k, \mathrm{m}([x_k]), p_e) + \frac{d^{p_e}}{dt^{p_e}} f([W]) \frac{h_k^{p_e+1}}{(p_e+1)!} + [A_k]([x_k] - \mathrm{m}(x_k)). \tag{40}$$

3. The Rearrangement Computations

It is well-known that the straightforward interval evaluation of the products of matrices and vectors leads immediately to the so-called *wrapping effect* (see, e.g., [**8**], [**13**], [**2**]). As will follow from the discussion below we can avoid it to some extent, by treating the products appearing in the algorithm carefully. We will refer to this part of the algorithm as *the rearrangement computations*.

3.1. *Evaluation of Equation* (40)

To discuss various methods of an evaluation of (40), following [**8**], [**9**], we decompose $[x_k]$ as follows:

$$[x_k] = x_k + [r_k] \qquad \text{where} \quad x_k = \mathrm{m}([x_k]), \quad [r_k] = [x_k] - x_k. \tag{41}$$

We set

$$x_{k+1} = \mathrm{m}\left(\Phi(h_k, \mathrm{m}([x_k]), p_e) + \frac{d^{p_e}}{dt^{p_e}} f([W]) \frac{h_k^{p_e+1}}{(p_e+1)!}\right),$$

$$[z_{k+1}] = \Phi(h_k, \mathrm{m}([x_k]), p_e) + \frac{d^{p_e}}{dt^{p_e}} f([W]) \frac{h_k^{p_e+1}}{(p_e+1)!} - x_{k+1}.$$

With these notations (40) becomes

$$[r_{k+1}] = [A_k][r_k] + [z_{k+1}]. \tag{42}$$

Evaluation 1. In [**11**] terminology this approach is called *an interval set*. This is a direct evaluation of (42) using interval arithmetic. This method is simple and fast, but usually produces very bad bounds due to the wrapping effect.

To avoid the wrapping effect Lohner proposed the following: instead of representing $[r_k]$ as an interval vector, he proposed using parallelograms (interval vectors in other coordinate systems), i.e., $[r_k] = B_k[\hat{r}_k]$, where B_k are nonsingular matrices and $[\hat{r}_k]$ are interval vectors. Equation (42) becomes

$$[r_{k+1}] = [A_k][r_k] + [z_{k+1}] = B_{k+1}(B_{k+1}^{-1}[A_k]B_k[\hat{r}_k] + B_{k+1}^{-1}[z_{k+1}]). \tag{43}$$

In computer calculations it is usually impossible to find an exact inverse of a given matrix. So in fact we deal with the interval matrices $[B_i]$ and $[B_i^{-1}]$.

Finally, we calculate $[\hat{r}_k]$ as follows:

$$[r_0] = [B_0][\hat{r}_0], \qquad [B_0] = \{\mathrm{Id}\}, \tag{44}$$

$$[\hat{r}_{k+1}] = ([B_{k+1}^{-1}][A_k][B_k])[\hat{r}_k] + [B_{k+1}^{-1}][z_{k+1}], \tag{45}$$

$$[r_{k+1}] = [B_{k+1}][\hat{r}_{k+1}]. \tag{46}$$

It should be stressed that only $[\hat{r}_k]$ is computed using interval arithmetic. When evaluating the right-hand side of (45) we first do matrix multiplications as indicated by square brackets. *This is the place where we can reduce the wrapping effect, if we make a good choice of* $[B_k]$*'s* . We evaluate formula (46) to compute rough enclosures and at the end of computations (while considering the intersection of the orbit of $[x_0]$ with a Poincaré section).

Evaluation 2. In [**11**] terminology this approach is called *a parallelepiped*. We choose $B_{k+1} \in [A_k][B_k]$. This is not a method recommended for general applications. The main problem with this method is the need to calculate the inverse of a matrix.

Evaluation 3. In [**11**] terminology this approach is called *a cuboid*. Choose a matrix $U \in [A_k][B_k]$, we perform an approximate floating-point QR-decomposition of U. Let the matrix Q be a Q factor from this decomposition. The matrix Q is very close to an orthogonal one. Next we apply an interval Gram–Schmidt procedure to columns of Q to obtain a matrix $[Q_0]$. Observe that $[Q_0]$ contains an orthogonal matrix, Q_1, hence $Q_1^{-1} = Q_1^T \in [Q_0]^T$.

We set $B_{k+1} = Q_1$, $B_{k+1}^{-1} = Q_1^T$. This leads to

$$[B_{k+1}] = [Q_0], \qquad [B_{k+1}^{-1}] = [Q_0]^T. \tag{47}$$

A slightly different approach (probably more efficient) was proposed originally by Lohner. Namely, instead of orthogonalizing the columns of Q he proposed to rigorously compute the inverse of Q. Observe that since Q is almost orthogonal, the rigorous Q^{-1} can be obtained very easily with little cost.

Evaluation 4. In [**11**] terminology this approach is called *a doubleton*. This method is designed to handle the situation, when the initial error is large in comparison to local errors produced at every step. Lohner proposed to keep track separately of the error originating from $[r_0]$ and the local errors produced at every step. An example of such a method is given by

$$[r_{k+1}] = [E_{k+1}][r_0] + [\tilde{r}_{k+1}], \tag{48}$$

where

$$[\tilde{r}_{k+1}] = [A_k][\tilde{r}_k] + [z_{k+1}], \tag{49}$$

and

$$[E_{k+1}] = [A_k][E_k], \qquad [E_0] = \mathrm{Id}, \tag{50}$$

and $[\tilde{r}_k]$ is evaluated using any method described previously (preferably Evaluation 3). It is easy to see that the matrix $[E_k]$ corresponds to $\partial\varphi(t_{k+1}, \cdot)/\partial x$. We still have some wrapping effect in the product $[A_k][E_k]$, which becomes more and

more important when we want to follow the trajectory for a longer time. To avoid this, Lohner proposed the following:

$$[r_{k+1}] = C_{k+1}[r_0] + [\tilde{r}_{k+1}], \tag{51}$$

where

$$[\tilde{r}_{k+1}] = [A_k][\tilde{r}_k] + [z_{k+1}] + ([A_k]C_k - C_{k+1})[r_0], \qquad [\tilde{r}_0] = 0, \tag{52}$$

and

$$C_0 = \mathrm{Id}, \qquad C_{k+1} \in [A_k]C_k, \tag{53}$$

and $[\tilde{r}_k]$ is evaluated using any method described previously (preferably Evaluation 3).

The tests in [**11**] on the Rössler equation and on the Lorenz equation show that the last approach—a doubleton—is far better than the previous ones.

3.2. *Evaluation of Equation* (39)

To evaluate (39) we mimic the approach from the previous subsection.

Evaluation 1. The direct interval evaluation.

To discuss other evaluations let us first decompose the interval matrices $[V_k]$ into the center point and the "remainder"

$$[V_k] = V_k + [\Delta V_k]. \tag{54}$$

We can rewrite (39) as follows:

$$V_{k+1} + [\Delta V_{k+1}] = [J_k]V_k + [J_k][\Delta V_k]. \tag{55}$$

Let

$$V_{k+1} = m([J_k]V_k), \tag{56}$$

$$[Z_{k+1}] = [J_k]V_k - V_{k+1}. \tag{57}$$

We have

$$[\Delta V_{k+1}] = [J_k] \cdot [\Delta V_k] + [Z_{k+1}]. \tag{58}$$

We see that this equation has the same structure as (42), where $[\Delta V_k]$ corresponds to $[r_k]$, $[Z_{k+1}]$ to $[z_{k+1}]$, and $[J_k]$ to $[A_k]$. Hence we can treat it with similar methods.

For any family of nonsingular matrices $[B_k]$, $k = 0, 1, \ldots$, we set $[\Delta V_k] = [B_k][\widetilde{\Delta V_k}]$. Equation (58) becomes

$$[\Delta V_{k+1}] = [B_{k+1}]([B_{k+1}^{-1}][J_k][B_k][\widetilde{\Delta V_k}] + [B_{k+1}^{-1}][Z_{k+1}]). \tag{59}$$

We define the following scheme for an evaluation of (58):

$$[\Delta V_k] = [B_k][\widetilde{\Delta V_k}],$$
$$[\widetilde{\Delta V_0}] = 0, \qquad [B_0] = \mathrm{Id},$$
$$[\widetilde{\Delta V_{k+1}}] = ([B_{k+1}^{-1}][J_k][B_k])[\widetilde{\Delta V_k}] + [B_{k+1}^{-1}][Z_{k+1}].$$

Evaluation 2. Choose B_{k+1} to be any matrix in $[J_k] \cdot [B_k]$.

Evaluation 3. Take any $U \in [J_k] \cdot [B_k]$, perform the QR-decomposition of U, and set $[B_{k+1}] = [Q]$. Just as in the previous subsection this method appears to be better than the previous evaluations.

