Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation

by

Nedialko Stoyanov Nedialkov

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation

Nedialko Stoyanov Nedialkov

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

1999

Compared to standard numerical methods for initial value problems (IVPs) for ordinary differential equations (ODEs), validated (also called interval) methods for IVPs for ODEs have two important advantages: if they return a solution to a problem, then (1) the problem is guaranteed to have a unique solution, and (2) an enclosure of the true solution is produced.

To date, the only effective approach for computing guaranteed enclosures of the solution of an IVP for an ODE has been interval methods based on Taylor series. This thesis derives a new approach, an interval Hermite-Obreschkoff (IHO) method, for computing such enclosures.

Compared to interval Taylor series (ITS) methods, for the same order and stepsize, our IHO scheme has a smaller truncation error and better stability. As a result, the IHO method allows larger stepsizes than the corresponding ITS methods, thus saving computation time. In addition, since fewer Taylor coefficients are required by IHO than ITS methods, the IHO method performs better than the ITS methods when the function for computing the right side contains many terms.

The stability properties of the ITS and IHO methods are investigated. We show as an important by-product of this analysis that the stability of an interval method is determined not only by the stability function of the underlying formula, as in a standard

method for an IVP for an ODE, but also by the associated formula for the truncation error.

This thesis also proposes a Taylor series method for validating existence and uniqueness of the solution, a simple stepsize control, and a program structure appropriate for a large class of validated ODE solvers.

Acknowledgements

My special thanks to Professors Ken Jackson and George Corliss. Ken was my supervisor during my Ph.D. studies. He had many valuable suggestions and was always available to discuss my work. George helped us to get started in this area, read many of my drafts, and constantly encouraged me. Although he could not be officially a member of my committee, I consider him as such.

I want to thank my committee members: Professors Christina Christara, Wayne Enright, Rudi Mathon, and Tom Fairgrieve for their helpful suggestions and prompt reading of my proposals. Thanks to Professor Luis Seco for his participation as an external committee member during my senate exam.

I am thankful to my external examiner Dr. John Pryce. His comments and questions forced me to understand even better some of the issues in validated ODE solving.

I must acknowledge Ole Stauning, Ron van Iwaarden, and Wayne Hayes. Ole and Ron provided two different interval automatic differentiation packages, which helped me to move on quickly in my software development and numerical experiments. Wayne had many good comments on the material and the validated solver that I am developing.

The late Professor Tom Hull is an unfading presence in my life.

I am grateful to my spouse Heidi for her belief in me, patience, and support.

My son Stoyan brought happiness to my work. He has already expressed interest in my thesis, but realizes he must grow up before he understands it.

I am grateful to my parents for encouraging my endless studies.

I gratefully acknowledge the financial support from the Department of Computer Science at the University of Toronto.

Contents

List of Algorithms

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Initial Value Problem

We consider the set of autonomous initial value problems (IVPs)

$$y'(t) = f(y) \tag{1.1.1}$$

$$y(t_0) \in [y_0], \tag{1.1.2}$$

where $t \in [t_0, T]$ for some $T > t_0$. Here t_0 and $T \in \mathbb{R}$, $f \in C^{k-1}(\mathcal{D})$, $\mathcal{D} \subseteq \mathbb{R}^n$ is open, $f : \mathcal{D} \to \mathbb{R}^n$, and $[y_0] \subseteq \mathcal{D}$. The condition (1.1.2) permits the initial value $y(t_0)$ to be in an interval, rather than specifying a particular value. We assume that the representation of f contains only a finite number of constants, variables, elementary operations, and standard functions. Since we assume $f \in C^{k-1}(\mathcal{D})$, we exclude functions that contain, for example, branches, abs, or min. For expositional convenience, we consider only autonomous systems. This is not a restriction of consequence since a nonautonomous system of ordinary differential equations (ODEs) can be converted into an autonomous system. Moreover, the methods discussed here can be extended easily to nonautonomous systems.

We consider a grid $t_0 < t_1 < \cdots < t_m = T$, which is not necessarily equally spaced, and denote the stepsize from t_j to t_{j+1} by h_j ($h_j = t_{j+1} - t_j$). The step from t_j to t_{j+1}

is referred to as the $(j+1)$st step. We denote the solution of (1.1.1) with an initial condition y_j at t_j by $y(t; t_j, y_j)$. For an interval, or an interval vector in general, $[y_j]$, we denote by $y(t; t_j, [y_j])$ the set of solutions

$$\{y(t; t_j, y_j) \mid y_j \in [y_j]\}.$$

Our goal is to compute interval vectors, $[y_j]$, $j = 1, 2, \ldots, m$, that are guaranteed to contain the solution of (1.1.1–1.1.2) at $t_1, t_2, \ldots, t_m$. That is,

$$y(t_j; t_0, [y_0]) \subseteq [y_j], \quad \text{for } j = 1, 2, \ldots, m.$$

Standard numerical methods for IVPs for ODEs attempt to compute an approximate solution that satisfies a user-specified tolerance. These methods are usually robust and reliable for most applications, but it is possible to find examples for which they return inaccurate results. On the other hand, if a validated method (also called an interval method) for IVPs for ODEs returns successfully, it not only produces a guaranteed bound for the true solution, but also verifies that a unique solution to the problem exists.

There are situations when guaranteed bounds are desired or needed. For example, a guaranteed bound of the solution could be used to prove a theorem [68]. Also, some calculations may be critical to the safety or reliability of a system. Therefore, it may be necessary or desirable to ensure that the true solution is within the computed bounds.

One reason why validated solutions to IVPs for ODEs have not been popular in the past is that their computation typically requires considerably more time and memory than the computation of standard methods. However, now that "chips are cheap", it seems natural to shift the burden of determining the reliability of a numerical solution from the user to the computer.

In addition, there are situations where interval methods for IVPs for ODEs may not be computationally more expensive than standard methods. For example, many ODEs arising in practical applications contain parameters. Often these parameters cannot be measured exactly, but are known to lie in certain intervals, as for example, in economic models or in robot control problems. In these situations, a user might want to compute solutions for ranges of parameters. If a standard numerical method is used, it has to be executed many times with different parameters, while an interval method can easily "capture" all the solutions at essentially no extra cost.

Significant developments in the area of validated solutions of IVPs for ODEs are the interval methods of Moore [48], [49], [50], Krückeberg [40], Eijgenraam [19], and Lohner [1], [44], [46]. All these methods are based on Taylor series. One reason for the popularity of the Taylor series approach is the simple form of the error term. In addition, the Taylor series coefficients can be readily generated by automatic differentiation, the order of the method can be changed easily by adding or deleting Taylor series terms, and the stepsize can be changed without doing extra work for recomputing Taylor series coefficients.

Usually, Taylor series methods for IVPs for ODEs are one-step methods, where each step consists of two phases: (1) validate existence and uniqueness of the solution with some stepsize, and (2) compute a tight enclosure for the solution. An algorithm to validate the existence of a unique solution typically uses the Picard-Lindelöf operator and the Banach fixed-point theorem. The computation of a tight enclosure is usually based on Taylor series plus remainder, the mean-value theorem, and various interval transformations.

The main difficulty in the first phase is how to validate existence and uniqueness with a given stepsize. The constant enclosure method [19] is the most commonly used method for validation [44], [69]. However, the stepsizes allowed by this method are restricted to "Euler steps"; thus, reducing the efficiency of any method using it. The main obstacle in the second phase is how to reduce the so-called "wrapping effect." Currently, Lohner's QR-factorization method is the standard scheme for reducing it.

Recently, Berz and Makino [7] proposed a method based on high-order Taylor series expansions with respect to time and the initial conditions that substantially reduces the wrapping effect (see also [8]). Their approach uses Taylor polynomials with real floating-point coefficients and a guaranteed error bound for the remainder term. Then, the arithmetic operations and standard functions are executed with such Taylor polynomials as operands, thus establishing a functional dependency between initial and final conditions. This dependency can be used to reduce the wrapping effect.

1.2 Contributions

This thesis makes the following contributions to the area of computing guaranteed bounds on the solution of an IVP for an ODE.

Method Development

- Taylor series has been the only effective approach for implementing interval methods for IVPs for ODEs. We have developed an interval Hermite-Obreschkoff (IHO) method for computing tight enclosures of the solution. Validated methods based on the Hermite-Obreschkoff formula [28], [55], [56] have not been derived or considered before. Although explicit Taylor series methods can be viewed as a special case of the more general Hermite-Obreschkoff methods, the method we propose is an implicit method with predictor and corrector phases.

- We have devised a method for validating existence and uniqueness of the solution based on the Taylor series approach proposed by Moore [50] and revisited by Corliss and Rihm [13]. While the underlying idea is not new, there has not been an implementation in the framework of a complete validated ODE solver, with a good method for computing tight enclosures.

- We suggest a simple stepsize control strategy and a structure of a program for computing validated solutions of IVPs for ODEs. This structure combines algorithms for validation, computing a tight enclosure, and selecting a stepsize. However, the methods we propose are still constant order.

Theoretical and Empirical Studies

We have studied and compared, both theoretically and empirically, our new interval Hermite-Obreschkoff method with the Taylor series based interval methods.

- We show that compared with ITS methods, for the same stepsize and order, our IHO scheme has a smaller truncation error, better stability, and may be less expensive for many problems, particularly when the code list of $f(y)$ contains many arithmetic operations and elementary functions.

- We believe that we have made an important step towards a better understanding of the stability of interval methods for IVPs for ODEs. We show that the stability of the ITS and IHO methods depends not only on the stability function of the underlying formula, as in the standard numerical methods for IVPs for ODEs, but also on the associated formula for the truncation error. In standard numerical methods, Hermite-Obreschkoff methods are known to be suitable for stiff systems [22], [24], [77], [78], but in the interval case, they still have a restriction on the stepsize. To develop an interval method for stiff problems, we need not only a stable formula for advancing the step, but also a stable associated formula for the truncation error.

- We have shown empirically that a solver with a Taylor series method for validating existence and uniqueness of the solution can reduce the total number of steps, compared to the constant enclosure method used in the past.

Software Development

- We have implemented an object-oriented design of a validated solver, called VNODE (Validated Numerical ODE), for IVPs for ODEs. This design embodies the current developments in object-oriented technology for numerical software. The VNODE package incorporates different techniques used in validated ODE solving in a systematic and flexible way. The structure of VNODE is modular; thus, allowing us to change the code easily and to experiment conveniently with different methods.

1.3 Thesis Outline

An outline of this thesis follows.

Chapter 2 contains background material that we need later. We introduce interval-arithmetic operations on real intervals, interval vectors, and interval matrices. We also define interval-valued functions, interval integration, and discuss a method for efficient generation of Taylor series coefficients.

In Chapter 3, we briefly survey Taylor series methods for computing guaranteed bounds on the solution of an IVP for an ODE. We consider the constant enclosure method for validating existence and uniqueness, explain the wrapping effect, and describe Lohner's algorithm for computing a tight enclosure of the solution. We also discuss the wrapping effect in generating Taylor coefficients and the overestimation in one step of interval Taylor series methods.

In Chapter 4, we derive the interval Hermite-Obreschkoff method for computing a tight enclosure of the solution and give an algorithmic description of this method. Then, we study it theoretically in the constant coefficient and general cases and compare it with interval Taylor series methods. We also discuss the stability of these methods.

Chapter 5 presents a Taylor series method for validating existence and uniqueness of the solution.

Chapter 6 discusses estimating and controlling the overestimation of the interval containing the solution in the methods considered in this thesis and proposes a simple stepsize control.

Chapter 7 describes the structure of a program that incorporates algorithms for validating existence and uniqueness, computing a tight enclosure of the solution, and selecting a stepsize.

Chapter 8 contains numerical results. First, we compare the IHO method with ITS methods on both constant coefficient and nonlinear problems. Then, we show numerical results comparing these methods when the validation phase uses constant enclosure and Taylor series methods.

Conclusions and directions for further research are given in Chapter 9.

Appendix A provides estimates for the number of arithmetic operations required to generate Taylor coefficients.

Appendix B presents the design of VNODE. First, we discuss the goals that we have set to achieve with the design of VNODE, software issues related to the implementation, and the choice of C++ to implement VNODE. Then, we describe the structure of VNODE and illustrate how it can be used.

Chapter 2

Preliminaries

2.1 Interval Arithmetic

The set of intervals on the real line $\mathbb{R}$ is defined by

$$\mathbb{IR} = \left\{ [a] = [\underline{a}, \bar{a}] \mid \underline{a}, \bar{a} \in \mathbb{R},\ \underline{a} \leq \bar{a} \right\}.$$

If $\underline{a} = \bar{a}$ then $[a]$ is a *thin* interval; if $\underline{a} \geq 0$ then $[a]$ is *nonnegative* ($[a] \geq 0$); and if $\underline{a} = -\bar{a}$ then $[a]$ is *symmetric*. Two intervals $[a]$ and $[b]$ are equal if $\underline{a} = \underline{b}$ and $\bar{a} = \bar{b}$.

Let $[a]$ and $[b] \in \mathbb{IR}$ and $\circ \in \{+, -, *, /\}$. The interval-arithmetic operations are defined [50, pp. 8–9] by

$$[a] \circ [b] = \{x \circ y \mid x \in [a]\,,\ y \in [b]\}\,, \qquad 0 \notin [b] \text{ when } \circ = /,$$

which can be written in the equivalent form (we omit $*$ in the notation):

$$\begin{aligned}
[a] + [b] &= \left[\underline{a} + \underline{b}, \bar{a} + \bar{b}\right], \\
[a] - [b] &= \left[\underline{a} - \bar{b}, \bar{a} - \underline{b}\right], \\
[a]\,[b] &= \left[\min\left\{\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b}\right\},\ \max\left\{\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b}\right\}\right], \\
[a]\,/\,[b] &= [\underline{a}, \bar{a}]\left[1/\bar{b}, 1/\underline{b}\right], \qquad 0 \notin [b]\,.
\end{aligned}$$

We have an inclusion of intervals

$$[a] \subseteq [b] \iff \underline{a} \geq \underline{b} \text{ and } \bar{a} \leq \bar{b}.$$

We also define the following quantities for intervals [50]:

- width $w([a]) = \bar{a} - \underline{a}$;
- midpoint $m([a]) = (\bar{a} + \underline{a})/2$;
- magnitude $|[a]| = \max\{|\bar{a}|, |\underline{a}|\}$.

The interval-arithmetic operations are inclusion monotone. That is, for real intervals $[a]$, $[a_1]$, $[b]$, and $[b_1]$ such that $[a] \subseteq [a_1]$ and $[b] \subseteq [b_1]$, we have

$$[a] \circ [b] \subseteq [a_1] \circ [b_1], \qquad \circ \in \{, -, *, /\}.$$

Although interval addition and multiplication are associative, the distributive law does not hold in general [2, pp. 3–5]. That is, we can easily find three intervals $[a]$, $[b]$, and $[c]$, for which

$$[a]([b] + [c]) \neq [a][b] + [a][c].$$

However, for any three intervals $[a]$, $[b]$, and $[c]$, the subdistributive law

$$[a]([b] + [c]) \subseteq [a][b] + [a][c],$$

does hold. Moreover, there are important cases in which the distributive law

$$[a]([b] + [c]) = [a][b] + [a][c]$$

does hold. For example, it holds if $[b][c] \geq 0$, if $[a]$ is a thin interval, or if $[b]$ and $[c]$ are symmetric.

Some other useful results for interval arithmetic follow. For $[a]$ and $[b] \in \mathbb{IR}$,

$$\begin{aligned} |[a] + [b]| &\leq |[a]| + |[b]|, && (2.1.1)\\ |[a][b]| &= |[a]|\,|[b]|, && (2.1.2)\\ w([a] \pm [b]) &= w([a]) + w([b]), && (2.1.3)\\ w([a][b]) &\geq \max\{|[a]|\,w([b]),\ w([a])\,|[b]|\}, \quad \text{and} && (2.1.4)\\ w([a][b]) &\leq |[a]|\,w([b]) + w([a])\,|[b]| && (2.1.5) \end{aligned}$$

[2, pp. 14–17]. If $[a]$ is symmetric, then

$$w([a][b]) = |[b]|\, w([a]). \tag{2.1.6}$$

From (2.1.4) and (2.1.6), if $[a]$ is a symmetric interval, then

$$w([a][b]) \le w([a'][b]),$$

for any $[a']$ with $w([a']) = w([a])$.

2.2 Interval Vectors and Matrices

By an *interval vector* we mean a vector with interval components. By an *interval matrix* we mean a matrix with interval components. We denote the set of n-dimensional real interval vectors by $\mathbb{IR}^n$ and the set of $n \times m$ real interval matrices by $\mathbb{IR}^{n\times m}$. The arithmetic operations involving interval vectors and matrices are defined by using the same formulas as in the scalar case, except that scalars are replaced by intervals. For example, if $[A] \in \mathbb{IR}^{n\times n}$ has components $[a_{ij}]$, and $[b] \in \mathbb{IR}^n$ has components $[b_k]$, then the components of $[c] = [A][b]$ are given by

$$[c_i] = \sum_{k=1}^{n} [a_{ik}][b_k].$$

An inclusion for interval matrices (and vectors) is defined component-wise by

$$[A] \subseteq [B] \iff [a_{ij}] \subseteq [b_{ij}] \quad \text{(for all } i, j).$$

The *maximum norm* of an interval vector $[a] \in \mathbb{IR}^n$ is given by

$$\|[a]\| = \max_{1\le i\le n} \{|[a_i]|\},$$

and of a matrix $[A]$ by

$$\|[A]\| = \max_{1\le i\le n} \sum_{j=1}^{n} |[a_{ij}]|.$$

We also use the symbol $\|\cdot\|$ to denote the maximum norm of scalar vectors, scalar matrices, and functions.

Let $\mathcal{A}$ and $\mathcal{B} \subset \mathbb{R}^n$ be compact non-empty sets. Let $q(\mathcal{A}, \mathcal{B})$ denote the Hausdorff distance between $\mathcal{A}$ and $\mathcal{B}$:

$$q(\mathcal{A}, \mathcal{B}) = \max\left\{\max_{x\in\mathcal{A}}\min_{y\in\mathcal{B}}\|x-y\|,\ \max_{y\in\mathcal{B}}\min_{x\in\mathcal{A}}\|x-y\|\right\}. \tag{2.2.1}$$

The distance between two intervals $[a]$ and $[b]$ is

$$q([a],[b]) = \max\left\{|\underline{a}-\underline{b}|, |\bar{a}-\bar{b}|\right\}, \tag{2.2.2}$$

and the distance between two interval vectors $[u]$ and $[v] \in \mathbb{IR}^n$ is

$$q([u],[v]) = \max_{1\le i\le n}\left\{q([u_i],[v_i])\right\}. \tag{2.2.3}$$

Let $[A] \in \mathbb{IR}^{n\times m}$. We define the following quantities component-wise for interval matrices (and vectors):

- width

$$w([A]) = \begin{pmatrix} w([a_{11}]) & \dots & w([a_{1m}]) \\ \vdots & & \vdots \\ w([a_{n1}]) & \dots & w([a_{nm}]) \end{pmatrix};$$

- midpoint

$$m([A]) = \begin{pmatrix} m([a_{11}]) & \dots & m([a_{1m}]) \\ \vdots & & \vdots \\ m([a_{n1}]) & \dots & m([a_{nm}]) \end{pmatrix};$$

- magnitude

$$|[A]| = \begin{pmatrix} |[a_{11}]| & \dots & |[a_{1m}]| \\ \vdots & & \vdots \\ |[a_{n1}]| & \dots & |[a_{nm}]| \end{pmatrix}.$$

Addition of interval matrices is associative, but multiplication of interval matrices is not associative in general [53, pp. 80–81]. Also, the distributive law does not hold in general for interval matrices [53, p. 79]. That is, we can easily find $[A] \in \mathbb{IR}^{n\times m}$ and $[B]$ and $[C] \in \mathbb{IR}^{m\times p}$, for which

$$[A]\,([B]+[C]) \neq [A]\,[B]+[A]\,[C]\,.$$

However, for any $[A] \in \mathbb{IR}^{n\times m}$ and $[B]$ and $[C] \in \mathbb{IR}^{m\times p}$, the subdistributive law

$$[A]\,([B]+[C]) \subseteq [A]\,[B]+[A]\,[C] \tag{2.2.4}$$

does hold. Moreover, there are important cases in which

$$[A]\,([B]+[C]) = [A]\,[B]+[A]\,[C]$$

does hold. For example, the distributive law holds if $[b_{ij}]\,[c_{ij}] \geq 0$ (for all i, j), if $[A]$ is a point matrix, or if all components of $[B]$ and $[C]$ are symmetric intervals.

Some other useful results for interval matrices follow. Let $[A]$ and $[B] \in \mathbb{IR}^{n\times n}$. Then

$$\begin{aligned}
|[A]+[B]| &\leq |[A]|+|[B]|\,, && (2.2.5)\\
|[A]\,[B]| &\leq |[A]|\,|[B]|\,, && (2.2.6)\\
w\,([A]\pm[B]) &= w\,([A])+w\,([B])\,, && (2.2.7)\\
w\,([A]\,[B]) &\geq \max\{|[A]|\,w\,([B]),\ w\,([A])\,|[B]|\}\,,\ \text{and} && (2.2.8)\\
w\,([A]\,[B]) &\leq |[A]|\,w\,([B])+w\,([A])\,|[B]| && (2.2.9)
\end{aligned}$$

[2, pp. 125–126]. Let the components of $[B]$ be symmetric intervals. Then

$$\left.\begin{aligned}
&w\,([A]\,[B]) = |[A]|\,w\,([B])\ \text{ and}\\
&w\,([A]\,[B]) \leq w\,([A]\,[B'])
\end{aligned}\right. \tag{2.2.10}$$

for any $[B']$ with $w\,([B']) = w\,([B])$.

Let $[c] \in \mathbb{IR}^n$ be a symmetric vector (all components of $[c]$ are symmetric intervals). Then

$$\begin{aligned}
w\big(([A]\,[B])\,[c])\big) &= |[A]\,[B]|\,w([c])\\
&\leq |[A]|\,|[B]|\,w([c]) = w\big([A]\,([B]\,[c])\big).
\end{aligned} \tag{2.2.11}$$

Throughout this thesis, we assume *exact* real interval arithmetic, as described in this subsection. In floating-point implementation, if one or both end-points of a real interval are not representable (which is often the case), then they must be rounded outward to the closest representable floating-point numbers. Interval arithmetic is often called a machine, or rounded, interval arithmetic. A discussion of its properties can be found in [41].

2.3 Interval-Valued Functions

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function on $\mathcal{D} \subseteq \mathbb{R}^n$. We consider functions whose representations contain only a finite number of constants, variables, arithmetic operations, and standard functions (sin, cos, log, exp, etc.).

We define the *range* of f over an interval vector $[a] \subseteq \mathcal{D}$ by

$$R\,(f;[a]) = \big\{ f(x) \mid x \in [a] \big\}.$$

A fundamental problem in interval arithmetic is to compute an enclosure for $R\,(f;[a])$. We want this enclosure to be as tight as possible. For example, in our work, we are interested in f being the right side of a differential equation. The naive *interval-arithmetic evaluation* of f on $[a]$, which we denote by $f([a])$, is obtained by replacing each occurrence of a real variable with a corresponding interval, by replacing the standard functions with enclosures of their ranges, and by performing interval-arithmetic operations instead of the real operations. In practice, $f([a])$ is not unique, because it depends on how f is evaluated in interval arithmetic. For example, expressions that are mathematically equivalent for scalars, such as $x(y+z)$ and $xy+xz$, may have different values if x, y, and z are intervals. However, since we are interested in the interval-arithmetic evaluation of f on a computer, we can assume that $f([a])$ is uniquely defined by the code list, or computational graph, of f. No matter how $f([a])$ is evaluated, it follows from the inclusion monotone property of the the interval operations that

$$R(f;[a]) \subseteq f([a]).$$

If f satisfies a Lipschitz condition on $\mathcal{D} \subseteq \mathbb{R}^n$, then for any $[a] \subseteq \mathcal{D}$,

$$q(R(f;[a]), f([a])) \leq c_1 \|w([a])\| \tag{2.3.12}$$

for some constant $c_1 \geq 0$ independent of $[a]$, where $q(\cdot,\cdot)$ is defined by (2.2.2), [50, p. 24], [2].

Mean-value form

If $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable on $\mathcal{D} \subseteq \mathbb{R}^n$ and $[a] \subseteq \mathcal{D}$, then for any y and $b \in [a]$,

$$f(y) \in f_M([a], b) = f(b) + f'([a])([a] - b) \tag{2.3.13}$$

[50, p. 47]. The expression $f(b) + f'([a])([a] - b)$ is called the mean-value form of f. Mathematically, f_M is not uniquely defined, but it is uniquely determined by the code list of f' and the choice of b. If, in addition, f' satisfies a Lipschitz condition on $\mathcal{D}$, then for any $[a] \subseteq \mathcal{D}$,

$$q(R(f;[a]), f_M([a], b)) \leq c_2 \|w([a])\|^2$$

for some constant $c_2 \geq 0$ independent of $[a]$, [53, pp. 55–56]. Therefore, the mean-value evaluation is *quadratically convergent* in the sense that the distance between $R(f;[a])$ and $f_M([a], b)$ approaches zero as the square of $\|w([a])\|$, as $\|w([a])\|$ approaches zero.

Similar results apply to functions from $\mathbb{R}^n$ to $\mathbb{R}^n$.

Integration

Let $f : \mathcal{D} \to \mathbb{R}^n$ be a continuous function on $\mathcal{D} \subseteq \mathbb{R}$ and $[a] \subseteq \mathcal{D}$. Then,

$$\int_{\underline{a}}^{\bar{a}} f(t)dt \in (\bar{a} - \underline{a}) f([a]) = w([a]) f([a]). \tag{2.3.14}$$

2.4 Automatic Generation of Taylor Coefficients

Moore [50, pp. 107–130] presents a method for efficient generation of Taylor coefficients. Rall [58] describes in detail algorithms for automatic differentiation and generation of Taylor coefficients. He also considers applications of automatic differentiation, including applications to ordinary differential equations. Two books containing papers and extensive bibliographies on automatic differentiation are [9] and [23].

Since we need point and interval Taylor coefficients, we briefly describe the idea of their recursive generation. Denote the ith Taylor coefficient of $u(t)$ evaluated at some point t_j by

$$(u_j)_i = \frac{u^{(i)}(t_j)}{i!},$$

where $u^{(i)}(t)$ is the ith derivative of $u(t)$. Let $(u_j)_i$ and $(v_j)_i$ be the ith Taylor coefficients of $u(t)$ and $v(t)$ at t_j. It can be shown that

$$\begin{aligned} (u_j \pm v_j)_i &= (u_j)_i \pm (v_j)_i\,, && (2.4.15)\\ (u_j v_j)_i &= \sum_{r=0}^{i} (u_j)_r\,(v_j)_{i-r}\,, \text{ and} && (2.4.16)\\ \left(\frac{u_j}{v_j}\right)_i &= \frac{1}{v_j}\left\{(u_j)_i - \sum_{r=1}^{i}(v_j)_r\left(\frac{u_j}{v_j}\right)_{i-r}\right\}. && (2.4.17)\end{aligned}$$

Similar formulas can be derived for the generation of Taylor coefficients for the standard functions [50, p. 114].

Consider the autonomous differential system

$$y'(t) = f(y), \quad y(t_j) = y_j. \tag{2.4.18}$$

We introduce the sequence of functions

$$\begin{aligned} f^{[0]}(y) &= y, && (2.4.19)\\ f^{[i]}(y) &= \frac{1}{i}\left(\frac{\partial f^{[i-1]}}{\partial y} f\right)(y), \quad \text{for } i \geq 1. && (2.4.20)\end{aligned}$$

Using (2.4.18–2.4.20), the Taylor coefficients of $y(t)$ at t_j satisfy

$$(y_j)_0 = f^{[0]}(y_j) = y_j, \quad \text{and} \tag{2.4.21}$$

$$\begin{aligned}(y_j)_i = f^{[i]}(y_j) &= \frac{1}{i}\left(\frac{\partial f^{[i-1]}}{\partial y} f\right)(y_j) \\ &= \frac{1}{i}\big(f(y_j)\big)_{i-1}, \quad \text{for } i \geq 1,\end{aligned} \tag{2.4.22}$$

where $\big(f(y_j)\big)_{i-1}$ is the $(i-1)$st coefficient of f evaluated at y_j. By using (2.4.15–2.4.17), similar formulas for the Taylor coefficients of the standard functions, and (2.4.22), we can recursively evaluate $(y_j)_i$, for $i \geq 1$. It can be shown that if the number of the arithmetic operations in the code list of f is N, then the number of arithmetic operations required for the generation of k Taylor coefficients is between Nk and $Nk(k-1)/2$, depending on the ratio of additions, multiplications, and divisions in the code list for f, [50, pp. 111–112] (see also Appendix A).

Let $y(t_j) = y_j \in [y_j]$. If we have a procedure to compute the point Taylor coefficients of $y(t)$ and perform the computations in interval arithmetic with $[y_j]$ instead of y_j, we obtain a procedure to compute the interval Taylor coefficients of $y(t)$. We denote the ith interval Taylor coefficient of $y(t)$ at t_j by $[y_j]_i = f^{[i]}([y_j])$.

Chapter 3

Taylor Series Methods for IVPs for ODEs

In most validated methods for IVPs for ODEs, each integration step consists of two phases [52]:

ALGORITHM I: Compute a stepsize h_j and an a priori enclosure $[\tilde{y}_j]$ of the solution such that $y(t; t_j, y_j)$ is guaranteed to exist for all $t \in [t_j, t_{j+1}]$ and all $y_j \in [y_j]$, and

$$y(t; t_j, [y_j]) \subseteq [\tilde{y}_j] \quad \text{for all } t \in [t_j, t_{j+1}] \,.$$

ALGORITHM II: Using $[\tilde{y}_j]$, compute a tighter enclosure $[y_{j+1}]$ of $y(t_{j+1}; t_0, [y_0])$.

Usually, the algorithm to validate the existence of a unique solution uses the Picard-Lindelöf operator and the Banach fixed-point theorem. In Taylor series methods, the computation of a tighter enclosure is based on Taylor series plus remainder, the mean-value theorem, and various interval transformations.

We discuss a constant enclosure method for implementing Algorithm I in §3.1. In §3.2, we present the basis of the ITS methods for implementing Algorithm II, illustrate the wrapping effect, and explain Lohner's method for reducing it. We also consider the wrapping effect in generating interval Taylor coefficients and the overestimation in one step of ITS methods.

Surveys of Taylor series and other interval methods can be found in [4], [14], [15], [51], [54], [60], [70], and [71]. These papers give a "high-level" description of existing methods. A more detailed discussion of Taylor series methods can be found in [52].

3.1 Validating Existence and Uniqueness of the Solution: The Constant Enclosure Method

Suppose that at t_j we have an enclosure $[y_j]$ of $y(t_j; t_0, [y_0])$. In this section, we consider how to find a stepsize $h_j > 0$ and an a priori enclosure $[\tilde{y}_j]$ such that for any $y_j \in [y_j]$

$$y'(t) = f(y), \qquad y(t_j) = y_j \tag{3.1.1}$$

has a unique solution $y(t; t_j, y_j) \in [\tilde{y}_j]$ for $t \in [t_j, t_{j+1}]$.

The constant enclosure method [19, pp. 59–67], [44, pp. 27–31] for validating existence and uniqueness of the solution is based on the application of the Picard-Lindelöf operator

$$(Ty)(t) = y_j + \int_{t_j}^{t} f(y(s))\, ds \tag{3.1.2}$$

to an appropriate set of functions and the Banach fixed-point theorem.

THEOREM 3.1 Banach fixed-point theorem. *Let $\Phi : Y \to Y$ be defined on a complete non-empty metric space Y with a metric $d(\cdot,\cdot)$. Let γ satisfy $0 \le \gamma < 1$, and let*

$$d(\Phi(x), \Phi(y)) \le \gamma\, d(x, y) \tag{3.1.3}$$

for all x and $y \in Y$. Then Φ has a unique fixed-point $y^* \in Y$.

Let $h_j = t_{j+1} - t_j$ and $[\tilde{y}_j]$ be such that

$$y_j + [0, h_j] f([\tilde{y}_j]) \subseteq [\tilde{y}_j] \tag{3.1.4}$$

($y_j \in [y_j]$). Consider the set of continuous functions on $[t_j, t_{j+1}]$ with ranges in $[\tilde{y}_j]$,

$$U = \left\{ u \mid u \in C^0([t_j, t_{j+1}]) \quad \text{and} \quad u(t) \in [\tilde{y}_j] \text{ for } t \in [t_j, t_{j+1}] \right\}.$$

For $\alpha_j > 0$, the exponential norm of a function $u \in C^0[t_j, t_{j+1}]$ is defined by

$$\|u\|_{\alpha_j} = \max_{t \in [t_j, t_{j+1}]} \left(e^{-\alpha_j (t - t_j)} \|u(t)\| \right).$$

The set U is complete in the maximum norm and therefore in the exponential norm.

By applying the Picard-Lindelöf operator (3.1.2) to $u \in U$ and using (3.1.4), we obtain

$$\begin{aligned} v(t) = (Tu)(t) &= y_j + \int_{t_j}^{t} f(u(s))\, ds \\ &\in y_j + \int_{t_j}^{t} f([\tilde{y}_j])\, ds \\ &\subseteq y_j + [0, h_j] f([\tilde{y}_j]) \subseteq [\tilde{y}_j]. \end{aligned} \tag{3.1.5}$$

Since $v(t) \in C^0[t_j, t_{j+1}]$ and $v(t) \in [\tilde{y}_j]$ for all $t \in [t_j, t_{j+1}]$, $v \in U$. Hence, if (3.1.4) holds, T maps U into itself.

Let $L_j = \|\partial f([\tilde{y}_j])/\partial y\|$. It can be shown that the Picard-Lindelöf operator is a contraction on U in the exponential norm with $\alpha_j > L_j$, which implies $\gamma = L_j/\alpha_j < 1$, [19, pp. 66–67] (see also [44, pp. 27–29]).

Therefore, if (3.1.4) holds, and we can compute $\partial f([\tilde{y}_j])/\partial y$, then T has a unique fixed point in U. This fixed point, which we denote by $y(t; t_j, y_j)$, satisfies (3.1.1) and $y(t; t_j, y_j) \in [\tilde{y}_j]$ for $t \in [t_j, t_{j+1}]$. Note that to prove existence and uniqueness of the solution of (3.1.1), we do not have to compute $\gamma < 1$ such that the operator T is a contraction. Note also that in bounding the kth Taylor coefficient over $[\tilde{y}_j]$ in Algorithm II (see §3.2), we evaluate $f^{[k]}([\tilde{y}_j])$. Because of the relation (2.4.22), if we cannot evaluate $\partial f([\tilde{y}_j])/\partial y$, then we are not able to evaluate $f^{[k]}([\tilde{y}_j])$.

Let h_j and $[\tilde{y}_j]$ be such that[1]

$$[\tilde{y}_j^1] = [y_j] + [0, h_j] f\left([\tilde{y}_j]\right) \subseteq [\tilde{y}_j]. \tag{3.1.6}$$

[1]We use superscripts on vectors to indicate different vectors, not powers.

Then (3.1.4) holds for any $y_j \in [y_j]$, and (3.1.1) has a unique solution $y(t; t_j, y_j)$ that satisfies

$$y(t; t_j, y_j) \in [\tilde{y}_j]$$

for all $t \in [t_j, t_{j+1}]$ and all $y_j \in [y_j]$. Furthermore, since $f([\tilde{y}_j^1]) \subseteq f([\tilde{y}_j])$, we have

$$y(t; t_j, y_j) \in [y_j] + [0, h_j] f([\tilde{y}_j^1])$$

for all $t \in [t_j, t_{j+1}]$ and all $y_j \in [y_j]$.