Evaluation 4. Since we start with a zero matrix ($[\Delta V_0] = 0$), there is no Lipschitz part at the beginning and we apply any of the previous evaluations (preferably the third one) till $\widetilde{\Delta V_k}$ becomes "thick" (its diameter becomes larger than some threshold value). Suppose that this happens after k_0 steps. Then we switch to a doubleton representation as follows:

$$[\Delta V_k] = C_k^V [\Delta V_0] + \overline{\Delta V_k}, \qquad k \geq k_0,$$
$$C_{k+1}^V \in [J_k] \cdot C_k^V, \qquad \overline{\Delta V_{k+1}} = [J_k] \cdot [\overline{\Delta V_k}] + ([J_k] \cdot C_k^V - C_{k+1}^V)[\Delta V_0] + [Z_{k+1}].$$

We initiate the variables $C_{k_0}^V$, $[\Delta V_0]$, $\overline{\Delta V_{k_0}}$, and redefine $[B_{k_0}]$ as follows:

$$[\Delta V_0] = [\widetilde{\Delta V_{k_0}}], \qquad C_{k_0}^V = m([B_{k_0}]),$$
$$\overline{\Delta V_{k_0}} = ([B_{k_0}] - C_{k_0}^V)[\Delta V_0], \qquad [B_{k_0}] = \mathrm{Id}.$$

The interval matrix $[\overline{\Delta V_k}]$ is evaluated using any of the previous evaluation methods (preferably Evaluation 3).

4. Rigorous Estimates between Time Steps

The goal of this section is to answer the following question, which is very important in the computation of the Poincaré map:

How to estimate $\varphi(t, x)$ and $\partial\varphi/\partial x$ between time steps?

Obviously for this purpose we can use the rough enclosures $[W_1]$ and $[W_3]$. Here we present a much more efficient and relatively cheap approach.

The following lemma tells us how well $\varphi(\tau + t, x)$ for $t \in (0, h)$ is approximated by the segment joining $\varphi(\tau, x)$ and $\varphi(\tau + h, x)$.

Lemma 6. *Let $Z \subset \mathbb{R}^n$ be a convex set such that $\varphi(\tau + t, x) \in Z$ for $t \in [0, h]$, then*

$$\left|\varphi_i(\tau + t, x) - \left(\left(1 - \frac{t}{h}\right)\varphi_i(\tau, x) + \frac{t}{h}\varphi_i(\tau + h, x)\right)\right| \leq \frac{h^2}{2} \max_{z \in Z} \left|\sum_j \frac{\partial f_i}{\partial x_j}(z) \cdot f_j(z)\right|. \quad \square \quad (60)$$

Proof. Without any loss of generality we can assume that $\tau = 0$, hence $\varphi(\tau, x) = x$. Observe that we can represent the interval joining x and $\varphi(h, x)$ as follows:

$$\left(1 - \frac{t}{h}\right)x + \frac{t}{h}\varphi_i(h, x) = x + t\frac{\varphi(h, x) - x}{h}, \quad (61)$$

$$\left(1 - \frac{t}{h}\right)x + \frac{t}{h}\varphi_i(h, x) = \varphi(h, x) - \left(1 - \frac{t}{h}\right)(\varphi(h, x) - x). \quad (62)$$

We will use formula (61) for $t \in [0, h/2]$ and formula (62) for $t \in [h/2, h]$.

Assume that $t \in [0, h/2]$. From the mean value theorem we obtain, for some $\theta_1, \theta_2, \theta_3 \in [0, 1]$:

$$\begin{aligned}\left|\varphi_i(t, x) - \left(x_i + t\frac{\varphi_i(h, x) - x_i}{h}\right)\right| &= \left|(\varphi_i(t, x) - x_i) - t\frac{\varphi_i(h, x) - x_i}{h}\right| \\ &= |t||f_i(\varphi(\theta_2 t, x)) - f_i(\varphi(\theta_1 h, x))| \\ &= |t||\theta_2 t - \theta_1 h| \\ &\quad \times \left|\sum_j \frac{\partial f_i}{\partial x_j}(\varphi(\theta_3 h, x)) \cdot f_j(\varphi(\theta_3 h, x))\right| \\ &\leq \frac{h^2}{2}\left|\sum_j \frac{\partial f_i}{\partial x_j}(z) \cdot f_j(z)\right|,\end{aligned}$$

where $z = \varphi(\theta_3 h, x) \in Z$.

Similarly, for $t \in [h/2, h]$, we have

$$\begin{aligned}&|\varphi_i(t, x) - (\varphi_i(h, x) - (1 - t/h)(\varphi_i(h, x) - x_i))| \\ &\quad = |(\varphi_i(t, x) - \varphi_i(h, x)) + (1 - t/h)(\varphi_i(h, x) - x_i))| \\ &\quad = |(t - h) f_i(\varphi_i(\theta_1 h, x) + (1 - t/h)h f_i(\varphi_i(\theta_2 h, x)| \\ &\quad = (h - t)|\theta_1 h - \theta_2 h|\left|\sum_j \frac{\partial f_i}{\partial x_j} f_j(z)\right| \leq \frac{h^2}{2}\left|\sum_j \frac{\partial f_i}{\partial x_j}(z) \cdot f_j(z)\right|,\end{aligned}$$

where $z = \varphi(\theta_3 h, x) \in Z$. $\square$

When we apply Lemma 6 to system (1) and (2), then for the right-hand side in (60) for V_{ij} we obtain the following expression

$$\sum_l \frac{\partial V'_{ij}}{\partial x_l} f_l + \sum_{lm} \frac{\partial V'_{ij}}{\partial V_{lm}} V'_{lm} = \sum_{kl} \frac{\partial^2 f_i}{\partial x_k \, \partial x_l} V_{kj} f_l + \sum_l \frac{\partial f_i}{\partial x_l} V'_{lj}$$
$$= \sum_{kl} \frac{\partial^2 f_i}{\partial x_k \, \partial x_l} V_{kj} f_l + \sum_{lm} \frac{\partial f_i}{\partial x_l} \frac{\partial f_l}{\partial x_m} V_{mj}.$$

Hence we have proved the following:

Lemma 7. *Let $Z \subset \mathbb{R}^n$ and $M \subset \mathbb{R}^{n\times n}$ be convex sets such that $\varphi(\tau + t, x) \in Z$ and $V(\tau + t, x) \in M$ for $t \in [0, h]$, then*

$$\left| V_{ij}(\tau + t, x) - \left(\left(1 - \frac{t}{h} \right) V_{ij}(\tau, x) + \frac{t}{h} V_{ij}(\tau + h, x) \right) \right|$$
$$\leq \frac{h^2}{2} \max_{z \in Z, A \in M} \left| \sum_{kl} \frac{\partial^2 f_i}{\partial x_k \, \partial x_l}(z) A_{kj} f_l(z) + \sum_{lm} \frac{\partial f_i}{\partial x_l}(z) \frac{\partial f_l}{\partial x_m}(z) A_{mj} \right|.$$

Lemmas 6 and 7 offer a justification for the following procedure for the estimation of $\varphi([0, h_k], [x_k])$ and $V(t_k + [0, h_k], [x_0])$.

PROCEDURE:

Input parameters:

- h_k is a time step;
- $[x_k] \subset \mathbb{R}^n$, such that $\varphi(t_k, [x_0]) \subset [x_k]$;
- $[x_{k+1}] \subset \mathbb{R}^n$ such that $\varphi(h_k, [x_k]) \subset [x_{k+1}]$;
- $[W_1] \subset \mathbb{R}^n$ compact and convex, such that $\varphi([0, h_k], [x_k]) \subset [W_1]$;
- (for C^1 algorithms) $[V_k] \subset \mathbb{R}^{n\times n}$, such that $V(t_k, [x_0]) \subset [V_k]$;
- (for C^1 algorithms) $[V_{k+1}] \subset \mathbb{R}^{n\times n}$, such that $V(t_k + h_k, [x_0]) \subset [V_{k+1}]$; and
- (for C^1 algorithms) $[W_3] \subset \mathbb{R}^{n\times n}$ compact and convex, such that $V([0, h_k], [x_k]) \subset [W_3]$

Output:

We compute $[E_k] \subset \mathbb{R}^n$ such that

$$\varphi([0, h_k], [x_k]) \subset [E_k],$$

and for $\mathcal{C}^1$ algorithms $[M_k] \subset \mathbb{R}^{n\times n}$ such that

$$V(t_k + [0, h_k], [x_0]) \subset [M_k].$$

We proceed as follows:

- if $0 \in f_i([W_1])$, then

$$[E_k]_i := \text{hull}([x_k]_i, [x_{k+1}]_i) + [-1, 1] \cdot \frac{h_k^2}{2} \left| \sum_j \frac{\partial f_i}{\partial x_j}([W_1]) \cdot f_j([W_1]) \right|;$$

- if $0 \notin f_i([W_1])_i$, then the ith coordinate is strictly monotone on $[W_1]$, hence we set

$$[E_k]_i = \text{hull}([x_k]_i, [x_{k+1}]_i).$$

For C^1 algorithms
We define

$$[W_4] = [W_3] \cdot [V_k]. \tag{63}$$

It is easy to see that $V(t_k + [0, h_k], [x_0]) \subset [W_4]$:

- if $V'_{ij}([W_1], [W_4]) = \sum_l (\partial f_i / \partial x_l)([W_1]) \cdot [W_4]_{lj}$ contains 0, then

$$\begin{aligned} [M_k]_{ij} \;=\; & \text{hull}([V_k]_{ij}, [V_{k+1}]_{ij}) \\ & + [-1, 1] \cdot \frac{h_k^2}{2} \left| \sum_{kl} \frac{\partial^2 f_i}{\partial x_k \, \partial x_l}([W_1]) \cdot [W_4]_{kj} \cdot f_l([W_1]) \right. \\ & \left. + \sum_{lm} \frac{\partial f_i}{\partial x_l}([W_1]) \cdot \frac{\partial f_l}{\partial x_m}([W_1]) \cdot [W_4]_{mj} \right|; \end{aligned}$$

- if $V'_{ij}([W_1], [W_4])$ does not contain 0, then $V_{ij}(t_k + t, [x_0])$ is strictly monotone for $t \in [0, h_k]$, hence we can set

$$[M_k]_{ij} = \text{hull}([V_k]_{ij}, [V_{k+1}]_{ij}). \qquad \square$$

5. Poincaré Map

The goal of this section is to show how, from the bounds for $\varphi(t, x)$ and $\partial\varphi/\partial x$ obtained by the $\mathcal{C}^0$ and $\mathcal{C}^1$ Lohner algorithms, we can compute $P(x)$ and $\partial P/\partial x$.

In Section 5.1 we derive the formulas, which express $\partial P/\partial x$ in terms of $\partial\varphi/\partial x$.

In Section 5.2 we discuss in detail the procedure which allows us to compute P and $\partial P/\partial x$ from the data obtained from the Lohner algorithm.

In Section 5.3 we discuss the issue of errors related to the computation of the intersection of the trajectory with a section. Insights gained in this section are then used in Section 5.4 to describe an efficient time-step selection scheme near the section.

Let $\alpha\colon \mathbb{R}^n \to \mathbb{R}$ be a $\mathcal{C}^1$ function. We define a section, Θ, by

$$\Theta = \{x \mid \alpha(x) = 0,\ f(x) \cdot \nabla\alpha(x) > 0\}. \tag{64}$$

We assume that Θ is a local section for (1).

For $x \in \mathbb{R}^n$ by $t_p(x)$ we will denote the return time to Θ. This means that

$$\varphi(t_p(x), x) \in \Theta. \tag{65}$$

The Poincaré map $P : \Theta \supset \mathrm{dom}(P) \to \Theta$ can be seen as a restriction, the section of the map $\tilde{P} : \mathbb{R}^n \supset U \to \mathbb{R}^n$ is given by

$$\tilde{P}(x) = \varphi(t_p(x), x). \tag{66}$$

From now on we will not distinguish between the maps $\tilde{P}$ and P, and we will use the same symbol P to denote both of them. Hence we effectively treat a Poincaré map as a map defined on some open subset in $\mathbb{R}^n$ for two reasons, we want to avoid any particular coordinate system on Θ and we want to allow for maps between the different sections.

To infer the derivatives of $P : \Theta \to \Theta$ we have consider how Θ is embedded in $\mathbb{R}^n$. For example, when $\alpha(x) = x_1 - C$ and we use $(x_2, \ldots, x_n)$ to represent the points on Θ, then we just drop the first row and the first column from the matrix $\partial\tilde{P}/\partial x$.

5.1. *A Formula for Partial Derivatives of the Poincaré Map*

The Poincaré map P is defined by

$$P(x) = \varphi(t_P(x), x), \tag{67}$$

where $t_P\colon \mathbb{R}^n \to \mathbb{R}$ satisfies the following equation

$$\alpha(\varphi(t_p(x), x)) = 0. \tag{68}$$

Since Θ is a local section, it follows easily from the implicit function theorem that t_p is a smooth function. We now compute the derivatives of t_p. We differentiate (68) with respect to x_j, $j = 1, \ldots, n$, to obtain

$$\sum_{i=1}^{n} \frac{\partial\alpha}{\partial x_i}\left(f_i \frac{\partial t_P}{\partial x_j} + \frac{\partial\varphi_i}{\partial x_j}\right) = 0,$$

$$(\nabla\alpha \cdot f)\frac{\partial t_P}{\partial x_j} + \sum_{i=1}^{n} \frac{\partial\alpha}{\partial x_i}\frac{\partial\varphi_i}{\partial x_j} = 0.$$

Hence

$$\frac{\partial t_P}{\partial x_j} = -\frac{1}{\nabla\alpha \cdot f}\sum_{i=1}^{n} \frac{\partial\alpha}{\partial x_i}\frac{\partial\varphi_i}{\partial x_j}. \tag{69}$$

Now we can compute $\partial P_i/\partial x_j$:

$$\frac{\partial \varphi_i(t_P(x), x)}{\partial x_j} = f_i \frac{\partial t_P}{\partial x_j} + \frac{\partial \varphi_i}{\partial x_j} = \frac{\partial \varphi_i}{\partial x_j} - \frac{f_i}{\nabla \alpha \cdot f} \sum_{k=1}^{n} \frac{\partial \alpha}{\partial x_k} \frac{\partial \varphi_k}{\partial x_j}. \tag{70}$$

For example, if $\alpha(x) = x_1 - C$, we obtain

$$\frac{\partial P_i}{\partial x_j} = -\frac{f_i}{f_1} \frac{\partial \varphi_1}{\partial x_j} + \frac{\partial \varphi_i}{\partial x_j}. \tag{71}$$

Hence we have proved the following:

Lemma 8.

$$\begin{aligned} \frac{\partial P_i}{\partial x_j}(x) \;=\; & \frac{\partial \varphi_i}{\partial x_j}(t_p(x), x) - \frac{f_i}{\nabla \alpha \cdot f} \left(\varphi(t_p(x), x)\right) \\ & \cdot \sum_{k=1}^{n} \frac{\partial \alpha}{\partial x_k}(\varphi(t_p(x), x)) \frac{\partial \varphi_k}{\partial x_j}(t_p(x), x). \end{aligned} \tag{72}$$

5.2. *Computation of a Poincaré Map*

In this section we describe how, from the data supplied by the Lohner algorithm, we can obtain $P([x_0])$ and $(\partial P/\partial x)([x])$. In this section we assume that we have a prescribed sequence of time steps h_k. A proposal for an efficient time-step selection scheme near the Poincaré section is given in Section 5.4.

Let us set $t_k = \sum_{i=0}^{k-1} h_i$, where h_i is the time step used in the $(i+1)$th step of the Lohner algorithm.

From the Lohner algorithm we have:

(1) a sequence $[x_{N+k}] \subset \mathbb{R}^n$, $k = 0, \ldots, s$, such that $\varphi(t_{N+k}, [x_0]) \subset [x_{N+k}]$; and
(2) a sequence $[V_{N+k}] \subset \mathbb{R}^{n\times n}$, $k = 0, \ldots, s$, such that $(\partial\varphi/\partial x)(t_{N+k}, [x_0]) \subset [V_{N+k}]$;

such that the following conditions are satisfied

$$\varphi((0, t_N], [x_0]) \cap \Theta \;=\; \emptyset, \tag{73}$$

$$\alpha([x_N]) \;<\; 0, \tag{74}$$

$$\alpha([x_{N+s}]) \;>\; 0. \tag{75}$$

For $k = 0, \ldots, s-1$ we define $[E_{N+k}] \subset \mathbb{R}^n$, $[M_{N+k}] \subset \mathbb{R}^{n\times n}$ as the output of the procedure described in Section 4. We have

$$\varphi([t_{N+k}, t_{N+k+1}], [x_0]) \;\subset\; \varphi([0, h_{N+k}], [x_{N+k}]) \subset [E_{N+k}], \tag{76}$$

$$\frac{\partial \varphi}{\partial x}([t_{N+k}, t_{N+k+1}], [x_0]) \;=\; V([t_{N+k}, t_{N+k+1}], [x_0]) \subset [M_{N+k}]. \tag{77}$$

We set

$$\langle P\rangle \;=\; \bigcup_{k=0}^{s-1}[E_{N+k}], \tag{78}$$

$$\langle V\rangle \;=\; \bigcup_{k=0}^{s-1}[M_{N+k}]. \tag{79}$$

The sums in the above equations are realized by taking the interval enclosures.

We have

$$\varphi([t_N, t_{N+s}], [x_0]) \;\subset\; \langle P\rangle, \tag{80}$$

$$\frac{\partial\varphi}{\partial x}([t_N, t_{N+s}], [x_0]) \;\subset\; \langle V\rangle. \tag{81}$$

Assume now that on $\langle P\rangle$ we have

$$\nabla\alpha(\langle P\rangle)\cdot f(\langle P\rangle) > 0. \tag{82}$$

From the above equation it follows that $\alpha(t) = \alpha(\varphi(t, x))$ is an increasing function for $x \in \langle P\rangle$. Hence from (73)–(75), (80), and (81) it follows immediately that, for all $x \in [x_0]$, the Poincaré map, P, is well-defined and

$$t_P([x_0]) \;\subset\; (t_N, t_{N+s}), \tag{83}$$

$$P([x_0]) \;\subset\; [P] := \langle P\rangle \cap \Theta, \tag{84}$$

$$\frac{\partial\varphi}{\partial x}(t_P(x), x) \;\in\; \langle V\rangle \quad \forall\, x \in [x_0]. \tag{85}$$

Finally, from Lemma 8, it follows that

$$\frac{\partial P_i}{\partial x_j}([x_0]) \subset \langle dP\rangle_{ij} := \langle V\rangle_{ij} - \frac{f_i}{\nabla\alpha\cdot f}([P])\cdot\sum_{k=1}^{n}\frac{\partial\alpha}{\partial x_k}([P])\langle V\rangle_{kj}. \quad \square \tag{86}$$

Let us remind the reader that the map P in the above formula is treated as map $P\colon \mathbb{R}^n \to \mathbb{R}^n$ (see the beginning of Section 5). To infer the derivatives of a Poincaré map $P\colon \Theta \to \Theta$ we have to consider how Θ is embedded in $\mathbb{R}^n$. For example, when $\alpha(x) = x_1 - C$, and we use $(x_2, \ldots, x_n)$ to represent the points on Θ, then we just drop the first row and column from the matrix $\langle dP\rangle$ defined in (86).