In (3.1.6), we should require $[y_j] \subseteq [\tilde{y}_j]$ and $[y_j] \neq [\tilde{y}_j]$. If $[y_j] = [\tilde{y}_j]$, then (3.1.6) becomes

$$[\tilde{y}_j^1] = [y_j] + [0, h_j] f([y_j]) \subseteq [y_j],$$

which implies either $h_j = 0$ or $f([y_j]) = [0, 0]$. If none of the corresponding endpoints of $[y_j]$ and $[\tilde{y}_j]$ are equal, the stepsize, h_j, can always be taken small enough such that the inclusion in (3.1.6) holds. In some cases, such a stepsize can be taken when some of the endpoints of $[y_j]$ and $[\tilde{y}_j]$ coincide.

The inclusion in (3.1.6) can be easily verified. However, a serious disadvantage of the method is that the stepsize is restricted to Euler steps, even when high-order methods are used in Algorithm II to tighten the a priori enclosure. One can obtain better methods by using polynomial enclosures [45] or more terms in the Taylor series for validation [50, pp. 100–103], [13], [52]. We do not discuss the polynomial enclosure method in this thesis, but propose in Chapter 5 a Taylor series method for validating existence and uniqueness. In §8.4, we show by numerical experiments that our Taylor series method for validation enables larger stepsizes than the constant enclosure method.

3.2 Computing a Tight Enclosure

Suppose that at the $(j+1)$st step we have computed an a priori enclosure $[\tilde{y}_j]$ such that

$$y(t; t_j, [y_j]) \subseteq [\tilde{y}_j], \quad \text{for all } t \in [t_j, t_{j+1}].$$

In this section, we show how to compute in Algorithm II a tighter enclosure $[y_{j+1}] \subseteq [\tilde{y}_j]$, for which $y(t_{j+1}; t_0, [y_0]) \subseteq [y_{j+1}]$.

Consider the Taylor expansion

$$y_{j+1} = y_j + \sum_{i=1}^{k-1} h_j^i f^{[i]}(y_j) + h_j^k f^{[k]}(y; t_j, t_{j+1}), \tag{3.2.1}$$

where $y_j \in [y_j]$ and $f^{[k]}(y; t_j, t_{j+1})$ denotes $f^{[k]}$ with its lth component evaluated at $y(\xi_{jl})$, for some $\xi_{jl} \in [t_j, t_{j+1}]$. If (3.2.1) is evaluated in interval arithmetic with y_j replaced by $[y_j]$, and $f^{[k]}(y; t_j, t_{j+1})$ replaced by $f^{[k]}([\tilde{y}_j])$, we obtain

$$[y_{j+1}] = [y_j] + \sum_{i=1}^{k-1} h_j^i f^{[i]}([y_j]) + h_j^k f^{[k]}([\tilde{y}_j]). \tag{3.2.2}$$

With (3.2.2), we can compute enclosures of the solution, but the width of $[y_j]$ always increases with j, even if the true solution contracts. This follows from property (2.1.3) applied to (3.2.2),

$$w([y_{j+1}]) = w([y_j]) + \sum_{i=1}^{k-1} h_j^i w\left(f^{[i]}([y_j])\right) + h_j^k w\left(f^{[k]}([\tilde{y}_j])\right) \geq w([y_j]),$$

where an equality is possible only in the trivial cases $h_j = 0$ or $w(f^{[i]}([y_j])) = 0$, $i = 1, \dots k-1$, and $w(f^{[k]}([\tilde{y}_j])) = 0$.

If we use the mean-value evaluation (2.3.13) for computing the enclosures of the ranges $R\left(f^{[i]}; [y_j]\right)$, $i = 1, \dots, k-1$, instead of the direct evaluation $f^{[i]}([y_j])$, we can often obtain enclosures with smaller widths than in (3.2.2) [60]. By applying the mean-value theorem to $f^{[i]}$ at some $\hat{y}_j \in [y_j]$, we have

$$f^{[i]}(y_j) = f^{[i]}(\hat{y}_j) + J\left(f^{[i]}; y_j, \hat{y}_j\right)(y_j - \hat{y}_j), \tag{3.2.3}$$

where $J\left(f^{[i]}; y_j, \hat{y}_j\right)$ is the Jacobian of $f^{[i]}$ with its lth row evaluated at $y_j + \theta_{il}(\hat{y}_j - y_j)$ for some $\theta_{il} \in [0,1]$ $(l = 1, \ldots, n)$. Then from (3.2.1) and (3.2.3),

$$\begin{aligned} y_{j+1} = \hat{y}_j &+ \sum_{i=1}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + h_j^k f^{[k]}(y; t_j, t_{j+1}) \\ &+ \left\{ I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; y_j, \hat{y}_j\right) \right\} (y_j - \hat{y}_j). \end{aligned} \tag{3.2.4}$$

This formula is the basis of the interval Taylor series methods of Moore [48], [49], [50], Eijgenraam [19], Lohner [1], [44], [46], and Rihm [61] (see also [52]). Before we explain how (3.2.4) can be used, we consider in §3.2.1 a major difficulty in interval methods: the wrapping effect.

3.2.1 The Wrapping Effect

The wrapping effect is clearly illustrated by Moore's example [50],

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= -y_1. \end{aligned} \tag{3.2.5}$$

The solution of (3.2.5) with an initial condition y_0 is given by $y(t) = A(t)y_0$, where

$$A(t) = \begin{pmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{pmatrix}.$$

Let $y_0 \in [y_0]$. The interval vector $[y_0] \in \mathbb{IR}^2$ can be viewed as a rectangle in the (y_1, y_2) plane. At $t_1 > t_0$, $[y_0]$ is mapped by $A(t_1)$ into a rectangle of the same size, as shown in Figure 3.1. If we want to enclose this rectangle in an interval vector, we have to wrap it by another rectangle with sides parallel to the y_1 and y_2 axes. This larger rectangle is rotated on the next step, and so must be enclosed in a still larger rectangle. Thus, at each step, the enclosing rectangles become larger and larger, but the set $\{A(t)y_0 \mid y_0 \in [y_0],\ t > t_0\}$ remains a rectangle of the same size. Moore [50, p. 134] showed that at $t = 2\pi$, the interval inclusion is inflated by a factor of $e^{2\pi} \approx 535$, as the stepsize approaches zero.

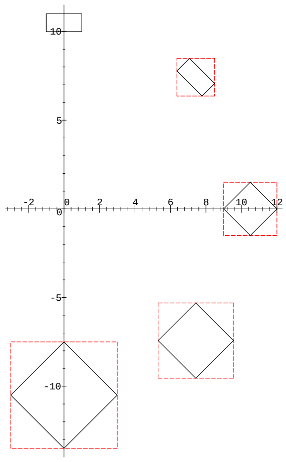


Figure 3.1: Wrapping of a rectangle specified by the interval vector $([-1,1],[10,11])^T$. The rotated rectangle is wrapped at $t = \frac{\pi}{4}n$, where $n = 1, \dots, 4$.

Jackson [32] gives a definition of wrapping.

Definition 3.1 *Let* $T \in \mathbb{R}^{n \times n}$, $[x] \in \mathbb{IR}^n$, *and* $c \in \mathbb{R}^n$. *Then the wrapping of the parallelepiped*

$$P = \{Tx + c \mid x \in [x]\}$$

is the tightest interval vector containing P.

It can be easily seen that the wrapping of the set $\{Tx + c \mid x \in [x]\}$ is given by $T[x] + c$, where $\{T[x]\}_i = \sum_{k=1}^{n} T_{ik}[x_k]$.

3.2.2 The Direct Method

A straightforward method for computing a tight enclosure $[y_{j+1}]$ at t_{j+1} is based on the evaluation of (3.2.4) in interval arithmetic. From (3.2.4), since

$$f^{[k]}(y; t_j, t_{j+1}) \in f^{[k]}([\tilde{y}_j]),$$

$$J\left(f^{[i]}; y_j, \hat{y}_j\right) \in J\left(f^{[i]}; [y_j]\right) \qquad (i = 1, 2, \dots, k-1), \text{ and}$$

$$y_j,\ \hat{y}_j \in [y_j],$$

we have

$$\begin{aligned} y(t_{j+1}; t_0, [y_0]) \subseteq [y_{j+1}] = \hat{y}_j + & \sum_{i=1}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + h_j^k f^{[k]}([\tilde{y}_j]) \\ & + \left\{ I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; [y_j]\right) \right\} ([y_j] - \hat{y}_j). \end{aligned} \tag{3.2.6}$$

Here, $[\tilde{y}_j]$ is an a priori enclosure of $y(t; t_j, [y_j])$ for all $t \in [t_j, t_{j+1}]$, $[y_j]$ is a tight enclosure of the solution at t_j, and $J\left(f^{[i]}; [y_j]\right)$ is the Jacobian of $f^{[i]}$ evaluated at $[y_j]$. We choose $\hat{y}_0$ to be the midpoint (we explain later why) of the initial interval $[y_0]$. Then, we choose

$$\hat{y}_{j+1} = m\left(\hat{y}_j + \sum_{i=1}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + h_j^k f^{[k]}([\tilde{y}_j])\right). \tag{3.2.7}$$

That is, $\hat{y}_{j+1}$ is the midpoint of the enclosure of the point solution at t_{j+1} starting from $\hat{y}_j$. For convenience, we introduce the notation $(j \geq 0)$

$$[v_{j+1}] = \hat{y}_j + \sum_{i=1}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + h_j^k f^{[k]}([\tilde{y}_j]) \quad \text{and} \tag{3.2.8}$$

$$[S_j] = I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; [y_j]\right). \tag{3.2.9}$$

Using (3.2.8–3.2.9), (3.2.6) can be written in the form

$$[y_{j+1}] = [v_{j+1}] + [S_j]\left([y_j] - \hat{y}_j\right). \tag{3.2.10}$$

By a *direct method* we mean one using (3.2.6), or (3.2.10), to compute a tight enclosure of the solution. This method is summarized in Algorithm 3.1. Note that from (3.2.7–3.2.8) and (3.2.10), $\hat{y}_{j+1} = m\left([v_{j+1}]\right) = m\left([y_{j+1}]\right)$. This equality holds because the interval vector $[S_j]([y_j] - \hat{y}_j)$ is symmetric.

Algorithm 3.1 Direct Method

INPUT:

$[\tilde{y}_j]$, h_j, $[y_j]$, $\hat{y}_j$.

COMPUTE:

$[v_{j+1}] := \hat{y}_j + \sum_{i=1}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + h_j^k f^{[k]}([\tilde{y}_j])$;

$[S_j] := I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; [y_j]\right)$;

$[y_{j+1}] := [v_{j+1}] + [S_j]\left([y_j] - \hat{y}_j\right)$;

$\hat{y}_{j+1} := m\left([v_{j+1}]\right) \quad (= m\left([y_{j+1}]\right))$.

OUTPUT:

$[y_{j+1}]$, $\hat{y}_{j+1}$.

Computing $[S_j]$

We show how the matrices $[S_j]$ can be computed [1]. Consider the variational equation

$$\Psi' = J\left(f; y\right)\Psi, \qquad \Psi(t_j) = I. \tag{3.2.11}$$

It can be shown that

$$\frac{\Psi^{(i)}(t)}{i!} = J\left(f^{[i]}; y\right)\Psi(t), \tag{3.2.12}$$

where $f^{[i]}$ is defined in (2.4.19–2.4.20), and $J\left(f^{[i]}; y\right)$ is the Jacobian of $f^{[i]}$. Then, from the Taylor series expansion of $\Psi(t)$ and (3.2.11–3.2.12), we have

$$\Psi(t_{j+1}) = I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; y\left(t_j\right)\right) + (\text{Remainder Term}). \tag{3.2.13}$$

Since in (3.2.13),

$$I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; y\left(t_j\right)\right) \in I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; [y_j]\right) = [S_j], \tag{3.2.14}$$

the interval matrices $[S_j]$ can be computed by computing the interval Taylor series (3.2.14) for the variational equation (3.2.11).

Alternatively, the Jacobians in (3.2.14) can be computed by differentiating the code list of the corresponding $f^{[i]}$, [5], [6].

Wrapping Effect in the Direct Method

If we use the direct method to compute the enclosures $[y_j]$, we might obtain unacceptably large interval vectors. This can be seen from the following considerations [60].

Using (3.2.10), we compute

$$\begin{aligned}
[y_1] &= [v_1] + [S_0]\,([y_0] - \hat{y}_0), \\
[y_2] &= [v_2] + [S_1]\,([y_1] - \hat{y}_1) \\
&= [v_2] + [S_1]\,(([v_1] - \hat{y}_1) + [S_0]\,([y_0] - \hat{y}_0)) \\
&= [v_2] + [S_1]\,([v_1] - \hat{y}_1) + [S_1]\,([S_0]\,([y_0] - \hat{y}_0)),
\end{aligned} \tag{3.2.15}$$

$$\vdots$$

$$\begin{aligned}
[y_{j+1}] &= [v_{j+1}] + [S_j]\,([y_j] - \hat{y}_j) \\
&= [v_{j+1}] + [S_j]\,([v_j] - \hat{y}_j) \\
&\qquad + [S_j]\,([S_{j-1}]\,([v_{j-1}] - \hat{y}_{j-1})) \\
&\qquad + \cdots \\
&\qquad + [S_j]\,([S_{j-1}] \cdots ([S_1]\,([S_0]\,([v_0] - \hat{y}_0))) \cdots),
\end{aligned} \tag{3.2.16}$$

where $[v_0] = [y_0]$. Note that the interval vectors $[v_l] - \hat{y}_l$ $(l = 0, \ldots, j)$ are symmetric, and denote them by $[\delta_l] = [v_l] - \hat{y}_l$. Let us consider one of the summands in (3.2.16), for example, the last one,

$$[S_j]\,([S_{j-1}] \cdots ([S_1]\,([S_0]\,[\delta_0])) \cdots). \tag{3.2.17}$$

To simplify our discussion, we assume that the matrices in (3.2.17) are point matrices and denote them by $S_j, S_{j-1}, \ldots, S_0$. We wish to compute the tightest interval vector that contains the set

$$\{S_j(S_{j-1} \cdots (S_1(S_0\delta_0)) \cdots) \mid \delta_0 \in [\delta_0]\}.$$

This set is the same as

$$\{(S_j S_{j-1} \cdots S_1 S_0)\delta_0 \mid \delta_0 \in [\delta_0]\},$$

which is wrapped by the interval vector

$$(S_j S_{j-1} \cdots S_1 S_0)\,[\delta_0] \tag{3.2.18}$$

(see §3.2.1). In practice, though, we compute

$$S_j(S_{j-1} \cdots (S_1(S_0\,[\delta_0])) \cdots), \tag{3.2.19}$$

and we can have wrapping at each step. That is, we first compute $S_0\,[\delta_0]$, resulting in one wrapping, then we compute $S_1(S_0\,[\delta_0])$, resulting in another wrapping, and so on. We can also see the result of the wrapping effect if we express the widths of the interval vectors in (3.2.18) and (3.2.19):

$$\begin{aligned} w\left((S_j S_{j-1} \cdots S_1 S_0)\,[\delta_0]\right) &= |S_j S_{j-1} \cdots S_1 S_0|\, w\left([\delta_0]\right) \\ &\leq |S_j|\,|S_{j-1}| \cdots |S_1|\,|S_0|\, w\left([\delta_0]\right) \\ &= w\left(S_j(S_{j-1} \cdots (S_1(S_0\,[\delta_0])) \cdots)\right). \end{aligned}$$

Frequently, $w\left((S_j S_{j-1} \cdots S_1 S_0)\,[\delta_0]\right) \ll w\left(S_j(S_{j-1} \cdots (S_1(S_0\,[\delta_0])) \cdots)\right)$ for j large, and the direct method often produces enclosures with increasing widths.

By choosing the vectors $\hat{y}_l = m\left([v_l]\right)$, we provide symmetric intervals $[v_l] - \hat{y}_l$, and by (2.2.10), we should have smaller overestimations in the evaluations of the enclosures than if we were to use nonsymmetric interval vectors.

Contracting Bounds

Here, we consider one of the best cases that can occur. If the diagonal elements of $J\left(f^{[1]}; [y_j]\right)$ are negative, then, in many cases, we can choose h_j such that

$$\left\|[S_j]\right\| = \left\| I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; [y_j]\right) \right\| < 1.$$

If $\hat{y}_j = m\left([y_j]\right)$, then

$$\left\|w\big([S_j]([y_j] - \hat{y}_j)\big)\right\| = \left\| |[S_j]|\, w\big([y_j] - \hat{y}_j\big)\right\| < \left\|w\big([y_j] - \hat{y}_j\big)\right\| = \left\|w\big([y_j]\big)\right\|.$$

That is, $[y_j] - \hat{y}_j$ propagates to a vector $[S_j]([y_j] - \hat{y}_j)$ at t_{j+1} with smaller norm of the width than $\|w([y_j])\|$.

3.2.3 Wrapping Effect in Generating Interval Taylor Coefficients

Consider the constant coefficient problem

$$y' = By, \quad y(0) \in [y_0]. \tag{3.2.20}$$

In practice, the relation (2.4.22) is used for generating interval Taylor coefficients. With this relation, we compute interval Taylor coefficients for the problem (3.2.20) as follows:

$$\begin{aligned} [y]_1 &= B[y_0], \\ [y]_2 &= \frac{1}{2}B[y_1] = \frac{1}{2}B\left(B[y_0]\right), \\ &\vdots \\ [y]_i &= \frac{1}{i}B[y]_{i-1} = \frac{1}{i!}B\big(B \cdots \left(B\left(B[y_0]\right)\right) \cdots \big). \end{aligned} \tag{3.2.21}$$

Therefore, the computation of the ith Taylor coefficient may involve i wrappings. In general, this implies that the computed enclosure of the kth Taylor coefficient, $f^{[k]}([\tilde{y}_j])$, on $[t_j, t_{j+1}]$ may be a significant overestimation of the range of $y^{(k)}(t)/k!$ on $[t_j, t_{j+1}]$. As a result, a variable stepsize control that controls the width of $h_j^k f^{[k]}([\tilde{y}_j]$ may impose a stepsize limitation much smaller than one would expect. In this example, it would be preferable to compute the coefficients directly by

$$[y]_i = \frac{1}{i!}B^i[y_0], \tag{3.2.22}$$

which involves at most one wrapping.