5.3. *Estimates for the Section Error*

Let us consider a planar example. We assume that we have a section Θ defined by $\alpha(x) = x_1 - C = 0$, and near Θ the flow is represented by the following system of equations

$$x_1' = 1, \quad x_2' = a, \qquad a \geq 0, \quad a \in \mathbb{R}. \tag{87}$$

It is easy to see that the trajectory of a point (x_1, x_2) where $x_1 < 0$ will intersect the section Θ at the point $P(x_1, x_2) = (0, a|x_1| + x_2)$ and $t_p(x_1, x_2) = |x_1|$.

Let $[x] = [x_1^-, x_1^+] \times [x_2^-, x_2^+]$, where $[x_1^-, x_1^+] < 0$. We have

$$\operatorname{diam}(t_p([x])) = x_1^+ - x_-^1, \tag{88}$$

$$P([x]) = \{0\} \times [a|x_1^+| + x_2^-, a|x_1^-| + x_2^+]. \tag{89}$$

Now consider the computation of $P([x])$ using the method described in Section 5.2. To this end assume that $[x_N] = [x]$ and the sum in (67) is realized as the interval enclosure, then we have

$$[x_1^-, x_1^+ + t_{N+s} - t_N] \times [x_2^-, x_2^+ + a(t_{N+s} - t_N)] \subset \langle P \rangle,$$
$$\{0\} \times [x_2^-, x_2^+ + a(t_{N+s} - t_N)] \subset [P].$$

Hence we obtain

$$\operatorname{diam}([P]) - \operatorname{diam}(P([x_N])) = a \cdot ((t_{N+s} - t_N) - \operatorname{diam}(t_P([x_N]))). \tag{90}$$

The above equation gives us the difference between the computed image and the true image of the Poincaré map. We will call this difference *a section error* and we will denote it by ΔS. In general, situation ΔS is an $(n-1)$-dimensional vector and ΔS_i is a section error for the ith coordinate on a section.

A higher-dimensional generalization of the above example leads us to the following formula (for small boxes $[x_N]$ and sections given by $\alpha(x) = x_j - C$) and to the considerations given in Section 4:

$$\Delta S_i \approx \frac{f_i}{f_1} \cdot \left((t_{N+s} - t_N) - \operatorname{diam}(t_p([x_N]))\right) + Ch^2, \tag{91}$$

where f_i/f_1 should be evaluated on $P([x_N])$, h is the time step used close to section Θ, and C is some constant depending on f (see Lemma 6).

From the above formula we see that, in order to minimize the section error, we can do three things:

(1) Take small time steps close to Θ, to minimize the Ch^2 term.
(2) Choose section Θ to be as orthogonal to the flow as possible (to make f_i/f_1 small). In this case it is reasonable to use a different coordinate, y, locally close to section Θ, so that in new coordinates, $\tilde{\alpha}(y) = y_1 - C$, and the flow is very close to the parallel flow $y_1' = \tilde{f}_1(y_1, \ldots, y_n)$, $y_i' = 0$ for $i > 1$.
(3) Minimize $t_{N+s} - t_N$ to make it as close as possible to $t_p([x_N])$. This issue is addressed in Section 5.4.

5.4. *The Choice of Time Steps Close to the Section*

Let us first estimate the diameter of the Poincaré return time, $t_p([x_0]) = t_p([x_N])$. If we assume that $[x_N]$ is small and we are very close to Θ, then we can assume

that $(d/dt)\alpha(\varphi(t,x)) = \nabla\alpha \cdot f(x)$ is approximately constant, while our trajectory is crossing the section. For simplicity we also assume that $\operatorname{diam}(\alpha(\varphi(t,[x_N])))$ is also constant. From this we obtain the following estimate

$$t_p([x_N]) \approx h_C = \frac{\operatorname{diam}(\alpha([x_N]))}{\nabla\alpha \cdot f(x)}, \tag{92}$$

where x is any point in $[x_N]$. The number h_C will be called *the estimated crossing time* in the sequel.

Similar considerations allow us to make nonrigorous, but reasonably good, guesses about the time needed to get very close to section and to the time needed to get past the section, or very close to it.

A TIME-STEP SELECTION SCHEME CLOSE TO THE SECTION

Let $h > 0$, $D > 1$ (we used $D = 12$, $D = 36$ in our implementation), and $\varepsilon > 0$ (this should be a small number, we used $\varepsilon = 10^{-2}$):

(1) We compute $[x_k]$, $[V_k]$, $k = 1, \ldots, N-1$, using the Lohner algorithm with a constant time step $h_k = h$ as long as the equation

$$\varphi([0,h_k],[x_k]) \cap \Theta = \emptyset, \tag{93}$$

is satisfied. For the Nth step we have

$$\varphi([0,h_{N-1}],[x_{N-1}]) \cap \Theta \neq \emptyset. \tag{94}$$

(2) We discard the data obtained for the Nth step. We find a new value for h_{N-1}, such that the Lohner algorithm applied to $[x_{N-1}]$ gives

$$\alpha([x_N]) < 0, \tag{95}$$

and

$$-\varepsilon \operatorname{diam}(\alpha([x_N])) < \operatorname{right}(\alpha([x_N])). \tag{96}$$

Obviously if we are doing a C^1 computation we have to recompute $[V_N]$.

(3) We set $\bar{h} = \min(h/D, h_C)$. This will be our time step, while we cross the section.

(4) With $h_{N+k} = \bar{h}$, $k = 0, \ldots,$ we continue to compute $[x_{N+k}]$ (and $[V_{N+k}]$) until, for $k = s$, we have

$$\alpha([x_{N+s}]) > 0. \tag{97}$$

(5) We check if the following condition holds

$$\operatorname{left}(\alpha([x_{N+s}])) < \varepsilon \cdot \operatorname{diam}([x_{N+s-1}]). \tag{98}$$

If the above condition is not true, then we discard the value of $[x_{N+s}]$ (and $[V_{N+s}]$). We decrease the value of h_{N+s-1} and we recompute $[x_{N+s}]$ (and $[V_{N+s}]$), until (97) and (98) are satisfied. □

Observe that from conditions (96) and (98) it follows that we overestimate the diameter of the Poincaré return time, $t_p([x_N])$, by a factor approximately equal to $1 + 2\varepsilon$.

Point (3) is necessary in the case when $[x_N]$ is relatively large, so that the estimated crossing time becomes unacceptable for us (either the term h^2C in (91) becomes too large or the enclosure procedures cannot be completed). In this case we use the previous time step divided by D. In this case we will cross the section in several time steps. When $[x_N]$ is small we cross the section in one or two steps.

6. Remarks and Discussion

6.1. *A Cost Comparison of C^0 and C^1 Algorithms*

Let us observe that the C^0 and C_2^1 algorithms are identical on the x-variables, i.e., for the same order and $[x_0]$, for the same sequence of time steps h_k, and for the same rough enclosure procedure they return the same estimate for $\varphi(t, [x_0])$. This is not the case for the C_1^1 algorithm because, instead of the partial derivative of the Taylor method, the partial derivative of the flow is used.

With respect to a cost for the same order observe that C^1 algorithms add the following:

(1) the computation of an enclosure $[W_2]$ (for the C_1^1 algorithm, only);
(2) the computation of the error term of the Taylor method for (2) (all derivatives of V up to order $p_v + 1$) (if $p_v \geq p_e$, then this is the most expensive part of the algorithm); and
(3) the rearrangement computations of V.

From the items listed above (1) and (3) appear to be quite cheap when compared to the second one. Hence we expect that the C^1 algorithm with $p_v = p_e = r$ will run in time comparable to the C^0 algorithm with the order $p_e = r - 1$. We tested this on Poincaré maps for the Rössler equation and the Galerkin projection of the Kuramoto–Sivashinsky (KS) equations described in Section 7. These tests show that the computation times of the C^1 algorithm with $p_v = p_e = r$ are usually less that twice the computation time for the C^0 algorithm with the same time step and with the order $p_e = r + 1$. The test was performed over the orders $3, 4, \ldots, 9$.

6.2. *Remarks on a Direct Integration of the System* (1)–(2) *Using the C^0 Lohner Algorithm*

As was mentioned in the Introduction we can do a C^1 computation by simply applying the original Lohner algorithm to the system (1)–(2).

Observe that the dimension of the phase space for this system is $n + n^2$, where n is the dimension appearing in (1). This has an immediate consequence for the

computation time of the Taylor expansion. For example, from the formulas given in the Appendix it follows that for a second degree polynomial and for an order p the computation time has a leading term $C(p)n^4$, where C is constant depending of the order only. Hence one can expect that the proposed algorithm will be about $(n+1)^4$ times faster.

But there is also one even more important issue. It turns out that the Lohner algorithm applied to system (1)–(2) performs quite poorly with respect to controlling the growth of the computed image. For example, we were not able, using this approach, to compute any of the Poincaré maps considered in Section 7. Apparently, the QR-decomposition for system (1)–(2), which plays a crucial role in controling the wrapping effect by choosing a good coordinate frame, becomes dominated very quickly by the variational variables (the V-variables). This produces a huge wrapping effect in the x-variables and leads to meaningless results, i.e., the computed rigorous bounds are so large that they become impractical or the program returns overflow. There is an obvious cure for this problem: do the QR-decomposition separately for the x-variables and for the variational part. This is what we are doing in the proposed C^1 algorithm.

7. Applications—Proofs of an Existence of Periodic Orbits for ODEs

In this section we report on some numerical experiments using the proposed C^1 Lohner algorithm for the computation of a Poincaré map, P, for some ODEs defined by second-degree polynomial vector fields. For this class of ODEs we implemented the C^1_1 Lohner algorithm in $C++$ (we used the *Borland C++* 5.01 compiler). The iterative formulas we used to obtain the Taylor coefficients are given in the Appendix. The computations were performed on a PC Pentium III 733 MHZ machine. The orders p_v and p_e were equal. We always used a fixed time step, h, but close to a Poincaré section we reduced h significantly (see Section 5 for more details).

7.1. *Interval Newton Method for Maps*

The goal of this subsection is to recall the interval version of the Newton method (see [**1**] and the references given there) for finding the zeros of a function $f\colon \mathbb{R}^n \to \mathbb{R}^n$.

The following theorem was proved in [**1, Theorem 1**].

Theorem 9. *Let $f\colon \mathbb{R}^n \to \mathbb{R}^n$ be a C^1 function. Let $X = \prod_{i=1}^n [a_i, b_i]$, $a_i < b_i$. Assume that the interval enclosure of $Df(X)$, $[Df(X)]_I$, is invertible. Let $x_0 \in X$ and we define*

$$N(x_0, X) = -[Df(X)]_I^{-1} f(x_0) + x_0. \tag{99}$$

Then:

(0) *if* $x_1, x_2 \in X$ *and* $f(x_1) = f(x_2)$, *then* $x_1 = x_2$;
(1) *if* $N(x_0, X) \subset X$, *then* $\exists! x^* \in X$ *such that* $f(x^*) = 0$;
(2) *if* $x_1 \in X$ *and* $f(x_1) = 0$, *then* $x_1 \in N(x_0, X)$; *and*
(3) *if* $N(x_0, X) \cap X = \emptyset$, *then* $f(x) \neq 0$ *for all* $x \in X$.

When looking for the fixed point of the map P we set $f = I - P$, hence the set $N(x_0, X)$ will be given by

$$N(x_0, X) = -[I - DP(X)]_I^{-1}(x_0 - P(x_0)) + x_0. \tag{100}$$

We can formulate an interval Newton algorithm to find the fixed points of the map P as follows.

INTERVAL NEWTON ALGORITHM. Let X be a product of intervals. We define the function $N(x_0, X)$ as follows:

$$x_0 = \text{mid}(X), \qquad N(x_0, X) = -[I - DP(X)]_I^{-1}(x_0 - P(x_0)) + x_0. \tag{101}$$

Step 1. Compute $N(x_0, X)$.
Step 2. If $N(x_0, X) \subset X$, then return **success**.
Step 3. If $X \cap N(x_0, X) = \emptyset$, then return **fail**. There are no fixed points in X.
Step 4. If $X \subset N(x_0, X)$, then we modify the computation parameters. For example, decrease a time step, eventually increase the order of the algorithm used to compute P. Go to Step 1.
Step 5. Define a new X by $X := X \cap N(x_0, X)$ and go to Step 1.

7.2. *Rössler Equations*

The Rössler equations [**17**] are given by

$$\begin{aligned} x' &= -(y + z), \\ y' &= x + 0.2y, \\ z' &= 0.2 + z(x - a), \end{aligned} \tag{102}$$

where a is a real parameter. We focus here on the values of $a = 2.2$, where numerical simulations strongly suggest the existence of an attracting limit cycle and $a = 5.7$, where numerical simulations suggest the existence of a strange attractor.

We investigate the Poincaré map on a section $\Pi = \{x = 0, y < 0\}$. In the sequel we will denote this map by P. On Π we will use (y, z) as the coordinates. All data for the Rössler equations presented in Subsections 7.3 and 7.4 are expressed in these coordinates.

7.3. *Rössler Equations for $a = 2.2$, a Proof of an Existence of an Attracting Periodic Orbit*

For $a = 2.2$ numerical simulations suggest the existence of an attracting limit cycle γ, which gives rise to an attracting fixed point, $x_0 \approx (-3.9205, 0.063858)$, for P on Π. The existence of such a periodic orbit (using a topological method based on the Conley index) was recently proved with computer assistance by Pilarczyk [**14**] (the computation time was around 2 hours). The fact that the orbit is actually attracting was not established there.

Using x_0, obtained by an iteration of a nonrigorous Newton method, as the starting location of the fixed point, we verified (for the first iterate of the interval Newton method) that $N(x_0, X) \subset X$, where $X = x_0 + [-2.5 \cdot 10^{-2}, 2.5 \cdot 10^{-2}]^2$. The time step was $h = 10^{-2}$ and the order of the numerical method was $p = 4$.

Below we list some data from this computation

$$\begin{aligned} N(x_0, X) &= x_0 + \{2.287225 \cdot 10^{-8} + [-1.094486 \cdot 10^{-7},\ 1.094486 \cdot 10^{-7}]\} \\ &\quad \times \{-1.273267 \cdot 10^{-10} + [-6.702682 \cdot 10^{-9},\ 6.702682 \cdot 10^{-9}]\} \\ &\subset X. \end{aligned}$$

For $x_0 - P(x_0)$ we obtained

$$\begin{aligned} x_0 - P(x_0) &= [-3.672362 \cdot 10^{-8},\ 2.247709 \cdot 10^{-8}] \\ &\quad \times [-8.317404 \cdot 10^{-10},\ 8.398115 \cdot 10^{-10}] \\ \operatorname{diam}(x_0 - P(x_0)) &= 5.920072 \cdot 10^{-8}. \end{aligned}$$

We obtained the following values for all entries of $DP(X)$:

$$\begin{aligned} DP(X)_{11} &= [-1.786834,\ 4.775295 \cdot 10^{-1}], \\ DP(X)_{12} &= [1.998447,\ 4.929028], \\ DP(X)_{21} &= [-4.231020 \cdot 10^{-2},\ 4.040204 \cdot 10^{-2}], \\ DP(X)_{22} &= [-3.912754 \cdot 10^{-2},\ 6.046568 \cdot 10^{-2}], \\ \operatorname{diam}(DP(X)) &= 2.930582. \end{aligned}$$

The computation times were 5 seconds for $DP(X)$ and 3 seconds for $P(x_0)$.

The reader may notice that the diameter of the computed $DP(X)$ is quite large, hence with these data we cannot establish that the fixed point found is attracting.

It turns out that the increasing order of the method, while keeping the size of X and the time step h constant, still gives the diameter of $DP(X)$ of the same size.

To prove that we have here an attracting fixed point for P one needs either to perform the next iterate of the interval Newton algorithm with the set $N(x_0, X)$ obtained above or to choose a smaller starting set. Below we present an example of such a set.

For the initial $X = x_0 + [-1, 1]^2 \cdot 10^{-6}$, $h = 10^{-2}$, and $p = 4$ we obtain

$$\begin{aligned} N(x_0, X) &\subset x_0 + \{4.546329 \cdot 10^{-9} + [-2.075644 \cdot 10^{-8},\ 2.075644 \cdot 10^{-8}]\} \\ &\quad \times \{-1.358635 \cdot 10^{-11} + [-8.82068 \cdot 10^{-10},\ 8.82068 \cdot 10^{-10}]\} \\ &\subset X. \end{aligned}$$

For $x_0 - P(x_0)$ we obtain

$$\begin{aligned} x_0 - P(x_0) &= ([-3.672362,\ 2.247709] \cdot 10^{-8}) \\ &\quad \times ([-8.317404,\ 8.398115] \cdot 10^{-10}), \\ \operatorname{diam}(x_0 - P(x_0)) &= 5.920072 \cdot 10^{-8}. \end{aligned}$$

The entries of $DP(X)$ are

$$\begin{aligned} DP(X)_{11} &= [-5.568081 \cdot 10^{-1},\ -5.567340 \cdot 10^{-1}], \\ DP(X)_{12} &= [3.377049,\ 3.377150], \\ DP(X)_{21} &= [-2.063501 \cdot 10^{-3},\ -2.061059 \cdot 10^{-3}], \\ DP(X)_{22} &= [1.246689 \cdot 10^{-2},\ 1.247005 \cdot 10^{-2}], \\ \operatorname{diam}(DP(X)) &= 1.019493 \cdot 10^{-4}. \end{aligned}$$

The eigenvalues of $DP(X)$ are given by

$$\begin{aligned} \lambda_1 &= [-9.0097845007 \cdot 10^{-5},\ 7.6326532192 \cdot 10^{-6}], \\ \lambda_2 &= [-5.4431024142 \cdot 10^{-1},\ -5.4421251092 \cdot 10^{-1}]. \end{aligned}$$

Hence the fixed point found is attracting.

7.4. *Rössler Equations for $a = 5.7$, a Proof of the Existence of an Unstable Periodic Orbit*

The goal of this section is to show that the proposed C^1 Lohner algorithm works well enough to obtain the existence of an unstable fixed point.

For the parameter value $a = 5.7$ numerical simulations suggest the existence of a strange attractor. In [**22**] (see also [**23**]) it was proved, with computer assistance, that the Poincaré map, P, has symbolic dynamics on three symbols. In particular, it follows from [**22**] that there exists a fixed point $x^* \in [-9, -7] \times [-0.02, 0.08]$ for P. Using below the C^1 Lohner algorithm and the interval Newton method we find coordinates of this point which much better precision.