In the constant coefficient case, we can easily avoid the evaluation (3.2.21) by using (3.2.22), but generally, we do not know how to reduce the overestimation due to the wrapping effect in generating interval Taylor coefficients.

3.2.4 Local Excess in Taylor Series Methods

We consider the overestimation in one step of a Taylor series method based on (3.2.4).

The Taylor coefficient $f^{[k]}(y; t_j, t_{j+1})$ is enclosed by $f^{[k]}([\tilde{y}_j])$. If $[\tilde{y}_j]$ is a good enclosure of $y(t; t_j, [y_j])$ on $[t_j, t_{j+1}]$, then $\| w\left([\tilde{y}_j]\right) \| = O(h_j)$, assuming that $\|w([y_j])\| = O(h_j^r)$ for some $r \geq 1$. From (2.3.12), the overestimation in $f^{[k]}([\tilde{y}_j])$ of the range of $f^{[k]}$ over $[\tilde{y}_j]$ is $O(\| w\left([\tilde{y}_j]\right) \|) = O(h_j)$. Therefore, the overestimation in $h_j^k f^{[k]}([\tilde{y}_j])$ is $O(h_j^{k+1})$.

The matrices $J\left(f^{[i]}; y_j, \hat{y}_j\right)$ are enclosed by $J\left(f^{[i]}; [y_j]\right)$. That is, by evaluating the Jacobian of $f^{[i]}$ on the interval $[y_j]$. As a result, the overestimation from the second line in (3.2.4) is of order $O(h_j \|w([y_j])\|^2)$, [19, pp. 87–90]. This may be a major difficulty for problems with interval initial conditions, but should be insignificant for point initial conditions or interval initial conditions with small widths, provided that the widths of the computed enclosures remain sufficiently small throughout the computation.

Hence, if $f^{[k]}(y; t_j, t_{j+1})$ and $J\left(f^{[i]}; y_j, \hat{y}_j\right)$ are enclosed by $f^{[k]}([\tilde{y}_j])$ and $J\left(f^{[i]}; [y_j]\right)$, respectively, the overestimation in one step of Taylor series methods is given by

$$O(h_j \|w([y_j])\|^2) + O(h_j^{k+1}) + (\text{higher-order terms}). \tag{3.2.23}$$

We refer to this overestimation as local excess and define it more formally in §6.1. Advancing the solution in one step of Taylor series methods usually introduces such an excess (see Figure 3.2).

We should point out that by computing $h_j^k f^{[k]}([\tilde{y}_j])$, we bound the local truncation error in ITS methods for all solutions $y(t; t_j, y_j)$ with $y_j \in [y_j]$. Since this includes all solutions $y(t; t_0, y_0)$ with $y_0 \in [y_0]$, we are in effect bounding the global truncation error too. Thus, the distinction between the local and global truncation errors is somewhat blurred. In this thesis, we call $h_j^k f^{[k]}([\tilde{y}_j])$ the truncation error. A similar use of the truncation error holds for the IHO method discussed later.

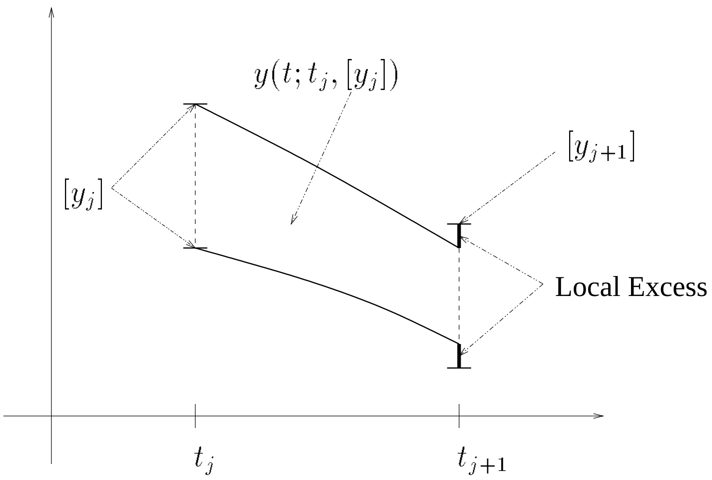


Figure 3.2: If $[y_j]$ is an enclosure of the solution at t_j, then the enclosure $[y_{j+1}]$ at t_{j+1} contains $y(t_{j+1};t_j,[y_j])$ and the local excess.

3.2.5 Lohner's Method

We derive Lohner's method from (3.2.4) in a different way than in [1], [44], and [46]. We show how $[y_1]$ and $[y_2]$ are computed and then give the algorithm for any $[y_j]$.

Let

$$z_{j+1} = h_j^k f^{[k]}(y;t_j,t_{j+1}) \in h_j^k f^{[k]}([\tilde{y}_j]) \equiv [z_{j+1}]\,, \tag{3.2.24}$$

$$s_{j+1} = m\left([z_{j+1}]\right), \tag{3.2.25}$$

$$\hat{y}_{j+1} = \hat{y}_j + \sum_{i=1}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + s_{j+1}, \quad \text{and} \tag{3.2.26}$$

$$S_j = I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]};y_j,\hat{y}_j\right) \in [S_j]\,, \tag{3.2.27}$$

where $[S_j]$ is defined in (3.2.9). Also let

$$A_0 = I,\ \hat{y}_0 = m\left([y_0]\right),\ \text{and}\ r_0 = y_0 - \hat{y}_0 \in [r_0] = [y_0] - \hat{y}_0, \tag{3.2.28}$$

where I is the identity matrix.

Using the notation (3.2.24–3.2.28), we obtain from (3.2.4)

$$
\begin{aligned}
y_1 &= \hat{y}_1 + S_0(y_0 - \hat{y}_0) + z_1 - s_1 \\
&\in \hat{y}_1 + ([S_0]A_0)\,[r_0] + [z_1] - s_1 \\
&\equiv [y_1], \quad \text{and} \\
y_1 &= \hat{y}_1 + S_0(y_0 - \hat{y}_0) + z_1 - s_1 \\
&= \hat{y}_1 + A_1\left(\left(A_1^{-1}S_0A_0\right) r_0 + A_1^{-1}(z_1 - s_1)\right) \\
&\in \left\{\hat{y}_1 + A_1 r_1 \mid r_1 \in [r_1]\right\},
\end{aligned}
$$

where $A_1 \in \mathbb{R}^{n\times n}$ is nonsingular and

$$
[r_1] = \left(A_1^{-1}([S_0]\,A_0)\right)[r_0] + A_1^{-1}([z_1] - s_1).
$$

We explain later how the matrices A_j $(j \geq 1)$ can be chosen.

Similarly,

$$
\begin{aligned}
y_2 &= \hat{y}_2 + S_1(y_1 - \hat{y}_1) + z_2 - s_2 \\
&= \hat{y}_2 + (S_1A_1)r_1 + z_2 - s_2 \\
&\in \hat{y}_2 + ([S_1]A_1)\,[r_1] + [z_2] - s_2 \\
&\equiv [y_2], \quad \text{and} \\
y_2 &= \hat{y}_2 + S_1(y_1 - \hat{y}_1) + z_2 - s_2 \\
&= \hat{y}_2 + A_2\left(\left(A_2^{-1}S_1A_1\right) r_1 + A_2^{-1}(z_2 - s_2)\right) \\
&\in \left\{\hat{y}_2 + A_2 r_2 \mid r_2 \in [r_2]\right\},
\end{aligned}
$$

where $A_2 \in \mathbb{R}^{n\times n}$ is nonsingular and

$$
[r_2] = \left(A_2^{-1}([S_1]A_1)\right)[r_1] + A_2^{-1}([z_2] - s_2).
$$

Continuing in this fashion, we obtain Lohner's method.

Algorithm 3.2 Lohner's Method

Input:

$[\tilde{y}_j]$, h_j;

$[y_j]$, $\hat{y}_j$, A_j, $[r_j]$.

Compute:

$[z_{j+1}] := h_j^k f^{[k]}([\tilde{y}_j])$;

$s_{j+1} := m\,([z_{j+1}])$;

$\hat{y}_{j+1} := \hat{y}_j + \sum_{i=1}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + s_{j+1}$;

$[S_j] := I + \sum_{i=1}^{k-1} h_j^i J\left(f^{[i]}; [y_j]\right)$;

Choose A_{j+1} as discussed below;

$[y_{j+1}] := \hat{y}_{j+1} + ([S_j]\,A_j)\,[r_j] + [z_{j+1}] - s_{j+1}$;

$[r_{j+1}] := \left(A_{j+1}^{-1}([S_j]\,A_j)\right)[r_j] + A_{j+1}^{-1}\,([z_{j+1}] - s_{j+1})$.

Output:

$[y_{j+1}]$, $\hat{y}_{j+1}$, A_{j+1}, $[r_{j+1}]$.

The Parallelepiped Method

If $A_{j+1} = m\,([S_j]\,A_j)$, then we have the parallelepiped method for reducing the wrapping effect. Let $\widehat{S}_j = m\,([S_j])$ and $[S_j] = \widehat{S}_j + [E_j]$. Then

$$\begin{aligned} A_{j+1} &= \widehat{S}_j A_j \quad \text{and} \\ A_{j+1}^{-1}([S_j]\,A_j) &= A_j^{-1}\widehat{S}_j^{-1}(\widehat{S}_j A_j + [E_j]A_j) \\ &= I + A_j^{-1}\widehat{S}_j^{-1}[E_j]A_j. \end{aligned}$$

Since

$$\left\|A_j^{-1}\widehat{S}_j^{-1}[E_j]A_j\right\| \leq \operatorname{cond}(A_j)\left\|\widehat{S}_j^{-1}[E_j]\right\|,$$

if $\left\|\widehat{S}_j^{-1}[E_j]\right\|$ is small and $\operatorname{cond}(A_j)$ is not too large, then $A_{j+1}^{-1}([S_j]\,A_j) \approx I$. As a result, there is no large overestimation in the evaluation of $\left(A_{j+1}^{-1}([S_j]\,A_j)\right)[r_j]$. However, the choice of A_{j+1} does not guarantee that it is well conditioned or even nonsingular. In fact, A_{j+1} may be ill conditioned, and a large overestimation may arise in this evaluation.

The QR-factorization Method

We describe Lohner's QR-factorization method, explain how it works, and illustrate it with a simple example.

Let $\widetilde{A}_{j+1} \in [S_j]\, A_j$, and let $\widehat{A}_{j+1} = \widetilde{A}_{j+1} P_{j+1}$, where P_{j+1} is a permutation matrix. We explain later in this subsection how P_{j+1} is chosen. We perform the QR-factorization $\widehat{A}_{j+1} = Q_{j+1} R_{j+1}$, where Q_{j+1} is an orthogonal matrix, and R_{j+1} is an upper triangular matrix. If A_{j+1} is chosen to be Q_{j+1} in Algorithm 3.2, we have the QR-factorization method for computing a tight enclosure of the solution.

We now give an intuitive explanation of how this method works. At each step, we want to compute an enclosure of the set

$$\left\{ \left(A_{j+1}^{-1} S_j A_j\right) r_j + A_{j+1}^{-1} \left(z_{j+1} - s_{j+1}\right) \;\middle|\; S_j \in [S_j]\,,\; r_j \in [r_j]\,,\; z_{j+1} \in [z_{j+1}] \right\} \tag{3.2.29}$$

that is as tight as possible. Consider first the set

$$\left\{ A_{j+1}^{-1} \left(z_{j+1} - s_{j+1}\right) \;\middle|\; z_{j+1} \in [z_{j+1}] \right\}. \tag{3.2.30}$$

If $\|A_{j+1}^{-1}\|$ is not much larger than 1, then

$$\left\| w\left(A_{j+1}^{-1}\left([z_{j+1}] - s_{j+1}\right)\right)\right\| = \left\| |A_{j+1}^{-1}| w\left([z_{j+1}]\right)\right\| \le \left\|A_{j+1}^{-1}\right\| \cdot \left\| w\left([z_{j+1}]\right)\right\|$$

will not be much larger than $\| w\left([z_{j+1}]\right) \|$. In this method, $A_{j+1}^{-1} = Q_{j+1}^{-1} = Q_{j+1}^{T}$ is orthogonal, so $\|A_{j+1}^{-1}\| \le \sqrt{n}$. In addition, $w\left([z_{j+1}]\right)$ can be made small by reducing the stepsize or changing the order of the Taylor series. Therefore, the set (3.2.30) can be enclosed in the interval vector

$$A_{j+1}^{-1}\left([z_{j+1}] - s_{j+1}\right),$$

whose width can be kept small.

Consider now the set

$$\left\{ \left(A_{j+1}^{-1} S_j A_j\right) r_j \;\middle|\; S_j \in [S_j]\,,\; r_j \in [r_j] \right\} \tag{3.2.31}$$

in (3.2.29). If $\widetilde{A}_{j+1} \in \{S_j A_j \mid S_j \in [S_j]\} \subseteq [S_j]\, A_j$ and $w\left([S_j]\right)$ is small, then

$$\{(S_j A_j) r_j \mid S_j \in [S_j]\,,\ r_j \in [r_j]\} \approx \{\widetilde{A}_{j+1} r_j \mid r_j \in [r_j]\}. \tag{3.2.32}$$

From (3.2.31) and (3.2.32), we have

$$\begin{aligned}
&\{(A_{j+1}^{-1} S_j A_j) r_j \mid S_j \in [S_j]\,,\ r_j \in [r_j]\} \\
&\qquad = \{(Q_{j+1}^{-1}(S_j A_j)) r_j \mid S_j \in [S_j]\,,\ r_j \in [r_j]\} \\
&\qquad \approx \{(Q_{j+1}^{-1} \widetilde{A}_{j+1}) r_j \mid r_j \in [r_j]\} \\
&\qquad \subseteq (Q_{j+1}^{-1} \widetilde{A}_{j+1})[r_j].
\end{aligned} \tag{3.2.33}$$

Note that $\widetilde{A}_{j+1}[r_j]$ is the wrapping of the set

$$\{\widetilde{A}_{j+1} r_j \mid r_j \in [r_j]\}, \tag{3.2.34}$$

while $(Q_{j+1}^{-1} \widetilde{A}_{j+1})[r_j]$ is the wrapping of the set

$$\{(Q_{j+1}^{-1} \widetilde{A}_{j+1}) r_j \mid r_j \in [r_j]\} = \{Q_{j+1}^{-1}(\widetilde{A}_{j+1} r_j) \mid r_j \in [r_j]\}, \tag{3.2.35}$$

which is the set $\{r_j \in [r_j]\}$ mapped by $\widetilde{A}_{j+1}$ and then the result mapped by Q_{j+1}^{-1}.

The vector corresponding to the first column of Q_{j+1} is parallel to the vector corresponding to the first column of $\widehat{A}_{j+1}$. The matrix Q_{j+1} induces an orthogonal coordinate system, where the axis corresponding to the first column of Q_{j+1} is parallel to those edges of the parallelepiped (3.2.34) that are parallel to the first column of $\widehat{A}_{j+1}$. Intuitively, we can expect an enclosure with less overestimation in the coordinate system induced by Q_{j+1} than in the original coordinate system. Furthermore, if the first column of Q_{j+1} is parallel to the longest edge of the parallelepiped in (3.2.34), we can expect a better result than if this column were parallel to a shorter edge. This is the reason for rearranging the columns of $\widetilde{A}_{j+1}$ by the permutation matrix P_{j+1}. Lohner suggests that P_{j+1} be chosen such that the first column of $\widehat{A}_{j+1}$ corresponds to the longest edge of (3.2.34), the second column to the second longest and so on.

If $\|\cdot\|_2$ is the Euclidean norm of a vector, $\widetilde{A}_{j+1,i}$ is the ith column of $\widetilde{A}_{j+1}$, and $[r_j]_i$ is the ith component of $[r_j]$, then the lengths of the edges of (3.2.34) are given by

$$l_i = \|\widetilde{A}_{j+1,i}\|_2 \cdot w\left([r_j]_i\right), \quad \text{for } i = 1, \ldots, n.$$

Let $l = (l_1, l_2, \ldots, l_n)^T$. The matrix P_{j+1} is such that the components of $l^T P_{j+1}$ are in non-increasing order (from 1 to n). As a result, the vector corresponding to the first column of $\widehat{A}_{j+1} = \widetilde{A}_{j+1}P_{j+1}$ is parallel to the longest edge of (3.2.34), and the first column of Q_{j+1} is parallel to that edge as well.

Example Let

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix} \quad \text{and} \quad [r] = \begin{pmatrix} [1,2] \\ [1,4] \end{pmatrix}.$$

The QR-factorization of A is

$$A = \left\{ -\frac{1}{\sqrt{5}} \begin{pmatrix} 1 & 2 \\ 2 & -1 \end{pmatrix} \right\} \left\{ -\frac{1}{\sqrt{5}} \begin{pmatrix} 5 & 3 \\ 0 & 1 \end{pmatrix} \right\} \equiv QR \tag{3.2.36}$$

Consider the set

$$\{Ar \mid r \in [r]\}. \tag{3.2.37}$$

The parallelepiped specified by $[r]$ (see Figure 3.3(a)) is mapped by A into the parallelepiped shown in Figure 3.3(b). The filled part in Figure 3.3(b) is the overestimation of (3.2.37) by $A[r]$. However, if the set in (3.2.37) is wrapped in the coordinate system induced by Q, we obtain a better enclosure (less overestimation) of this set (see Figure 3.3(c)).

Consider now the set

$$\{Q^{-1}Ar \mid r \in [r]\}. \tag{3.2.38}$$

The matrix Q^{-1} maps (3.2.37) into a parallelepiped with its shorter edge parallel to the original x axis. As a result, the wrapping of (3.2.38) is $(Q^{-1}A)[r]$ (see Figure 3.3(d)).