For $x_0 \approx (-8.38095, 0.0295902)$, $X = x_0 + [-10^{-3},\ 10^{-3}]^2$, $h = 10^{-2}$, and $p = 4$ we were able to verify that

$$\begin{aligned} N(x_0, X) = x_0 &+ \{-4.327904 \cdot 10^{-8} + [-3.322230 \cdot 10^{-7},\ 3.322230 \cdot 10^{-7}]\} \\ &\times \{-3.937174 \cdot 10^{-8} + [-6.728499 \cdot 10^{-9}, 6.728499 \cdot 10^{-9}]\} \subset X. \end{aligned}$$

For $x_0 - P(x_0)$ we obtained the following values

$$\begin{aligned} x_0 - P(x_0) &= [-1.038311 \cdot 10^{-6},\ 1.176263 \cdot 10^{-6}] \\ &\quad \times [3.310963 \cdot 10^{-8},\ 4.566477 \cdot 10^{-8}], \\ \operatorname{diam}(x_0 - P(x_0)) &= 2.214574 \cdot 10^{-6}. \end{aligned}$$

The computed bounds for $DP(X)$ are given by

$$\begin{aligned} DP(X)_{11} &= [-2.438481,\ -2.372972], \\ DP(X)_{12} &= [1.946089,\ 1.988827], \\ DP(X)_{21} &= [-1.334871 \cdot 10^{-3},\ -8.601484 \cdot 10^{-4}], \\ DP(X)_{22} &= [7.680544 \cdot 10^{-4},\ 1.019349 \cdot 10^{-3}], \\ \operatorname{diam}(DP(X)) &= 6.550938 \cdot 10^{-2}. \end{aligned}$$

The eigenvalues of $DP(X)$ are

$$\begin{aligned} \lambda_1 &= [-3.3231261746 \cdot 10^{-2},\ 3.3213281643 \cdot 10^{-2}], \\ \lambda_2 &= [-2.4380463797,\ -2.3716018363]. \end{aligned}$$

Hence the fixed point found is hyperbolic with one stable and one unstable direction.

7.5. *A 14-Dimensional Galerkin Projection of Kuramoto–Sivashinsky Equations*

The Kuramoto–Sivashinsky equation [**7**], [**18**] (we will use the shorthand KS equation in the sequel), introduced in the context of a wave-front propagation, is given by

$$u_t = -\nu u_{xxxx} - u_{xx} + 2uu_x, \qquad (t, x) \in [0, \infty) \times (-\pi, \pi), \qquad \nu > 0. \tag{103}$$

Assuming odd and periodic boundary conditions the KS equation can be reduced (see [**24**]) to the following infinite system of ODEs for the coefficients of the Fourier expansion of u:

$$\dot{a}_k = k^2(1 - \nu k^2)a_k - k \sum_{n=1}^{k-1} a_n a_{k-n} + 2k \sum_{n=1}^{\infty} a_n a_{n+k}, \qquad k = 1, 2, 3, \ldots. \tag{104}$$

We will refer to the coordinates a_k as *modes*. In this paper we focus on $\nu = 0.127$. For this parameter value, numerical simulations suggest the existence of an attracting limit cycle, whose projection (a_1, a_3) is an ellipse on which the point moves in the clockwise direction.

We focus on the following two sections:

Θ_1: angle $-\pi/4$ in the (a_1, a_3)-plane: $a_1 + a_3 = 0, a_1 > 0, a_1' + a_3' < 0$; and
Θ_2: angle $-5\pi/4$ in the (a_1, a_3)-plane: $a_1 + a_3 = 0, a_1 < 0, a_1' + a_3' > 0$.

On these sections we use the following coordinates: $(c_1, c_2, c_3, c_4, \ldots) = (\sqrt{a_1^2 + a_3^2}, a_2, a_4, a_5, \ldots)$. We will call them *section coordinates*.

Let R be a map which leaves even modes and changes the sign of odd modes: $a_{2k} \to a_{2k}$ and $a_{2k+1} \to -a_{2k+1}$. It is easy to see that R leaves the system (104) invariant (R is the symmetry of the equations) and sections Θ_1 and Θ_2 are mapped one onto another by this symmetry.

Let $P_{1\to 2}$: $\Theta_1 \to \Theta_2$ and $P_{2\to 1}$: $\Theta_2 \to \Theta_1$ denote the Poincaré maps between sections Θ_1 and Θ_2. We have

$$P_{2\to 1} = R P_{1\to 2} R. \tag{105}$$

For the full Poincaré map on section Θ_1 we obtain that

$$P = P_{2\to 1} P_{1\to 2} = R P_{1\to 2} R P_{1\to 2} = (R P_{1\to 2})^2. \tag{106}$$

Hence any fixed point for $R P_{1\to 2}$ is a fixed point for P.

For $\nu = 0.127$ and the dimension of the Galerkin projection $d = 14$ we have the following approximate fixed point for $R P_{1\to 2}$ (expressed in section coordinates):

$$\begin{aligned} x_0 \;=\; & (0.548852,\ 1.32064,\ -0.34417,\ 0.106402,\ 0.0322448,\\ & -0.0153075,\ -0.00196743,\ 0.00166589,\ 2.79272 \cdot 10^{-5},\\ & -0.000147416,\ 1.04171 \cdot 10^{-5},\ 1.11144 \cdot 10^{-5},\ -1.76484 \cdot 10^{-6}). \end{aligned}$$

We started the interval Newton algorithm with the following parameters

$$\begin{aligned} \delta \;&=\; 10^{-5}, \qquad & X &= x_0 + [-\delta, \delta]^{d-1},\\ p \;&=\; 4, & h &= \frac{1}{2d^2(\nu d^2 - 1)} \approx 0.000106773, \end{aligned}$$

and were able to check the assumptions of Theorem 9 and show that the fixed point found is attracting. Below we give some numerical data from these computations:

$$\begin{aligned} N(x_0, X) = x_0 + (&-4.799947 \cdot 10^{-7} + [-2.301085 \cdot 10^{-7},\ 2.301085 \cdot 10^{-7}],\\ &4.638543 \cdot 10^{-7} + [-1.421597 \cdot 10^{-7},\ 1.421597 \cdot 10^{-7}],\\ &-3.642497 \cdot 10^{-7} + [-1.266866 \cdot 10^{-7},\ 1.266866 \cdot 10^{-7}],\\ &-4.326831 \cdot 10^{-8} + [-5.090406 \cdot 10^{-8},\ 5.090406 \cdot 10^{-8}],\\ &7.367633 \cdot 10^{-8} + [-3.519784 \cdot 10^{-8},\ 3.519784 \cdot 10^{-8}],\\ &-2.027772 \cdot 10^{-9} + [-9.687707 \cdot 10^{-9},\ 9.687707 \cdot 10^{-9}], \end{aligned}$$

$$
\begin{aligned}
&-1.034572 \cdot 10^{-8} + [-7.233898 \cdot 10^{-9},\ 7.233898 \cdot 10^{-9}],\\
&1.377686 \cdot 10^{-9} + [-1.960368 \cdot 10^{-9},\ 1.960368 \cdot 10^{-9}],\\
&1.100782 \cdot 10^{-9} + [-1.555556 \cdot 10^{-9},\ 1.555556 \cdot 10^{-9}],\\
&-2.520153 \cdot 10^{-10} + [-5.082655 \cdot 10^{-10},\ 5.082655 \cdot 10^{-10}],\\
&-8.833202 \cdot 10^{-11} + [-7.110429 \cdot 10^{-10},\ 7.110429 \cdot 10^{-10}],\\
&3.422440 \cdot 10^{-11} + [-7.501527 \cdot 10^{-10},\ 7.501527 \cdot 10^{-10}],\\
&9.362639 \cdot 10^{-12} + [-2.636542 \cdot 10^{-9},\ 2.636542 \cdot 10^{-9}]) \subset x_0 + X.
\end{aligned}
$$

The rigorous bound for the matrix norm of $DRP_{1\to 2}(X)$, corresponding to the norm $\|x\|_m = \max_i |x_i|$, is

$$\|DRP_{1\to 2}(X)\|_m = \max_{i=1,\ldots,d-1} \sum_j |DRP_{1\to 2}(X)_{ij}| < 0.82, \tag{107}$$

which shows that the fixed point found is attracting.

In Tables 1 and 2 we list two first rows and columns $DRP_{1\to 2}$ to give the reader a feeling about the sizes of the obtained bounds. The entries $DRP_{1\to 2}(X)_{ij}$ are decaying as i and j increase. For the diameter we have

$$\operatorname{diam}(DRP_{1\to 2}) = 2.538517 \cdot 10^{-1}$$

and it is reached for the (1, 1) entry.

Table 3 contains the computed bounds for $x_0 - RP_{1\to 2}(x_0)$.

The period of the fixed point found is approximately equal to 2.242. The computation times were 1540 seconds (25.7 minutes) and 628 seconds (10.5 minutes) for C^1 and C^0 computations, respectively.

Table 1. Two first rows of $DRP_{1\to 2}(X)$.

j	$DRP_{1\to 2}(X)_{1j}$	$DRP_{1\to 2}(X)_{2j}$
1	$[4.450702,\ 6.989218] \cdot 10^{-1}$	$[-5.042949,\ -4.544209] \cdot 10^{-1}$
2	$[0.8580582,\ 7.208375] \cdot 10^{-2}$	$[6.742856,\ 7.952010] \cdot 10^{-2}$
3	$[-2.103405 \cdot 10^{-2},\ 1.077716 \cdot 10^{-3}]$	$[4.114357 \cdot 10^{-2},\ 4.521749 \cdot 10^{-2}]$
4	$[7.242173 \cdot 10^{-3},\ 1.785252 \cdot 10^{-2}]$	$[-1.489896 \cdot 10^{-2},\ -1.274972 \cdot 10^{-2}]$
5	$[-4.405318 \cdot 10^{-3},\ -2.183634 \cdot 10^{-3}]$	$[4.344733 \cdot 10^{-3},\ 4.762542 \cdot 10^{-3}]$
6	$[-7.412482 \cdot 10^{-4},\ -5.316928 \cdot 10^{-4}]$	$[1.905524 \cdot 10^{-4},\ 2.181210 \cdot 10^{-4}]$
7	$[1.844130 \cdot 10^{-5},\ 1.331995 \cdot 10^{-4}]$	$[-1.752168 \cdot 10^{-4},\ -1.579311 \cdot 10^{-4}]$
8	$[-4.380603 \cdot 10^{-5},\ 8.325807 \cdot 10^{-6}]$	$[3.447287 \cdot 10^{-5},\ 4.318816 \cdot 10^{-5}]$
9	$[1.565435 \cdot 10^{-6},\ 2.055378 \cdot 10^{-5}]$	$[-1.268904 \cdot 10^{-5},\ -9.857161 \cdot 10^{-6}]$
10	$[-3.177638 \cdot 10^{-6},\ 4.527249 \cdot 10^{-6}]$	$[-7.197865 \cdot 10^{-7},\ 2.291789 \cdot 10^{-7}]$
11	$[-3.236594 \cdot 10^{-6},\ 1.487707 \cdot 10^{-6}]$	$[6.155755 \cdot 10^{-7},\ 1.206583 \cdot 10^{-6}]$
12	$[-1.415723 \cdot 10^{-6},\ 1.494335 \cdot 10^{-6}]$	$[-2.936788 \cdot 10^{-7},\ 4.853580 \cdot 10^{-8}]$
13	$[-9.496676 \cdot 10^{-7},\ 1.017148 \cdot 10^{-6}]$	$[-1.434915 \cdot 10^{-7},\ 8.756737 \cdot 10^{-8}]$

Table 2. Two first columns of $DRP_{1\to 2}(X)$.

i	$DRP_{1\to 2}(X)_{i1}$	$DRP_{1\to 2}(X)_{i2}$
1	$[4.450702 \cdot 10^{-1}, 6.989218 \cdot 10^{-1}]$	$[8.580582 \cdot 10^{-3}, 7.208375 \cdot 10^{-2}]$
2	$[-5.042949 \cdot 10^{-1}, -4.544209 \cdot 10^{-1}]$	$[6.742856 \cdot 10^{-2}, 7.952010 \cdot 10^{-2}]$
3	$[3.448295 \cdot 10^{-1}, 4.253800 \cdot 10^{-1}]$	$[-5.718379 \cdot 10^{-2}, -3.723603 \cdot 10^{-2}]$
4	$[2.070086 \cdot 10^{-2}, 1.004633 \cdot 10^{-1}]$	$[5.415046 \cdot 10^{-3}, 2.537533 \cdot 10^{-2}]$
5	$[-9.715707 \cdot 10^{-2}, -6.493140 \cdot 10^{-2}]$	$[2.745561 \cdot 10^{-3}, 1.070263 \cdot 10^{-2}]$
6	$[-8.989916 \cdot 10^{-3}, 8.344327 \cdot 10^{-3}]$	$[-5.325052 \cdot 10^{-3}, -1.055837 \cdot 10^{-3}]$
7	$[6.416967 \cdot 10^{-3}, 1.640791 \cdot 10^{-2}]$	$[-1.843014 \cdot 10^{-3}, 6.005894 \cdot 10^{-4}]$
8	$[-2.897343 \cdot 10^{-3}, 6.205670 \cdot 10^{-4}]$	$[2.720738 \cdot 10^{-5}, 8.916281 \cdot 10^{-4}]$
9	$[-2.582180 \cdot 10^{-3}, 2.064264 \cdot 10^{-4}]$	$[-3.203044 \cdot 10^{-4}, 3.720348 \cdot 10^{-4}]$
10	$[-2.153901 \cdot 10^{-4}, 6.846829 \cdot 10^{-4}]$	$[-1.697652 \cdot 10^{-4}, 6.874759 \cdot 10^{-5}]$
11	$[-5.261595 \cdot 10^{-4}, 7.236297 \cdot 10^{-4}]$	$[-1.706955 \cdot 10^{-4}, 1.737552 \cdot 10^{-4}]$
12	$[-6.866004 \cdot 10^{-4}, 6.194646 \cdot 10^{-4}]$	$[-1.857865 \cdot 10^{-4}, 1.971323 \cdot 10^{-4}]$
13	$[-2.308010 \cdot 10^{-3}, 2.286113 \cdot 10^{-3}]$	$[-6.751461 \cdot 10^{-4}, 6.790530 \cdot 10^{-4}]$

We performed a similar computation for $d = (7, 8, 9)$-dimensional Galerkin projection and each time we obtained a proof of the existence of an attracting fixed point. For a comparison, for $d = 7$, the computation times for $h = 5 \cdot 10^{-4}$, $\delta = 10^{-5}$ were 52 and 25 seconds for C^1 and C^0 computations, respectively.

8. Conclusions and Future Directions

It is quite obvious that we can hope to improve the algorithm proposed here, in terms of bounds produced at a given computation cost, by constructing much better rough enclosures (by using a low-order Taylor method (see e.g., [**8**])) or by introducing a smart time-step selection scheme.

Table 3. The bounds for $x_0 - RP_{1\to 2}(x_0)$.

i	$x_{0,i} - RP_{1\to 2}(x_0)_i$
1	$[2.100324 \cdot 10^{-7}, 2.120372 \cdot 10^{-7}]$
2	$[-2.094023 \cdot 10^{-7}, -2.085218 \cdot 10^{-7}]$
3	$[1.639110 \cdot 10^{-7}, 1.647684 \cdot 10^{-7}]$
4	$[1.873067 \cdot 10^{-8}, 1.925487 \cdot 10^{-8}]$
5	$[-3.372191 \cdot 10^{-8}, -3.350485 \cdot 10^{-8}]$
6	$[8.123026 \cdot 10^{-10}, 8.859718 \cdot 10^{-10}]$
7	$[4.608611 \cdot 10^{-9}, 4.645857 \cdot 10^{-9}]$
8	$[-5.926317 \cdot 10^{-10}, -5.824013 \cdot 10^{-10}]$
9	$[-4.717564 \cdot 10^{-10}, -4.629766 \cdot 10^{-10}]$
10	$[1.055226 \cdot 10^{-10}, 1.099981 \cdot 10^{-10}]$
11	$[3.324677 \cdot 10^{-11}, 4.100305 \cdot 10^{-11}]$
12	$[-1.666509 \cdot 10^{-11}, -1.160854 \cdot 10^{-11}]$
13	$[-6.625590 \cdot 10^{-12}, 1.340692 \cdot 10^{-12}]$

But it appears to us that major improvements can be obtained if we are able to control the Lipschitz part of the error (as was discussed in the Introduction). Another possible huge improvement should be possible if we can evaluate the Taylor formula (or any other formula used as a numerical method) in a way which will allow for as many cancellations of the various terms as possible, so that the computed range using interval arithmetic will be close to the ideal one. To explain what we mean let us consider the computation of e^{-x} using a series expansion

$$e^{-x} = 1 - x + x^2/2! - x^3/3! + \cdots. \tag{108}$$

Assume that we want to compute $e^{-[0,h]}$ for some $h > 0$. It is easy to see that the direct interval evaluation of (108) will give us an interval containing $[e^{-h}, \cosh(h)]$, because to compute a possible upper bound we insert $x = 0$ for the odd powers and $x = h$ for the even powers. We can imagine that much worse things may happen in a higher-dimensional situation, when we evaluate the higher-order terms of the Taylor method, which are complicated expressions, on a rough enclosure which, in many applications, can have a considerable diameter. An example of such an approach, which deals adequately with this problem for the Euler method, was presented in [**19**], [**20**], where the monotonicity properties of the vector field were exploited.

Let us finish by pointing out two important directions, in the context of the application of rigorous numerics to the dynamics of ODEs and partial differential equations (PDEs), which apparently require the development of new efficient algorithms.

Bifurcation of periodic orbits for ODEs. The C^1 algorithm developed here should allow us to rigorously trace branches of periodic orbits for ODEs, but to treat a rigorous bifurcation point one needs some C^2 information, which clearly requires *an efficient C^2 algorithm for ODEs*.

The dynamics of dissipative PDEs. The dynamics of many important PDEs (e.g., the Navier–Stokes equations) is effectively finite-dimensional (see, e.g., [**24**] and the references given there). The tools developed in [**24**] promise that we can exploit this finite-dimensionality to perform rigorous computations for these equations. To this end we need to develop *an efficient algorithm for solving ODEs with a controlled perturbation*.