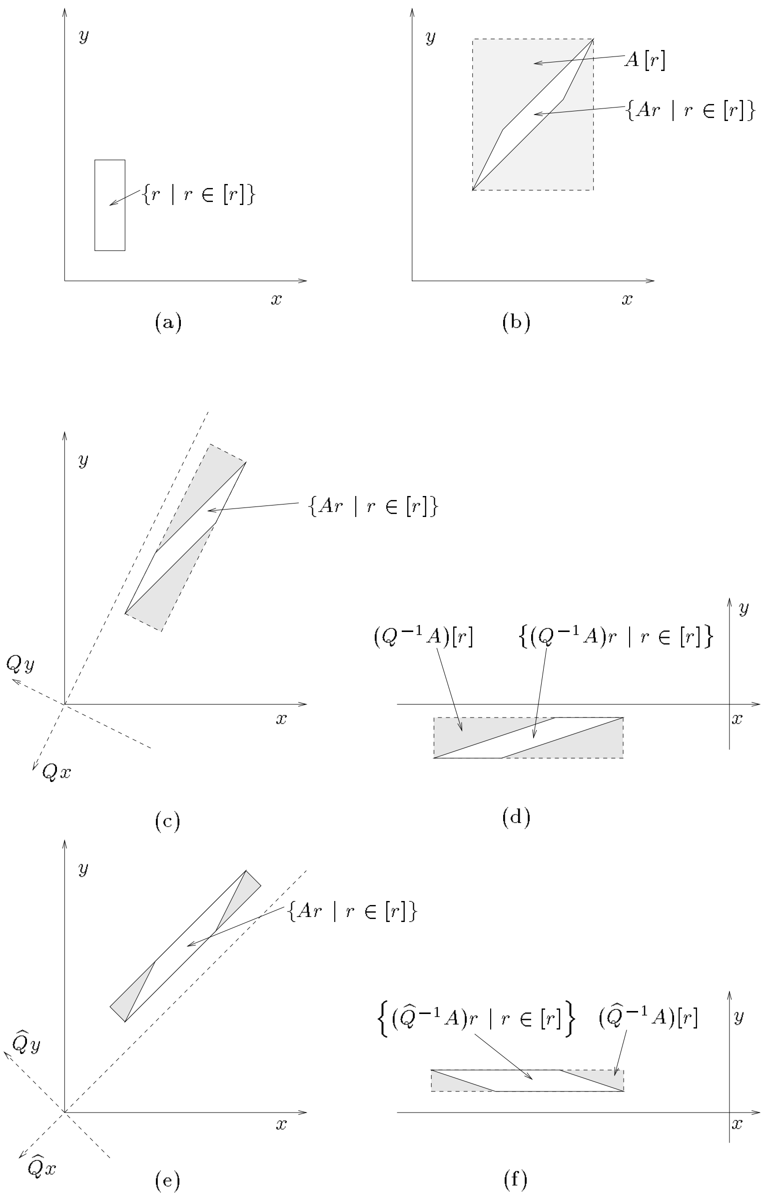


Figure 3.3: (a) The set $\{r \mid r \in [r]\}$.

(b) $\{Ar \mid r \in [r]\}$ enclosed by $A\,[r]$.

(c) $\{Ar \mid r \in [r]\}$ enclosed in the coordinate system induced by Q.

(d) $\{(Q^{-1}A)r \mid r \in [r]\}$ enclosed by $(Q^{-1}A)\,[r]$.

(e) $\{Ar \mid r \in [r]\}$ enclosed in the coordinate system induced by $\widehat{Q}$.

(f) $\{(\widehat{Q}^{-1}A)r \mid r \in [r]\}$ enclosed by $(\widehat{Q}^{-1}A)\,[r]$.

Now, interchange the columns of A, denote the new matrix by $\widehat{A}$, and compute the QR-factorization

$$\widehat{A} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} = \left\{ -\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right\} \left\{ -\frac{1}{\sqrt{2}} \begin{pmatrix} 2 & 3 \\ 0 & -1 \end{pmatrix} \right\} \equiv \widehat{Q}\widehat{R}. \tag{3.2.39}$$

If we wrap the set (3.2.37) in the coordinate system induced by $\widehat{Q}$ (see Figure 3.3(e)), we obtain a better enclosure than in the coordinate system induced by Q. In Figure 3.3(f), the parallelepiped $\{Ar \mid r \in [r]\}$ is rotated by $\widehat{Q}^{-1}$. The longest edge of the rotated parallelepiped is parallel to the x axis, and the overestimation in $(\widehat{Q}^{-1}A)[r]$ is smaller than in $(Q^{-1}A)[r]$ and $A[r]$.

To summarize, let $A \in \mathbb{R}^{n\times n}$, $[r] \in \mathbb{IR}^n$, and $A = QR$, where Q is an orthogonal matrix and R is an upper triangular matrix. Normally, if we wrap the parallelepiped $\{Ar \mid r \in [r]\}$ in the coordinate system induced by Q, we obtain a better enclosure than in the original coordinate system. Moreover, if we rearrange the columns of A, as described in this subsection, before computing Q, we usually obtain a better enclosure than without rearranging those columns.

Chapter 4

An Interval Hermite-Obreschkoff Method

In this chapter, we derive an interval Hermite-Obreschkoff (IHO) method and compare it with the "standard" interval Taylor series methods.

Hermite-Obreschkoff methods are usually considered for computing an approximate solution of a stiff problem [22], [24], [77], [78]. Here, we are not interested in obtaining a method that is targeted specifically to solving stiff problems—our purpose is to obtain a general-purpose method that produces better enclosures at a smaller cost than the explicit validated methods based on Taylor series.

Hermite-Obreschkoff methods have smaller truncation errors and better stability than Taylor series methods with the same stepsize and order. Also, for the same order, the IHO method needs fewer Taylor coefficients for the solution to the IVP and its variational equation than an ITS method. However, the former requires that we enclose the solution of a generally nonlinear system, while the latter does not. The extra cost of enclosing such a solution includes one matrix inversion and a few matrix-matrix multiplications.

The method that we propose consists of two phases, which can be considered as a predictor and a corrector. The predictor computes an enclosure $[y_{j+1}^{(0)}]$ of the solution at

t_{j+1}. Using $[y_{j+1}^{(0)}]$, the corrector computes a tighter enclosure $[y_{j+1}] \subseteq [y_{j+1}^{(0)}]$ at t_{j+1}.

In the next section, we derive the interval Hermite-Obreschkoff method; in §4.2, we give an algorithmic description of it; and in §4.3, we explain why the IHO method may perform better than ITS methods.

4.1 Derivation of the Interval Hermite-Obreschkoff Method

First, in §4.1.1, we show how the point Hermite-Obreschkoff method can be obtained. Then in §4.1.2, we outline our new IHO method. Finally, in §4.1.3, we derive it: we describe how to improve the predicted enclosure and how to represent the improved enclosure in a manner that reduces the wrapping effect in propagating the solution.

4.1.1 The Point Method

Let

$$P_{p,q}(s) = \frac{s^q(s-1)^p}{(p+q)!}, \tag{4.1.1}$$

$$c_i^{q,p} = \frac{q!}{(p+q)!}\frac{(q+p-i)!}{(q-i)!}, \quad \text{and} \tag{4.1.2}$$

$$g_i(s) = \frac{g^{(i)}(s)}{i!}, \tag{4.1.3}$$

where $p \geq 0$, $q \geq 0$, $0 \leq i \leq q$, and $g(t)$ is any $(p+q+1)$ times differentiable function. If we integrate $\int_0^1 P_{p,q}(s)g^{(p+q+1)}(s)\,ds$ repeatedly by parts, we find[1]

$$(-1)^{(p+q)}\int_0^1 P_{p,q}(s)g^{(p+q+1)}(s)\,ds = \sum_{i=0}^{q}(-1)^i c_i^{q,p} g_i(1) - \sum_{i=0}^{p} c_i^{p,q} g_i(0). \tag{4.1.4}$$

If $y(t)$ is the solution to the IVP

$$y' = f(y), \quad y(t_j) = y_j, \tag{4.1.5}$$

[1]This derivation is sometimes attributed to Darboux [16] and Hermite [28].

and we set $g(s) = y(t_j + sh_j)$, then

$$g^{(p+q+1)}(s) = h_j^{p+q+1} y^{(p+q+1)}(t_j + sh_j), \tag{4.1.6}$$

$$g_i(0) = \frac{g^{(i)}(0)}{i!} = h_j^i \frac{y^{(i)}(t_j)}{i!} = h_j^i f^{[i]}(y_j), \quad \text{and} \tag{4.1.7}$$

$$g_i(1) = \frac{g^{(i)}(1)}{i!} = h_j^i \frac{y^{(i)}(t_j + h_j)}{i!} = h_j^i f^{[i]}(y_{j+1}), \tag{4.1.8}$$

where $y_{j+1} = y(t_j + h_j)$, and the functions $f^{[i]}$ are defined in (2.4.19–2.4.20). Also,

$$\begin{aligned}(-1)^{p+q} &\int_0^1 P_{p,q}(s) g^{(p+q+1)}(s)\, ds \\ &= (-1)^{p+q} h_j^{p+q+1} \int_0^1 P_{p,q}(s) y^{(p+q+1)}(t_j + sh_j)\, ds \\ &= (-1)^q \frac{q!p!}{(p+q)!} h_j^{p+q+1} \frac{y^{(p+q+1)}(t; t_j, t_{j+1})}{(p+q+1)!},\end{aligned} \tag{4.1.9}$$

where the lth component of $y^{(p+q+1)}(t; t_j, t_{j+1})$ is evaluated at some $\xi_{jl} \in [t_j, t_{j+1}]$.

From (4.1.4) and (4.1.7–4.1.9),

$$\begin{aligned}\sum_{i=0}^{q} (-1)^i c_i^{q,p} h_j^i f^{[i]}(y_{j+1}) = \sum_{i=0}^{p} c_i^{p,q} h_j^i f^{[i]}(y_j) \\ +(-1)^q \frac{q!p!}{(p+q)!} h_j^{p+q+1} \frac{y^{(p+q+1)}(t; t_j, t_{j+1})}{(p+q+1)!}.\end{aligned} \tag{4.1.10}$$

For a given y_j, if we solve the nonlinear (in general) system of equations

$$\sum_{i=0}^{q} (-1)^i c_i^{q,p} h_j^i f^{[i]}(y_{j+1}) = \sum_{i=0}^{p} c_i^{p,q} h_j^i f^{[i]}(y_j) \tag{4.1.11}$$

for y_{j+1}, we obtain an approximation of local order $O(h_j^{p+q+1})$ to the solution of (4.1.5). The system (4.1.11) defines the point (q,p) Hermite-Obreschkoff method [22], [24], [27, p. 277], [77], [78].

Remarks

1. If $p > 0$ and $q = 0$, we obtain an explicit Taylor series formula:

$$y_{j+1} = \sum_{i=0}^{p} h_j^i f^{[i]}(y_j) + \frac{h_j^{p+1}}{(p+1)!} y^{(p+1)}(t; t_j, t_{j+1}).$$

2. If $p = 0$ and $q > 0$, then (4.1.10) becomes an implicit Taylor series formula:

$$y_j = \sum_{i=0}^{q} (-1)^i h_j^i f^{[i]}(y_{j+1}) + (-1)^{q+1} \frac{h_j^{q+1}}{(q+1)!} y^{(q+1)}(t; t_j, t_{j+1}).$$

Therefore, we can consider the Hermite-Obreschkoff methods that we obtain from (4.1.10) as a generalization of Taylor series methods.

4.1.2 An Outline of the Interval Method

Suppose that we have computed an enclosure of the solution at t_j. The idea behind our IHO method is to compute bounds on the solution at t_{j+1}, for all y_j in the solution set at t_j, by enclosing the solution of the generally nonlinear system (4.1.10). We enclose this solution in two phases, which we denote as a predictor and a corrector.

PREDICTOR: Compute an enclosure of the solution at t_{j+1} using an interval Taylor series method of order $(q+1)$.

CORRECTOR: Improve this enclosure by enclosing the solution of (4.1.10).

In the corrector, we perform a Newton-like step to tighten the bounds computed by the predictor. From (4.1.10), we have to bound the $(p+q+1)$st Taylor coefficient on $[t_j, t_{j+1}]$. We can enclose this coefficient by generating it with the a priori enclosure computed in Algorithm I. This computation is the same as enclosing the remainder term in ITS methods (see §3.2).

4.1.3 The Interval Method

Suppose that we have computed $[y_j]$, $\hat{y}_j$, A_j, and $[r_j]$ at t_j such that

$$y(t_j; t_0, [y_0]) \subseteq [y_j] \quad \text{and} \tag{4.1.12}$$

$$y(t_j; t_0, [y_0]) \subseteq \{\hat{y}_j + A_j r_j \mid r_j \in [r_j]\}, \tag{4.1.13}$$

where $\hat{y}_j = m([y_j])$, $A_j \in \mathbb{R}^{n\times n}$ is nonsingular, and $[r_j] \in \mathbb{IR}^n$. The interval vectors $[y_j]$ and $\hat{y}_j + A_j[r_j]$ are not necessarily the same. We use the representation $\{\hat{y}_j + A_j r_j \mid r_j \in [r_j]\}$ to reduce the wrapping effect in propagating the solution and the representation $[y_j]$ to compute the coefficients for the solution to the variational equation (see §3.2.2). Suppose also that we have verified existence and uniqueness of the solution on $[t_j, t_{j+1}]$ and have computed an a priori enclosure $[\tilde{y}_j]$ on $[t_j, t_{j+1}]$ and an enclosure $[y_{j+1}^{(0)}] \subseteq [\tilde{y}_j]$ at t_{j+1}. We show in §4.2.2 how to compute $[y_{j+1}^{(0)}]$ in the predictor. Here, we describe how to construct a corrector based on (4.1.10).

Our goal is to compute (at t_{j+1}) a tighter enclosure $[y_{j+1}]$ of the solution than $[y_{j+1}^{(0)}]$ and a representation of the enclosure set in the form

$$\{\hat{y}_{j+1} + A_{j+1} r_{j+1} \mid r_{j+1} \in [r_{j+1}]\}.$$

That is, we have to compute $[y_{j+1}]$, $\hat{y}_{j+1}$, A_{j+1}, and $[r_{j+1}]$ for the next step.

Let

$$y_j = y(t_j; t_0, y_0) \quad \text{and} \quad y_{j+1} = y(t_{j+1}; t_0, y_0)$$

for some $y_0 \in [y_0]$, and $\hat{y}_{j+1}^{(0)} = m\big([y_{j+1}^{(0)}]\big)$. Since

$$y_{j+1},\ \hat{y}_{j+1}^{(0)} \in [y_{j+1}^{(0)}] \quad \text{and} \quad y_j,\ \hat{y}_j \in [y_j],$$

we can apply the mean-value theorem to the two sums in (4.1.10) to obtain

$$\begin{aligned}
\sum_{i=0}^{q} (-1)^i c_i^{q,p} h_j^i f^{[i]}(\hat{y}_{j+1}^{(0)}) + \left(\sum_{i=0}^{q} (-1)^i c_i^{q,p} h_j^i J\left(f^{[i]}; y_{j+1}, \hat{y}_{j+1}^{(0)}\right) \right) (y_{j+1} - \hat{y}_{j+1}^{(0)}) \\
= \sum_{i=0}^{p} c_i^{p,q} h_j^i f^{[i]}(\hat{y}_j) + \left(\sum_{i=0}^{p} c_i^{p,q} h_j^i J\left(f^{[i]}; y_j, \hat{y}_j\right) \right) (y_j - \hat{y}_j) \\
+ (-1)^q \frac{q!p!}{(p+q)!} h_j^{p+q+1} \frac{y^{(p+q+1)}(t; t_j, t_{j+1})}{(p+q+1)!},
\end{aligned} \tag{4.1.14}$$

where $J\left(f^{[i]}; y_{j+1}, \hat{y}_{j+1}^{(0)}\right)$ is the Jacobian of $f^{[i]}$ with its lth row evaluated at $y_{j+1} + \theta_{il}(\hat{y}_{j+1}^{(0)} - y_{j+1})$ for some $\theta_{il} \in [0,1]$, and $J\left(f^{[i]}; y_j, \hat{y}_j\right)$ is the Jacobian of $f^{[i]}$ with its lth row evaluated at $y_j + \eta_{il}(\hat{y}_j - y_j)$ for some $\eta_{il} \in [0,1]$, $l = 1, \ldots, n$.

Using (4.1.14), we show how to compute a tighter enclosure than $[y_{j+1}^{(0)}]$ at t_{j+1}.

Let

$$\begin{aligned} S_{j+1,-} &= \sum_{i=0}^{q} (-1)^i c_i^{q,p} h_j^i J\left(f^{[i]}; y_{j+1}, \hat{y}_{j+1}^{(0)}\right) \\ &\in \sum_{i=0}^{q} (-1)^i c_i^{q,p} h_j^i J\left(f^{[i]}; [y_{j+1}^{(0)}]\right) \equiv [S_{j+1,-}], \end{aligned} \tag{4.1.15}$$

$$\widehat{S}_{j+1,-} = m\left([S_{j+1,-}]\right), \tag{4.1.16}$$

$$\begin{aligned} S_{j,+} &= \sum_{i=0}^{p} c_i^{p,q} h_j^i J\left(f^{[i]}; y_j, \hat{y}_j\right) \\ &\in \sum_{i=0}^{p} c_i^{p,q} h_j^i J\left(f^{[i]}; [y_j]\right) \equiv [S_{j,+}], \end{aligned} \tag{4.1.17}$$

$$[B_j] = \left(\widehat{S}_{j+1,-}^{-1}[S_{j,+}]\right) A_j, \tag{4.1.18}$$

$$[C_j] = I - \widehat{S}_{j+1,-}^{-1}[S_{j+1,-}], \tag{4.1.19}$$

$$[v_j] = [y_{j+1}^{(0)}] - \hat{y}_{j+1}^{(0)}, \tag{4.1.20}$$

$$\begin{aligned} \epsilon_{j+1} &= (-1)^q \frac{q!p!}{(p+q)!} h_j^{p+q+1} \frac{y^{(p+q+1)}(t; t_j, t_{j+1})}{(p+q+1)!} \\ &\in (-1)^q \frac{q!p!}{(p+q)!} h_j^{p+q+1} f^{[p+q+1]}([\tilde{y}_j]) \equiv [\epsilon_{j+1}], \end{aligned} \tag{4.1.21}$$

$$g_{j+1} = \sum_{i=0}^{p} c_i^{p,q} h_j^i f^{[i]}(\hat{y}_j) - \sum_{i=0}^{q} (-1)^i c_i^{q,p} h_j^i f^{[i]}(\hat{y}_{j+1}^{(0)}), \quad \text{and} \tag{4.1.22}$$

$$\delta_{j+1} = g_{j+1} + \epsilon_{j+1} \in g_{j+1} + [\epsilon_{j+1}] \equiv [\delta_{j+1}]. \tag{4.1.23}$$

With the notation (4.1.15–4.1.23), we write (4.1.14) as

$$\begin{aligned} \widehat{S}_{j+1,-}(y_{j+1} - \hat{y}_{j+1}^{(0)}) &= S_{j,+}(y_j - \hat{y}_j) + \delta_{j+1} \\ &\qquad - (S_{j+1,-} - \widehat{S}_{j+1,-})(y_{j+1} - \hat{y}_{j+1}^{(0)}). \end{aligned} \tag{4.1.24}$$

Since

$$y_j - \hat{y}_j \in \left\{ A_j r_j \mid r_j \in [r_j] \right\},$$

there exists $r_j \in [r_j]$ such that $y_j - \hat{y}_j = A_j r_j$. Therefore, we can transform (4.1.24) into

$$\widehat{S}_{j+1,-}(y_{j+1} - \hat{y}_{j+1}^{(0)}) = (S_{j,+}A_j) r_j + \delta_{j+1} - (S_{j+1,-} - \widehat{S}_{j+1,-})(y_{j+1} - \hat{y}_{j+1}^{(0)}). \tag{4.1.25}$$

For small h_j, we can compute the inverse of $\widehat{S}_{j+1,-}$. Then from (4.1.25) and using (4.1.15–4.1.23),

$$\begin{aligned} y_{j+1} - \hat{y}_{j+1}^{(0)} &= \Big((\widehat{S}_{j+1,-}^{-1} S_{j,+}) A_j \Big) r_j + \widehat{S}_{j+1,-}^{-1} \delta_{j+1} + (I - \widehat{S}_{j+1,-}^{-1} S_{j+1,-})(y_{j+1} - \hat{y}_{j+1}^{(0)}) \\ &\in \Big((\widehat{S}_{j+1,-}^{-1} [S_{j,+}]) A_j \Big) [r_j] + \widehat{S}_{j+1,-}^{-1} [\delta_{j+1}] \\ &\qquad + (I - \widehat{S}_{j+1,-}^{-1} [S_{j+1,-}])([y_{j+1}^{(0)}] - \hat{y}_{j+1}^{(0)}) \\ &= [B_j][r_j] + [C_j][v_j] + \widehat{S}_{j+1,-}^{-1} [\delta_{j+1}]. \end{aligned} \tag{4.1.26}$$

Since

$$y_{j+1} = y(t_{j+1}; t_0, y_0) \in \hat{y}_{j+1}^{(0)} + [B_j][r_j] + [C_j][v_j] + \widehat{S}_{j+1,-}^{-1} [\delta_{j+1}]$$

for an arbitrary $y_0 \in [y_0]$, then

$$y(t_{j+1}; t_0, [y_0]) \in \hat{y}_{j+1}^{(0)} + [B_j][r_j] + [C_j][v_j] + \widehat{S}_{j+1,-}^{-1} [\delta_{j+1}].$$

We compute an interval vector that is a tight enclosure of the solution at t_{j+1} by

$$[y_{j+1}] = \Big(\hat{y}_{j+1}^{(0)} + [B_j][r_j] + [C_j][v_j] + \widehat{S}_{j+1,-}^{-1} [\delta_{j+1}] \Big) \cap [y_{j+1}^{(0)}], \tag{4.1.27}$$

where "$\cap$" denotes intersection of interval vectors. For the next step, we propagate

$$\hat{y}_{j+1} = m\left([y_{j+1}]\right), \tag{4.1.28}$$

A_{j+1}, which is the Q-factor from the QR-factorization of $m\left([B_j]\right)$, and

$$\begin{aligned} [r_{j+1}] &= (A_{j+1}^{-1} [B_j])[r_j] + (A_{j+1}^{-1} [C_j])[v_j] \\ &\qquad + (A_{j+1}^{-1} \widehat{S}_{j+1,-}^{-1})[\delta_{j+1}] + A_{j+1}^{-1} (\hat{y}_{j+1}^{(0)} - \hat{y}_{j+1}). \end{aligned} \tag{4.1.29}$$

Remarks

1. Since we enclose the Taylor coefficient

$$\frac{y^{(p+q+1)}(t; t_j, t_{j+1})}{(p+q+1)!} \quad \text{by} \quad f^{[p+q+1]}([\tilde{y}_j]),$$

the overestimation in the term $h_j^{p+q+1} f^{[p+q+1]}([\tilde{y}_j])$ is of $O(h_j^{p+q+2})$, provided that $\|w([\tilde{y}_j])\| = O(h_j)$; see §3.2.4. Therefore, the order of the IHO method is $(p+q+1)$. Note that in the point case, the order of an Hermite-Obreschkoff method is $(p+q)$.

In §8.2, we verify empirically that the order of an IHO method with p and q is indeed $(p+q+1)$.

2. We have explicitly used the inverse of $\widehat{S}_{j+1,-}$ in our method. This is due in part to the software available to us. It may be useful to consider other ways to perform this computation at a later date.

3. We could use the inverse of the interval matrix $[S_{j+1,-}]$ instead of $\widehat{S}_{j+1,-}^{-1}$. However, it is easier to compute the enclosure of the inverse of a point matrix than of an interval matrix. In fact, computing a tight enclosure of the inverse of an interval matrix is NP hard in general [63].

4. In (4.1.27), we intersect $\hat{y}_{j+1}^{(0)} + [B_j][r_j] + [C_j][v_j] + \widehat{S}_{j+1,-}^{-1}[\delta_{j+1}]$ and $[y_{j+1}^{(0)}]$. As a result, the computed enclosure, $[y_{j+1}]$, is always contained in

$$[y_{j+1}^{(0)}] \quad \text{and} \quad \hat{y}_{j+1}^{(0)} + [B_j][r_j] + [C_j][v_j] + \widehat{S}_{j+1,-}^{-1}[\delta_{j+1}].$$

Therefore, we can never compute a wider enclosure than $[y_{j+1}^{(0)}]$.

5. Once we obtain $[y_{j+1}]$, we can set $[y_{j+1}^{(0)}] = [y_{j+1}]$ and compute another enclosure, hopefully tighter than $[y_{j+1}]$, by repeating the same procedure. Thus, we can improve this enclosure iteratively. The experiments that we have performed show, however, that this iteration does not improve the results significantly, but increases the cost.

6. If we intersect the computed enclosure as in (4.1.27), it is important to choose $\hat{y}_{j+1} \in [y_{j+1}]$. If we set $\hat{y}_{j+1} = \hat{y}_{j+1}^{(0)}$, it might happen that $\hat{y}_{j+1} = \hat{y}_{j+1}^{(0)} \notin [y_{j+1}]$,

because $\hat{y}_{j+1}^{(0)}$ is the midpoint of $[y_{j+1}^{(0)}]$, which is generally a wider enclosure than $[y_{j+1}]$.

7. The interval vectors $[r_j]$ $(j > 0)$ are not symmetric in general, but they are symmetric in Lohner's method (see §3.2.5).

4.2 Algorithmic Description of the Interval Hermite-Obreschkoff Method

In this section, we show how to compute the coefficients $c_i^{p,q}$ and $c_i^{q,p}$. Then, we describe the predictor and corrector phases of the IHO method in a form suitable for implementation.

4.2.1 Computing the Coefficients $c_i^{p,q}$ and $c_i^{q,p}$

From (4.1.2)

$$\begin{aligned} c_i^{q,p} &= \frac{q!}{(p+q)!}\frac{(q+p-i)!}{(q-i)!} = \frac{q!}{(p+q)!}\frac{(q+p-i+1)!}{(q-i+1)!}\frac{q-i+1}{q+p-i+1} \\ &= c_{i-1}^{q,p}\frac{q-i+1}{q+p-i+1}. \end{aligned} \tag{4.2.1}$$

Since $c_0^{q,p} = 1$, we can compute the coefficients $c_i^{q,p}$ for $i = 1, \dots, q$ by (4.2.1). In a similar way, we compute $c_i^{p,q}$ for $i = 1, \dots, p$.

4.2.2 Predicting an Enclosure

We compute an enclosure $[y_{j+1}^{(0)}]$ for the solution at t_{j+1} by Algorithm 4.2, which is part of Lohner's method (see §3.2.5).

Algorithm 4.1 Compute the coefficients $c_i^{p,q}$ and $c_i^{q,p}$.

Input:

p, q.

Compute:

$c_0^{q,p} := 1;$

for $i := 1$ **to** q

$c_i^{q,p} := c_{i-1}^{q,p}(q-i+1)/(q+p-i+1);$

end

$c_0^{p,q} := 1;$

for $i := 1$ **to** p

$c_i^{p,q} := c_{i-1}^{p,q}(p-i+1)/(q+p-i+1);$

end

Output:

$c_i^{p,q}$, for $i = 0, \ldots, p$;

$c_i^{q,p}$, for $i = 0, \ldots, q$.

Algorithm 4.2 Predictor: compute an enclosure with order $q+1$.

Input:

q, h_j, $[z_{j+1}] := h_j^{q+1} f^{[q+1]}([\tilde{y}_j])$;

$\hat{y}_j$, A_j, $[r_j]$, $[y_j]$.

Compute:

$f_{j,i}^{+} := h_j^i f^{[i]}(\hat{y}_j)$, for $i = 1, \ldots, q$;

$[u_{j+1}] := \hat{y}_j + \sum_{i=1}^{q} f_{j,i}^{+} + [z_{j+1}]$;

$[F_{j,i}^{+}] := h_j^i J\left(f^{[i]}; [y_j]\right)$, for $i = 1, \ldots, q$;

$[S_j] := I + \sum_{i=1}^{q} [F_{j,i}^{+}]$;

$[y_{j+1}^{(0)}] := [u_{j+1}] + [z_{j+1}] + ([S_j]A_j)[r_j]$.

Output:

$[y_{j+1}^{(0)}]$;

$f_{j,i}^{+}$, $[F_{j,i}^{+}]$, for $i = 1, \ldots, q$.

4.2.3 Improving the Predicted Enclosure

Suppose that we have computed an enclosure $[y_{j+1}^{(0)}]$ of $y(t_{j+1}; t_0, [y_0])$ with Algorithm 4.2. In Algorithm 4.3, we describe an algorithm based on the derivations in §4.1.3 for improving $[y_{j+1}^{(0)}]$.

Remarks

1. We could use the a priori enclosure $[\tilde{y}_j]$ from Algorithm I instead of computing $[y_{j+1}^{(0)}]$. We briefly explain the reasons for computing $[y_{j+1}^{(0)}]$.

   (a) The a priori enclosure $[\tilde{y}_j]$ may be too wide and the corrector phase may not produce a tight enough enclosure in one iteration. As a result, the corrector, which is the expensive part, may need more than one iteration to obtain a tight enough enclosure (see §8.3.2, p. 110).

   (b) Predicting a reasonably tight enclosure $[y_{j+1}^{(0)}]$ is not expensive: we need to generate the terms $[f_{j,i}]$ and $[F_{j,i}]$, for $i = 1, \dots, q$. We need them in the corrector, but for $i = 1, \dots, p$. Usually, a good choice for q is $q \in \{p, p+1, p+2\}$ (see §4.3.1). Therefore, we do not create much extra work when generating these terms in Algorithm 4.2.

2. Algorithm 4.3 describes a general method. If, for example, the problem being solved does not exhibit exponential growth of the widths of the enclosures due to the wrapping effect, we do not have to compute a QR-factorization and represent the enclosure as in (4.1.13).

3. The matrix A_{j+1} is a floating-point approximation to an orthogonal matrix. Since A_{j+1}^{-1} is not necessarily equal to the transpose of A_{j+1}, A_{j+1}^{-1} must be enclosed in interval arithmetic.

Algorithm 4.3 Corrector: improve the enclosure and prepare for the next step.

INPUT:

p, q, $\quad c_i^{p,q}$ for $i = 0, \ldots, p$, $\quad c_i^{q,p}$ for $i = 0, \ldots, q$;

h_j, $\hat{y}_j$, A_j, $[r_j]$, $[y_{j+1}^{(0)}]$;

$f_{j,i}$, $[F_{j,i}]$, $\quad$ for $i = 1, \ldots, q$;

$[z_{j+1}] := h_j^{p+q+1} f^{[p+q+1]}([\tilde{y}_j])$.

COMPUTE:

$\hat{y}_{j+1}^{(0)} := m\big([y_{j+1}^{(0)}]\big)$;

$f_{j+1,i}^- := h_j^i f^{[i]}(\hat{y}_{j+1}^{(0)})$, $\quad$ for $i = 1, \ldots, q$;

$g_{j+1} := \hat{y}_j - \hat{y}_{j+1}^{(0)} + \sum_{i=1}^p c_i^{p,q} f_{j,i} - \sum_{i=1}^q (-1)^i c_i^{q,p} f_{j+1,i}^-$;

$\gamma := (-1)^q q!p!/(p+q)!$;

$[\delta_{j+1}] := g_{j+1} + \gamma[z_{j+1}]$;

$[S_{j,+}] := I + \sum_{i=1}^p c_i^{p,q}[F_{j,i}]$;

$[F_{j+1,i}^-] := h_j^i J\left(f^{[i]}; [y_{j+1}^{(0)}]\right)$, $\quad$ for $i = 1, \ldots, q$;

$[S_{j+1,-}] := I + \sum_{i=1}^q (-1)^i c_i^{q,p}[F_{j+1,i}^-]$;

$\widehat{S}_{j+1,-} := m\left([S_{j+1,-}]\right)$;

$[B_j] := (\widehat{S}_{j+1,-}^{-1}[S_{j,+}])A_j$;

$[C_j] := I - \widehat{S}_{j+1,-}^{-1}[S_{j+1,-}]$;

$[v_j] := [y_{j+1}^{(0)}] - \hat{y}_{j+1}^{(0)}$;

$[y_{j+1}] := \big(\hat{y}_{j+1}^{(0)} + [B_j][r_j] + [C_j][v_j] + \widehat{S}_{j+1,-}^{-1}[\delta_{j+1}]\big) \cap [y_{j+1}^{(0)}]$;

$\widehat{A}_{j+1} := m\left([B_j]\right)$;

$A_{j+1} := Q$ factor of the QR-factorization of $\widehat{A}_{j+1}$;

$\hat{y}_{j+1} := m\left([y_{j+1}]\right)$;

$[r_{j+1}] := (A_{j+1}^{-1}[B_j])[r_j] + (A_{j+1}^{-1}[C_j])[v_j] + (A_{j+1}^{-1}\widehat{S}_{j+1,-}^{-1})[\delta_{j+1}] + A_{j+1}^{-1}(\hat{y}_{j+1}^{(0)} - \hat{y}_{j+1})$.

OUTPUT:

$\hat{y}_{j+1}$, A_{j+1}, $[r_{j+1}]$, $[y_{j+1}]$.

4. It is convenient to compute the terms $h_j^i f^{[i]}([\tilde{y}_j])$ for $i = 1, 2, \ldots, (p+q+1)$ in Algorithm I (see Chapter 7). Then, we do not have to recompute $h_j^{q+1} f^{[q+1]}([\tilde{y}_j])$ in the predictor and $h_j^{p+q+1} f^{[p+q+1]}([\tilde{y}_j])$ in the corrector.

4.3 Comparison with Interval Taylor Series Methods

We explain why the IHO method may perform better than the ITS methods. First, in §4.3.1 and §4.3.2, we show that on constant coefficient problems, the IHO method is more stable and produces smaller enclosures than an ITS method with the same stepsize and order. Then, in §4.3.3, we study one step of these methods in the general case and show again that the IHO should produce smaller enclosures than the ITS methods. Finally, in §4.3.4, we consider the amount of work in one step of each of these methods.

In this section, we assume that both methods have the same order of the truncation error. That is, if the order of the Taylor series is k, we consider an IHO method with p and q such that $p + q + 1 = k$.

4.3.1 The One-Dimensional Constant Coefficient Case. Instability Results

Consider the problem

$$y' = \lambda y, \qquad y(0) \in [y_0], \tag{4.3.1}$$

where $\lambda \in \mathbb{R}$ and $\lambda < 0$.[2]

DEFINITION 4.1 *We say that an interval method for enclosing the solution of (4.3.1) with a constant stepsize is asymptotically unstable, if*

$$w([y_j]) \to \infty, \qquad \text{as } j \to \infty.$$

[2]Since we have not defined complex interval arithmetic, we do not consider problems with λ complex.

In this and in the next subsection, we consider methods with constant stepsize h for simplicity of analysis.