9. Appendix. Iterative Formulas for the Taylor Method for a Second Degree Polynomial

Consider the ODE

$$x' = f(x), \qquad x \in \mathbb{R}^n, \quad f \text{ is a polynomial of degree 2}, \tag{109}$$

and let $\varphi(t, x_0)$ be a local flow induced by (109).

Theorem 10. *Let* $x^{(s)} = x^{(s)}(t, x_0) = (d^s/dt^s)\varphi(t, x_0)$, *then*

$$x_i^{(1)} = f_i, \tag{110}$$

$$x_i^{(2)} = \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} x_j^{(1)}, \tag{111}$$

$$x_i^{(r)} = \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} x_j^{(r-1)} + \sum_{j,k=1}^{n} \frac{\partial^2 f_i}{\partial x_j x_k} a_{rkj} \qquad \text{if} \quad r > 2, \tag{112}$$

where

$$a_{rkj} = \left\{ \sum_{s=0}^{r-3} \binom{r-2}{s} x_k^{(r-2-s)} x_j^{(1+s)} \right\}. \tag{113}$$

In the above formulas the functions f_i, $\partial f_i/\partial x_j$, *and* $\partial^2 f_i/\partial x_j x_k$ *are all evaluated at* $x = \varphi(t, x_0)$.

Proof. The formula for $r \leq 3$ is obviously satisfied. So suppose that the formula for $r-1$ holds and we will show it for r. The term involving $\partial f_i/\partial x_j$ is obviously correct.

Consider now the coefficient of the term $x_k^{(r-2-s)} x_j^{(1+s)}$. If $s = 0$, then this coefficient may result only from the differentiation of $x_k^{(r-3)} x_j^{(1)}$ which was multiplied by $\binom{r-3}{0} = \binom{r-2}{0}$.

For $0 < s \leq r-1-3$ we have two possibilities of obtaining $x_k^{(r-2-s)} x_j^{(1+s)}$: we differentiate either x_k or x_j. This gives us the coefficient $\binom{r-3}{s} + \binom{r-3}{s-1} = \binom{r-2}{s}$.

For $s = r-3$ we have the coefficient $\binom{r-3}{r-4}$ from the last term in the sum over s and we have to add to it $1 = \binom{r-3}{r-3}$ which arises from the differentiation of $(\partial f_i/\partial x_j) x_j^{(r-2)}$. Hence we obtain a coefficient $\binom{r-2}{r-3}$. □

By taking partial derivatives of $x_i^{(r)}(t, x_0)$ we easily obtain the following:

Theorem 11. *Let* $D_b x_i^{(r)} = D_b x_i^{(r)}(t, x) := (\partial x_i^{(r)}/\partial x_b)(t, x)$. *We have*

$$D_b x_i^{(1)} = \sum_{k=1}^{n} \frac{\partial f_i}{\partial x_k} D_b x_k^{(0)}, \tag{114}$$

$$D_b x_i^{(2)} = \sum_{j,k=1}^{n} \frac{\partial^2 f_i}{\partial x_j \, \partial x_k} x_j^{(1)} D_b x_k^{(0)} + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} D_b x_j^{(1)}, \tag{115}$$

and, for $r > 2$,

$$D_b x_i^{(r)} = \sum_{j,k=1}^{n} \frac{\partial^2 f_i}{\partial x_j \, \partial x_k} x_j^{(r-1)} D_b x_k^{(0)} + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} D_b x_j^{(r-1)} + \sum_{j,k=1}^{n} \frac{\partial^2 f_i}{\partial x_j x_k} D_b a_{rkj},$$

where

$$\begin{aligned} a_{rkj} &= \sum_{s=0}^{r-3} \binom{r-2}{s} x_k^{(r-2-s)} x_j^{(1+s)}, \\ D_b a_{rkj} &= \sum_{s=0}^{r-3} \binom{r-2}{s} \{D_b(x_k^{(r-2-s)}) \cdot x_j^{(1+s)} + x_k^{(r-2-s)} \cdot D_b(x_j^{(1+s)})\}. \end{aligned}$$

In the above formulas the functions f_i, $\partial f_i / \partial x_j$, *and* $\partial^2 f_i / \partial x_j x_k$ *are all evaluated at* $\varphi(t, x)$.

Observe that $V_{ib}(t, x) = (\partial \varphi_i / \partial x_b)(t, x)$ is in the notation used in the above theorem equal to $D_b x_i^{(0)}(t, x)$.

It is clear from the expressions given in Theorems 10 and 11, that to compute the functions $x^{(s)}$ and $D_b x^{(s)}$, it is enough to specify $x^{(0)} = \varphi(t, x_0)$ and $D_b x_i^{(0)} = (\partial \varphi_i / \partial x_b)(t, x_0)$.

The functions $(d^i / dt^i) f$ and $(d^i / dt^i)\,((\partial f / \partial x) V)$, used in Section 2, are given by

$$\begin{aligned} \left(\frac{d^s}{dt^s} f \right)(x_0) &= x^{(s+1)}(x_0), \\ \left(\frac{d^s}{dt^s} \left(\frac{\partial f}{\partial x} V \right) \right)_{ib} (x_0, V_0) &= D_b x_i^{(s+1)}(x_0, V_0), \end{aligned}$$

where $x_0 \in \mathbb{R}^n$ and $V_0 \in \mathbb{R}^{n \times n}$.

The derivative of the Taylor method of order p and time step h is given by

$$\begin{aligned} \frac{\partial}{\partial x_b} \Phi_i(h, x, p) &= \delta_{ib} + \sum_{s=1}^{p} \left(\frac{d^{s-1}}{dt^{s-1}} \left(\frac{\partial f}{\partial x} V \right) \right)_{ib} (x, \mathrm{Id}) \frac{h^s}{s!} \\ &= \delta_{ib} + \sum_{s=1}^{p} D_b x^{(i)}(x, \mathrm{Id}) \frac{h^s}{s!}, \end{aligned}$$

where δ_{ij} is a Kronecker symbol: $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise.

Acknowledgments

Research supported in part by Polish KBN Grant 2 P03A 011 18 and by NSF Grant DMS-9706903.

References

[1] G. Alefeld, Inclusion methods for systems of nonlinear equations—the interval Newton method and modifications, in *Topics in Validated Computations* (J. Herzberger, ed.), Elsevier Science, New York, 1994.

[2] G. Arioli and P. Zgliczyński, Symbolic dynamics for the Hénon–Heiles Hamiltonian on the critical energy level, *J. Differential Equations* **171** (2001), 173–202.

[3] G. Arioli and P. Zgliczyński, in preparation.

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

[5] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations* I, *Nonstiff Problems*, Springer-Verlag, Berlin, 1987.

[6] B. Hassard, J. Zhang, S. Hastings, and W. Troy, A computer proof that the Lorenz equations have "chaotic" solutions, *Appl. Math. Lett.*, **7** (1994), 79–83.

[7] Y. Kuramoto and T. Tsuzuki, Persistent propagation of concentration waves in dissipative media far from thermal equilibrium, *Progr. Theoret. Phys.* **55** (1976), 365.

[8] R. J. Lohner, Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems, in *Computational Ordinary Differential Equations* (J. R. Cash and I. Gladwell, eds.), Clarendon Press, Oxford, UK, 1992.

[9] R. J. Lohner, Einschliessung der Lösung gewonhnlicher Anfangs- and Randwertaufgaben und Anwendungen, Universität Karlruhe (TH), these, 1988.

[10] R. E. Moore, *Interval Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1966.

[11] M. Mrozek and P. Zgliczyński, Set arithmetic and the enclosing problem in dynamics, *Ann. Polon. Math.* **LXXIV** (2000), 237–259.

[12] K. Mischaikow and M. Mrozek, Chaos in the Lorenz equations: A computer-assisted proof. Part II: Details, *Math. Comp.* **67** (1998), 1023–1046.

[13] N. S. Nedialkov and 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, in *Developments in Reliable Computing* (Budapest, 1998), Kluwer Academic, Dordrecht, 1999, pp. 289–310.

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

[15] T. Rage, A. Neumaier, and C. Schlier, Rigorous verification of chaos in a molecular model, *Phys. Rev. E* **50** (1994), 2682–2688.

[16] L. B. Rall, *Automatic Differentiation*: *Techniques and Applications*, Lecture Notes in Computer Science, Vol. 120, Springer-Verlag, Berlin, 1981.

[17] O. E. Rössler, An equation for continous chaos, *Phys. Lett. A* **57** (1976), 397–398.

[18] G. I. Sivashinsky, Nonlinear analysis of hydrodynamical instability in laminar flames—1. Derivation of basic equations, *Acta Astronom.* **4** (1977), 1177.

[19] W. Tucker, Lorenz attractor exists, *C. R. Acad. Sci. Paris Sér I* **328** (1999), 1197–1202.

[20] W. Tucker, A rigorous ODE solver and Smale's 14th problem, *Found. Comp. Math.* **2** (2002), 53–117.

[21] D. Wilczak and P. Zgliczyński, Heteroclinic connections between periodic orbits in planar restricted circular three-body problem—A computer-assisted proof, submitted, http://www.im.uj.edu.pl/˜zgliczyn

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

[23] P. Zgliczyński, Multidimensional perturbations of one-dimensional maps and stability of Sharkovskii ordering, *Internat. J. Bifurcation Chaos* **9** (1999), 1867–1876.

[24] P. Zgliczyński and K. Mischaikow, Rigorous numerics for partial differential equations: The Kuramoto–Sivashinsky equation, *Found. Comput. Math.* **1** (2001), 255–2889.