The Interval Taylor series method

Suppose that at $t_j > 0$, we have computed a tight enclosure $[y_j^{ITS}]$ of the solution with an ITS method, and $[\tilde{y}_j^{ITS}]$ is an a priori enclosure of the solution on $[t_j, t_{j+1}]$, for all $y_j \in [y_j^{ITS}]$, where $[y_0^{ITS}] = [y_0]$. Denote

$$[z_{j+1}^{ITS}] = \frac{(\lambda h)^k}{k!}[\tilde{y}_j^{ITS}] \quad (j \geq 0) \tag{4.3.2}$$

and let

$$T_r(z) = \sum_{i=0}^{r} \frac{z^i}{i!}. \tag{4.3.3}$$

Using (4.3.2–4.3.3), an interval Taylor series method for computing tight enclosures of the solution to (4.3.1) can be written as

$$[y_{j+1}^{ITS}] = T_{k-1}(\lambda h)[y_j^{ITS}] + [z_{j+1}^{ITS}]; \tag{4.3.4}$$

cf. (3.2.4). Since $w([\tilde{y}_j^{ITS}]) \geq w([y_j^{ITS}])$, we derive from (4.3.2–4.3.4)

$$\begin{aligned} w([y_{j+1}^{ITS}]) &= |T_{k-1}(\lambda h)| w([y_j^{ITS}]) + \frac{|\lambda h|^k}{k!} w([\tilde{y}_j^{ITS}]) \\ &\geq |T_{k-1}(\lambda h)| w([y_j^{ITS}]) + \frac{|\lambda h|^k}{k!} w([y_j^{ITS}]) \\ &= \left(|T_{k-1}(\lambda h)| + \frac{|\lambda h|^k}{k!} \right) w([y_j^{ITS}]). \end{aligned}$$

Therefore, the ITS method given by (4.3.4) is asymptotically unstable for stepsizes h such that

$$|T_{k-1}(\lambda h)| + \frac{|\lambda h|^k}{k!} > 1. \tag{4.3.5}$$

This result implies that we have restrictions on the stepsize not only from the function $T_{k-1}(\lambda h)$, as in point methods for IVPs for ODEs, but also from the factor $|\lambda h|^k/k!$ in the remainder term. Note also that the stepsize restriction arising from (4.3.5) is more severe than the one that would arise from the standard Taylor series methods of order k or $k+1$.

The Interval Hermite-Obreschkoff method

Let $y_j \in [y_j^{IHO}]$, where we assume that $[y_j^{IHO}]$ is computed with an IHO method and $[y_0^{IHO}] = [y_0]$. From (4.1.10), the true solution y_{j+1} corresponding to the point y_j satisfies

$$\left(\sum_{i=0}^{q}(-1)^i c_i^{q,p}\frac{(\lambda h)^i}{i!}\right) y_{j+1} = \left(\sum_{i=0}^{p} c_i^{p,q}\frac{(\lambda h)^i}{i!}\right) y_j + (-1)^q \frac{q!p!}{(p+q)!}\frac{(\lambda h)^{p+q+1}}{(p+q+1)!}y(\xi), \tag{4.3.6}$$

where $\xi \in [t_j, t_{j+1}]$. Let

$$R_{p,q}(z) = \frac{\sum_{i=0}^{p} c_i^{p,q}\frac{z^i}{i!}}{\sum_{i=0}^{q} c_i^{q,p}\frac{(-z)^i}{i!}} \quad \text{and} \tag{4.3.7}$$

$$Q_{p,q}(z) = \sum_{i=0}^{q} c_i^{q,p}\frac{(-z)^i}{i!}, \tag{4.3.8}$$

where $c_i^{q,p}$ ($c_i^{p,q}$) are defined in (4.1.2). Also let

$$[z_{j+1}^{IHO}] = \frac{(\lambda h)^k}{k!}[\tilde{y}_j^{IHO}] \tag{4.3.9}$$

($k = p+q+1$), where $[\tilde{y}_j^{IHO}]$ is an a priori enclosure of the solution on $[t_j, t_{j+1}]$ for any $y_j \in [y_j^{IHO}]$.

Let $\gamma_{p,q} = q!p!/(p+q)!$. From (4.3.6–4.3.9), we compute a tight enclosure $[y_{j+1}^{IHO}]$ by

$$[y_{j+1}^{IHO}] = R_{p,q}(\lambda h)[y_j^{IHO}] + (-1)^q \frac{\gamma_{p,q}}{Q_{p,q}(\lambda h)}[z_{j+1}^{IHO}]. \tag{4.3.10}$$

From (4.3.9–4.3.10),

$$\begin{aligned} w([y_{j+1}^{IHO}]) &= |R_{p,q}(\lambda h)| w([y_j^{IHO}]) + \frac{\gamma_{p,q}}{|Q_{p,q}(\lambda h)|}\frac{|\lambda h|^k}{k!} w([\tilde{y}_j^{IHO}]) \\ &\geq |R_{p,q}(\lambda h)| w([y_j^{IHO}]) + \frac{\gamma_{p,q}}{|Q_{p,q}(\lambda h)|}\frac{|\lambda h|^k}{k!} w([y_j^{IHO}]) \\ &= \left(|R_{p,q}(\lambda h)| + \frac{\gamma_{p,q}}{|Q_{p,q}(\lambda h)|}\frac{|\lambda h|^k}{k!}\right) w([y_j^{IHO}]). \end{aligned}$$

Therefore, the IHO method is asymptotically unstable for h such that

$$|R_{p,q}(\lambda h)| + \frac{\gamma_{p,q}}{|Q_{p,q}(\lambda h)|} \frac{|\lambda h|^k}{k!} > 1. \tag{4.3.11}$$

In (4.3.5) and (4.3.11),

$$T_{k-1}(z) = e^z + O(z^k) \quad \text{and} \quad R_{p,q}(z) = e^z + O(z^{p+q+1}) = e^z + O(z^k)$$

are approximations to e^z of the same order. However, $R_{p,q}(z)$ is a rational Padé approximation to e^z (see for example [59]). If z is complex with $\mathrm{Re}(z) < 0$, the following results are known:

- if $p = q$, then $|R_{p,q}(z)| < 1$, and $|R_{p,q}(z)| \to 1$ as $|z| \to \infty$ [10];
- if $q = p+1$ or $q = p+2$, then $|R_{p,q}(z)| < 1$, and $R_{p,q}(z) \to 0$ as $|z| \to \infty$ [18]; and
- if $q > p$ and $z \in \mathbb{R}$, $z < 0$, then $|R_{p,q}(z)| < 1$, and $R_{p,q}(z) \to 0$ as $|z| \to \infty$ [73]

(see also [42, pp. 236–237]). Consider (4.3.5) and (4.3.11). For the ITS method, $|T_{k-1}(\lambda h)| < 1$ when λh is in the stability region of $T_{k-1}(z)$. However, for the IHO method with $\lambda \in \mathbb{R}$, $\lambda < 0$, $|R_{p,q}(\lambda h)| < 1$ for any $h > 0$ when $q \geq p$, and $R_{p,q}(\lambda h) \to 0$ as $\lambda h \to -\infty$ when $q > p$. Roughly speaking, the stepsize in the ITS method is restricted by both

$$|T_{k-1}(\lambda h)| \quad \text{and} \quad \frac{|\lambda h|^k}{k!},$$

while in the IHO method, the stepsize is limited mainly by

$$\frac{\gamma_{p,q}}{|Q_{p,q}(\lambda h)|} \frac{|\lambda h|^k}{k!}. \tag{4.3.12}$$

Since $\gamma_{p,q}/\left|Q_{p,q}(\lambda h)\right|$ is usually much smaller than one, $|\lambda h|^k/k! < 1$ implies a more severe restriction on the stepsize than (4.3.12). Thus, the stepsize limit for the IHO method is usually much larger than for the ITS method.

An important point to note here is that an interval version of a standard numerical method that is suitable for stiff problems may still have a restriction on the stepsize. To obtain an interval method without a stepsize restriction, we must find a stable formula not only for advancing the step, but also for the associated truncation error.

Consider again (4.3.4) and (4.3.10). From (4.3.4), we can derive

$$[y_{j+1}^{ITS}] = \big(T_{k-1}(\lambda h)\big)^{j+1}[y_0] + \sum_{i=1}^{j+1}\big(T_{k-1}(\lambda h)\big)^{j+1-i}[z_i^{ITS}].$$

The width of $[y_{j+1}^{ITS}]$ is

$$w([y_{j+1}^{ITS}]) = \big|\big(T_{k-1}(\lambda h)\big)^{j+1}\big|w([y_0]) + \sum_{i=1}^{j+1}\big|\big(T_{k-1}(\lambda h)\big)^{j+1-i}\big|w([z_i^{ITS}]). \tag{4.3.13}$$

We derive from (4.3.10),

$$[y_{j+1}^{IHO}] = \big(R_{p,q}(\lambda h)\big)^{j+1}[y_0] + (-1)^q\frac{\gamma_{p,q}}{Q_{p,q}(\lambda h)}\sum_{i=1}^{j+1}\big(R_{p,q}(\lambda h)\big)^{j+1-i}[z_i^{IHO}].$$

The width of $[y_{j+1}^{IHO}]$ is

$$\begin{aligned} w([y_{j+1}^{IHO}]) &= \big|\big(R_{p,q}(\lambda h)\big)^{j+1}\big|w([y_0]) \\ &\qquad + \frac{\gamma_{p,q}}{|Q_{p,q}(\lambda h)|}\sum_{i=1}^{j+1}\big|\big(R_{p,q}(\lambda h)\big)^{j+1-i}\big|w([z_i^{IHO}]). \end{aligned} \tag{4.3.14}$$

If h is such that

$$T_{k-1}(\lambda h) \approx R_{p,q}(\lambda h) \quad \text{and} \quad \big|T_{k-1}(\lambda h)\big| < 1,$$

and if we assume that

$$w([y_0]) = 0 \quad \text{and} \quad [z_i^{IHO}] \approx [z_i^{ITS}], \quad \text{for } i = 1, 2, \ldots j+1,$$

then from (4.3.13) and (4.3.14),

$$w([y_{j+1}^{IHO}]) \approx \frac{\gamma_{p,q}}{|Q_{p,q}(\lambda h)|}w([y_{j+1}^{ITS}]). \tag{4.3.15}$$

That is, for $\lambda < 0$ and small h, the widths of the intervals in the IHO method are approximately $\gamma_{p,q}/|Q_{p,q}(\lambda h)| < 1$ times the corresponding widths of the intervals produced by the ITS method. As the stepsize increases, $\big|T_{k-1}(\lambda h)\big| + |\lambda h|^k/k!$ becomes greater than one. Then, the ITS method is asymptotically unstable and produces intervals with increasing widths. For the same stepsizes, the IHO method may produce intervals with decreasing widths when $q \geq p$.

In Table 4.1, we give approximate values for $\gamma_{p,q} = q!p!/(p+q)!$, for $p = 3, 4, , \ldots, 13$ and $q \in \{p, p+1, p+2\}$. As can be seen from this table, the error constant $q!p!/(p+q)!$ becomes very small as p and q increase.

In §8.3.1, we show numerical results comparing the ITS and IHO methods on (4.3.1) for $\lambda = -10$.

p	$\gamma_{p,p}$	$\gamma_{p,p+1}$	$\gamma_{p,p+2}$
3	5.0×10^{-2}	2.9×10^{-2}	1.8×10^{-2}
4	1.4×10^{-2}	7.9×10^{-3}	4.8×10^{-3}
5	4.0×10^{-3}	2.2×10^{-3}	1.3×10^{-3}
6	1.1×10^{-3}	5.8×10^{-4}	3.3×10^{-4}
7	2.9×10^{-4}	1.6×10^{-4}	8.7×10^{-5}
8	7.8×10^{-5}	4.1×10^{-5}	2.3×10^{-5}
9	2.1×10^{-5}	1.1×10^{-5}	6.0×10^{-6}
10	5.4×10^{-6}	2.8×10^{-6}	1.5×10^{-6}
11	1.4×10^{-6}	7.4×10^{-7}	4.0×10^{-7}
12	3.7×10^{-7}	1.9×10^{-7}	1.0×10^{-7}
13	9.6×10^{-8}	5.0×10^{-8}	2.7×10^{-8}

Table 4.1: Approximate values for $\gamma_{p,q}$, $p = 3, 4, \ldots 13$, $q \in \{p, p+1, p+2\}$.

4.3.2 The n-Dimensional Constant Coefficient Case

Consider the IVP

$$y' = By, \quad y_0 \in [y_0], \tag{4.3.16}$$

where $B \in \mathbb{R}^{n\times n}$ and $n > 1$.

We compare one step of an ITS method, which uses Lohner's technique for reducing the wrapping effect, and one step of the IHO method, which uses a similar technique for reducing the wrapping effect. Then, we compare the enclosures after several steps of these methods. We assume that in addition to an enclosure $[y_j]$ of the solution at t_j, we also have a representation of the enclosure in the form

$$\{\hat{y}_j + A_j r_j \mid r_j \in [r_j]\}, \tag{4.3.17}$$

where $\hat{y}_j \in [y_j]$, $A_j \in \mathbb{R}^{n\times n}$ is nonsingular, and $[r_j] \in \mathbb{IR}^n$. We also assume that we have an a priori enclosure $[\tilde{y}_j]$ of the solution on $[t_j, t_{j+1}]$, where $h = t_{j+1} - t_j$.

Enclosures after One Step

The Interval Taylor Series Method Using (4.3.3), we can write an ITS method, with Lohner's coordinate transformation, as

$$[y_{j+1}^{ITS}] = T_{k-1}(hB)\hat{y}_j + \big(T_{k-1}(hB)A_j\big)[r_j] + [z_{j+1}], \tag{4.3.18}$$

where

$$[z_{j+1}] = \frac{h^k}{k!} B^k \, [\tilde{y}_j]\,. \tag{4.3.19}$$

The width of $[y_{j+1}^{ITS}]$ is

$$w([y_{j+1}^{ITS}]) = \big|T_{k-1}(hB)A_j\big| w([r_j]) + w([z_{j+1}]). \tag{4.3.20}$$

The Interval Hermite-Obreschkoff method Using (4.3.7–4.3.8) and (4.3.19), the IHO method can be expressed by

$$[y_{j+1}^{IHO}] = R_{p,q}(hB)\hat{y}_j + \big(R_{p,q}(hB)A_j\big)[r_j] + (-1)^q \gamma_{p,q}\big(Q_{p,q}(hB)\big)^{-1}[z_{j+1}].$$

(Note that for h small, we can compute the inverse of the matrix $Q_{p,q}(hB)$.) The width of $[y_{j+1}^{IHO}]$ is given by

$$w([y_{j+1}^{IHO}]) = \big|R_{p,q}(hB)A_j\big|w([r_j]) + \gamma_{p,q}\big|\big(Q_{p,q}(hB)\big)^{-1}\big|w([z_{j+1}]). \tag{4.3.21}$$

Comparing (4.3.21) and (4.3.20), we see that in the IHO method we multiply the width of the error term, $w([z_{j+1}])$, from the ITS method by $\gamma_{p,q}\big|\big(Q_{p,q}(hB)\big)^{-1}\big|$. If, for example, $p = q = 8$, then

$$\gamma_{8,8} \approx 7.8 \times 10^{-5}$$

(see Table 4.1). Consider $\big(Q_{p,q}(hB)\big)^{-1}$ and suppose that $q > 0$. For small h,

$$\big(Q_{p,q}(hB)\big)^{-1} \approx (I - c_1^{q,p}hB)^{-1} \approx I + c_1^{q,p}hB.$$

This implies that for small h, multiplying by the matrix $\big|\big(Q_{p,q}(hB)\big)^{-1}\big|$ does not significantly increase $w([z_{j+1}])$. Furthermore, it often happens that $\|\big(Q_{p,q}(hB)\big)^{-1}\| < 1$. Hence, multiplying by this matrix may reduce $w([z_{j+1}])$ still further.

In Lohner's method, we propagate $\big(T_{k-1}(hB)A_j\big)[r_j]$, where $T_{k-1}(hB)$ is an approximation of the matrix exponential of order k:

$$T_{k-1}(hB) = e^{hB} + O(h^k).$$

In the IHO method, we propagate $\big(R_{p,q}(hB)A_j\big)[r_j]$, where $R_{p,q}(hB)$ is a rational approximation to the matrix exponential of order k:

$$R_{p,q}(hB) = e^{hB} + O(h^{p+q+1}) = e^{hB} + O(h^k).$$

If hB is small, then

$$\big|T_{k-1}(hB)A_j\big|w([r_j]) \approx \big|R_{p,q}(hB)A_j\big|w([r_j])$$

in (4.3.20) and (4.3.21).

Enclosures after Several Steps

Now, we study how the enclosures propagate after several steps in the ITS and the IHO methods. For simplicity, we assume that the matrix B in (4.3.16) is diagonalizable and can be represented in the form

$$B = X^{-1} D X,$$

where $D = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ are the eigenvalues of B.

DEFINITION 4.2 *We say that an interval method for enclosing the solution of (4.3.16) with a constant stepsize is asymptotically unstable, if*

$$\|w([y_j])\| \to \infty, \qquad \textit{as } j \to \infty.$$

The Interval Taylor series method We compute $[y_1^{ITS}]$ by

$$[y_1^{ITS}] = T_{k-1}(hB)[y_0] + [z_1^{ITS}],$$

where

$$[z_{i+1}^{ITS}] = \frac{h^k}{k!}[\tilde{y}_i^{ITS}] \quad (i \geq 0),$$

and $[\tilde{y}_i^{ITS}]$ is an a priori enclosure of the solution on $[t_i, t_{i+1}]$ for all $y_i \in [y_i^{ITS}]$. Then, instead of representing the enclosure at t_1 in the form of a parallelepiped as in (4.3.17) and computing an enclosure at t_2 by the formula (4.3.18), we assume that we compute

$$[y_2^{ITS}] = \big(T_{k-1}(hB)\big)^2[y_0] + T_{k-1}(hB)[z_1^{ITS}] + [z_2^{ITS}],$$

where there may be wrappings in the evaluation of $\big(T_{k-1}(hB)\big)^2[y_0]$ and $T_{k-1}(hB)[z_1]$. Following this procedure, we assume that we compute $[y_{j+1}^{ITS}]$ by

$$\begin{aligned}
[y_{j+1}^{ITS}] &= \big(T_{k-1}(hB)\big)^{j+1}[y_0] + \sum_{i=1}^{j+1}\big(T_{k-1}(hB)\big)^{j+1-i}[z_i^{ITS}] \\
&= \Big(X^{-1}\big(T_{k-1}(hD)\big)^{j+1}X\Big)\,[y_0] \qquad (4.3.22)\\
&\qquad + \sum_{i=1}^{j+1}\Big(X^{-1}\big(T_{k-1}(hD)\big)^{j+1-i}X\Big)\,[z_i^{ITS}].
\end{aligned}$$

We make this assumption to obtain a simple formula for $[y_{j+1}^{ITS}]$ in terms of $[y_0]$ and $[z_i^{ITS}]$, $i = 1, \ldots, (j+1)$. Otherwise, if we had used (4.3.18), we would have products involving the transformation matrices A_j and a more complicated formula to analyze. The formula (4.3.22) gives, in general, tighter enclosures than (4.3.18) (see §3.2.2).

The width of $[y_{j+1}^{ITS}]$ is given by

$$\begin{aligned} w([y_{j+1}^{ITS}]) =& \left|X^{-1}\big(T_{k-1}(hD)\big)^{j+1}X\right|w([y_0]) \\ &+ \sum_{i=1}^{j+1}\left|X^{-1}\big(T_{k-1}(hD)\big)^{j+1-i}X\right|w([z_i^{ITS}]). \end{aligned} \tag{4.3.23}$$

The Interval Hermite-Obreschkoff method Similar to the considerations in §4.3.1 and in the previous paragraph, we can derive for the IHO method

$$\begin{aligned} [y_{j+1}^{IHO}] &= \big(R_{p,q}(hB)\big)^{j+1}[y_0] + (-1)^q\gamma_{p,q}\sum_{i=1}^{j+1}\big(R_{p,q}(hB)\big)^{j+1-i}\left(\big(Q_{p,q}(hB)\big)^{-1}[z_i^{IHO}]\right) \\ &= \left(X^{-1}\big(R_{p,q}(hD)\big)^{j+1}X\right)[y_0] \\ &\quad + (-1)^q\gamma_{p,q}\sum_{i=1}^{j+1}\left(X^{-1}\big(R_{p,q}(hD)\big)^{j+1-i}X\right)\left(\big(Q_{p,q}(hB)\big)^{-1}[z_i^{IHO}]\right) \end{aligned}$$

where

$$[z_{i+1}^{IHO}] = \frac{h^k}{k!}B^k[\tilde{y}_i^{IHO}],$$

and $[\tilde{y}_i^{IHO}]$ is an a priori enclosure on $[t_i, t_{i+1}]$ for $y_i \in [y_i^{IHO}]$. The width of $[y_{j+1}^{IHO}]$ is

$$\begin{aligned} w([y_{j+1}^{IHO}]) =& \left|X^{-1}\big(R_{p,q}(hD)\big)^{j+1}X\right|w([y_0]) \\ &+ \gamma_{p,q}\sum_{i=1}^{j+1}\left|X^{-1}\big(R_{p,q}(hD)\big)^{j+1-i}X\right|\left|\big(Q_{p,q}(hB)\big)^{-1}\right|w([z_i^{IHO}]). \end{aligned} \tag{4.3.24}$$

Consider (4.3.23) and (4.3.24) and suppose that $\mathrm{Re}(\lambda_i) < 0$ for $i = 1, \ldots, n$. The matrices $T_{k-1}(hD)$ and $R_{p,q}(hD)$ are diagonal with diagonal elements $T_{k-1}(h\lambda_i)$ and $R_{p,q}(h\lambda_i)$, respectively, where λ_i is an eigenvalue of B. As h increases, $\|T_{k-1}(hD)\|$ will eventually become greater than one, and then the ITS method is asymptotically unstable. However, for any $h > 0$, $\|R_{p,q}(hD)\| < 1$ for $q = p$, $p+1$, or $p+2$, and $\|R_{p,q}(hD)\| \to 0$

as $h \to \infty$ for $q = p+1$ or $q = p+2$ (see §4.3.1). Therefore, if we ignore the wrapping effect, the IHO method does not have stability restrictions from the associated stability function $R_{p,q}(z)$ when $q = \{p, p+1, p+2\}$. However, it still has a restriction from the formula for the truncation error.

We can show for the ITS method that

$$w([y_{j+1}^{ITS}]) \geq \frac{h^k}{k!}\left|B^k\right| w([y_j^{ITS}]),$$

and for the IHO method that

$$w([y_{j+1}^{IHO}]) \geq \frac{h^k}{k!}\gamma_{p,q}\left|\big(Q_{p,q}(hB)\big)^{-1}\right|\left|B^k\right| w([y_j^{IHO}]).$$

These two inequalities suggest that the restriction on the stepsize in the IHO method occurs at values significantly larger than in ITS methods.

As in the previous subsection, if $w([y_0]) = 0$, then for small stepsizes and small hB, we should expect

$$\|w([y_{j+1}^{IHO}])\| \lessapprox \gamma_{p,q}\|\big(Q_{p,q}(hB)\big)^{-1}\| \, \|w([y_{j+1}^{ITS}])\|. \tag{4.3.25}$$

Moreover, for larger stepsizes and eigenvalues satisfying $\mathrm{Re}(\lambda_i) < 0$, $i = 1, ..., n$, the IHO with $q = p$, $p+1$, or $p+2$ is more stable than the ITS method.

In §8.3.1, we show numerical results comparing the two methods on a two-dimensional constant coefficient problem.

4.3.3 The General Case

Comparing an ITS method with the IHO method in the nonlinear case is not as simple as in the constant coefficient case. We can easily compare the corresponding remainder terms on each step, but we cannot make precise conclusions, as in the constant coefficient case, about the propagation of the set $\{A_j r_j \mid r_j \in [r_j]\}$. However, we show by numerical experiments in §8.3.2 the advantages of the IHO method over ITS methods on some nonlinear problems.

The Interval Taylor Series Method

In Lohner's method,

$$[y_{j+1}^{ITS}] = \hat{y}_j + \sum_{i=1}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + ([S_j]A_j)[r_j] + h_j^k f^{[k]}([\tilde{y}_j])$$

(see §3.2.5), and

$$w([y_{j+1}^{ITS}]) = |[S_j]A_j| w([r_j]) + h_j^k w(f^{[k]}([\tilde{y}_j])). \tag{4.3.26}$$

The Interval Hermite-Obreschkoff Method

From (4.1.26), we compute a tight enclosure by the formula

$$\begin{aligned}[y_{j+1}^{IHO}] = \hat{y}_{j+1}^{(0)} + \big((\widehat{S}_{j+1,-}^{-1}[S_{j,+}])A_j\big)[r_j] + \widehat{S}_{j+1,-}^{-1}[\delta_{j+1}] \\ + (I - \widehat{S}_{j+1,-}^{-1}[S_{j+1,-}])([y_{j+1}^{(0)}] - \hat{y}_{j+1}^{(0)}).\end{aligned}$$

For simplicity in the discussion, we do not intersect $[y_{j+1}^{IHO}]$ with $[y_{j+1}^{(0)}]$ as in (4.1.27); such an intersection produces an enclosure $[y_{j+1}] \subseteq [y_{j+1}^{IHO}]$. Therefore, our conclusions are valid for $[y_{j+1}]$. The width of $[y_{j+1}^{IHO}]$ is[3]

$$\begin{aligned}w\left([y_{j+1}^{IHO}]\right) = \left|\big(\widehat{S}_{j+1,-}^{-1}[S_{j,+}]\big)A_j\right| w([r_j]) + |\widehat{S}_{j+1,-}^{-1}| w([\delta_{j+1}]) \\ + |I - \widehat{S}_{j+1,-}^{-1}[S_{j+1,-}]| w([y_{j+1}^{(0)}]).\end{aligned} \tag{4.3.27}$$

Let again $k = p + q + 1$ and consider the terms in (4.3.27).

The term $|\widehat{S}_{j+1,-}^{-1}| w([\delta_{j+1}])$. Since $w([\delta_{j+1}]) = w([\epsilon_{j+1}])$, (see (4.1.23)),

$$|\widehat{S}_{j+1,-}^{-1}| w([\delta_{j+1}]) = |\widehat{S}_{j+1,-}^{-1}| w([\epsilon_{j+1}]) = \big(\gamma_{p,q} |\widehat{S}_{j+1,-}^{-1}|\big) h_j^k w(f^{[k]}([\tilde{y}_j])). \tag{4.3.28}$$

Comparing the terms involving $h_j^k w(f^{[k]}([\tilde{y}_j]))$ in (4.3.26) and (4.3.28), we see that in (4.3.28) the reduction is roughly $\gamma_{p,q}$, assuming that the components of $|\widehat{S}_{j+1,-}^{-1}|$ are not large (which is the case if h_j is sufficiently small). This situation is similar to the n-dimensional constant coefficient case.

[3] If $[r_j]$ is symmetric, then for an interval matrix $[A]$, $w([A][r_j]) = |[A]| w([r_j])$ (cf. (2.2.10)). The interval vector $[r_j]$ ($j > 0$) is symmetric in Lohner's method, but it is generally nonsymmetric in the IHO method. Assuming $[r_j]$ symmetric, we obtain a simple formula as in (4.3.27).

The term $|I - \widehat{S}^{-1}_{j+1,-}[S_{j+1,-}]|w([y^{(0)}_{j+1}])$**.** Let

$$[S_{j+1,-}] = \widehat{S}_{j+1,-} + [-E_{j+1}, E_{j+1}],$$

where E_{j+1} is a point matrix, and $\widehat{S}_{j+1,-} = m\left([S_{j+1,-}]\right)$; cf. (4.1.15). Then

$$\begin{aligned}
|I - \widehat{S}^{-1}_{j+1,-}[S_{j+1,-}]| &= |I - \widehat{S}^{-1}_{j+1,-}(\widehat{S}_{j+1,-} + [-E_{j+1}, E_{j+1}])| \\
&= |\widehat{S}^{-1}_{j+1,-}[-E_{j+1}, E_{j+1}]| \\
&\le |\widehat{S}^{-1}_{j+1,-}| \times |[-E_{j+1}, E_{j+1}]| \\
&= \frac{1}{2}|\widehat{S}^{-1}_{j+1,-}|w([-E_{j+1}, E_{j+1}]) \\
&= \frac{1}{2}|\widehat{S}^{-1}_{j+1,-}|w([S_{j+1,-}]) \\
&\approx \frac{1}{2}h_j|\widehat{S}^{-1}_{j+1,-}|w\left(\frac{\partial f}{\partial y}([y^{(0)}_{j+1}])\right) \\
&= O(h_j\|w([y^{(0)}_{j+1}])\|).
\end{aligned}$$

If $\|w([y^{(0)}_{j+1}])\| = O(h_j^{q+2})$ (see Algorithm 4.2 and §3.2.4) then

$$|I - \widehat{S}^{-1}_{j+1,-}[S_{j+1,-}]|w([y^{(0)}_{j+1}]) = O(h_j\|w([y^{(0)}_{j+1}])\|^2) = O(h_j^{2q+4}). \tag{4.3.29}$$

If, for example, $p = q = (k-1)/2$, then $O(h_j^{2q+4}) = O(h_j^{k+3})$, which is two orders higher than the order of the truncation error in the ITS method.

The term $|(\widehat{S}^{-1}_{j+1,-}[S_{j,+}])A_j|w([r_j])$**.** In the IHO method,

$$\begin{aligned}
\widehat{S}^{-1}_{j+1,-}[S_{j,+}] &\approx \left(I + m\left(c_1^{q,p}h_j\frac{\partial f}{\partial y}([y^{(0)}_{j+1}])\right)\right)\left(I + c_1^{p,q}h_j\frac{\partial f}{\partial y}([y_j])\right) \\
&= I + c_1^{q,p}h_j\, m\left(\frac{\partial f}{\partial y}([y^{(0)}_{j+1}])\right) + c_1^{p,q}h_j\frac{\partial f}{\partial y}([y_j]) + O(h_j^2),
\end{aligned} \tag{4.3.30}$$

while in the ITS method,

$$[S_j] = I + h_j\frac{\partial f}{\partial y}([y_j]) + O(h_j^2). \tag{4.3.31}$$

Assuming that the Jacobian $\partial f/\partial y$ does not change significantly from step to step, we have from (4.3.30) and (4.3.31),

$$\widehat{S}^{-1}_{j+1,-}[S_{j,+}] \approx I + c_1^{q,p}h_j\frac{\partial f}{\partial y}([y_j]) + c_1^{p,q}h_j\frac{\partial f}{\partial y}([y_j]) + O(h_j^2) = [S_j]$$

$(c_1^{q,p} + c_1^{p,q} = 1)$. Therefore, we should expect

$$[S_j] \approx \widehat{S}_{j+1,-}^{-1}[S_{j+1}]. \tag{4.3.32}$$

Comparing (4.3.26) and (4.3.27), and taking into account (4.3.28), (4.3.29), and (4.3.32), we conclude that the propagation of the set $\{y_j - \hat{y}_j = A_j r_j \mid r_j \in [r_j]\}$ is similar in the IHO and Lohner's methods, but the truncation error can be much smaller in the former than in the latter.

4.3.4 Work per Step

We briefly discuss the most expensive parts of the ITS and IHO methods: generating high-order Jacobians, matrix-matrix multiplications, and enclosing the inverse of a point matrix. We measure the work by the number of floating-point operations. However, the time spent on memory operations may not be insignificant for the following reasons.

- The packages for automatic differentiation are often implemented through operator overloading [5], [6], [25], which may involve many memory allocations and deallocations.

- In generating Taylor coefficients, there may be a significant overhead caused by reading and storing the Taylor coefficients, $f^{[i]}$, and their Jacobians [22].

Generating High-Order Jacobians

To obtain an approximate bound for the number of floating point operations to generate $(k-1)$ Jacobians, $\partial f^{[i]}/\partial y$ for $i = 1, \ldots, (k-1)$, we assume that they are computed by differentiating the code list of the corresponding $f^{[i]}$ and using information from the previously computed $\partial f^{[l]}/\partial y$, for $l = 1, \ldots, (i-1)$. The FADBAD/TADIFF [5], [6] and IADOL-C [31] packages compute $\partial f^{[i]}/\partial y$ by differentiating the code list of $f^{[i]}$ (IADOL-C is an interval version of ADOL-C [25]). We also assume that the cost of evaluating $\partial f^{[i]}/\partial y$ is roughly n times the cost of evaluating $f^{[i]}$, [22].

For simplicity, suppose that f contains only arithmetic operations. If N is the number of operations, and $c_f \geq 0$ is the ratio of multiplications and divisions to additions and subtractions in these N operations, then to generate k coefficients $f^{[i]}$, $i = 1, \ldots, k$, we need $c_f N k^2 + O(Nk)$ operations [50, pp. 111–112] (see Appendix A).

Let $\mathrm{Ops}\left(f^{[i]}\right)$ be the number of arithmetic operations in the code list for evaluating $f^{[i]}$ from the already computed Taylor coefficients. In Appendix A, we show that

$$\mathrm{Ops}\left(f^{[i]}\right) = 2c_f N i + O(N), \quad \text{for } i > 0.$$

Since

$$\sum_{i=1}^{k-1} n \mathrm{Ops}\left(f^{[i]}\right) = n \sum_{i=1}^{k-1} \big(2c_f N i + O(N)\big) = c_f n N k^2 + O(nNk),$$

to generate $k-1$ Jacobians in an ITS method, we use

$$c_f n N k^2 + O(nNk) \tag{4.3.33}$$

arithmetic operations. Let $p = q$ and $k = p + q + 1$. In the IHO method we generate $p = (k-1)/2$ terms for the forward solution and $q = p = (k-1)/2$ terms for the backward one. The corresponding work is

$$c_f n N k^2/2 + O(nNk). \tag{4.3.34}$$

That is, the IHO method requires about half as much work as the ITS method of the same order to generate high-order Jacobians.

Matrix Inverses and Matrix-Matrix Multiplications

In Lohner's method and in the IHO method with the QR-factorization technique, we compute an enclosure of the inverse of a point matrix, which is a floating-point approximation to an orthogonal matrix. However, in the IHO method, we also enclose the inverse of a point matrix (see §4.1.3). In general, enclosing the inverse of an arbitrary point matrix is more expensive than enclosing the inverse of a floating-point approximation to an orthogonal matrix. However, we can still enclose the inverse of an arbitrary point matrix in $O(n^3)$ operations [2].

Lohner's method has 2 matrix-matrix multiplications, while the IHO method has 6 matrix-matrix multiplications.

To summarize, in the IHO method, we reduce the work for generating Jacobians, but increase the number of matrix operations. Suppose that $N \approx n^2$. This number can be easily achieved if each component of f contains approximately n operations, as happens, for example, in N-body problems. Then, (4.3.33) and (4.3.34) become

$$c_f n^3 k^2 + O(n^3 k) \quad \text{and} \quad c_f n^3 k^2/2 + O(n^3 k).$$

Therefore, we should expect the IHO method to outperform ITS methods in terms of the amount of work per step when the right side of the problem contains many terms. If the right side contains a few terms only, an ITS method may be less expensive for low orders, but we expect that the IHO method will perform better for higher orders. Note also that we expect the IHO method to allow larger stepsizes for methods of the same order, thus saving computation time during the whole integration. In addition, the IHO method (with $p = q$) needs half the memory for storing the point Taylor coefficients and the high-order Jacobians.

In §8.3.2, we study empirically the amount of work per step on Van der Pol's equation.

Chapter 5

A Taylor Series Method for Validation

We introduce a Taylor series method that is based on the validation test suggested by Moore [50, pp. 100–103] (see also [13] and [52]) for proving existence and uniqueness of the solution. Our goal is to obtain a method that validates existence and uniqueness with the supplied stepsize, if possible, or a stepsize that is not much smaller than the supplied one. Furthermore, we want to avoid as many stepsize reductions in this method as possible.

Usually, a Taylor series method for validation enables larger stepsizes than the constant enclosure method, which has been used in the past [44], [69]. As we pointed out in §3.1, the constant enclosure method restricts the stepsizes to Euler steps. We also combine better algorithms for computing tight enclosures, such as Lohner's method and the IHO method, with our algorithm for validating existence and uniqueness. As a result, we obtain a method that behaves similarly to the traditional numerical methods for IVPs for ODEs in the sense that the stepsize is controlled more by the accuracy requirements of Algorithm II than by restrictions imposed by Algorithm I.

Section 5.1 defines the validation problem; Section 5.2 describes how to compute an initial guess for the a priori enclosure; and Section 5.3 gives an algorithmic description of the method we propose.

5.1 The Validation Problem

Let $y_j \in [\tilde{y}_j]$ and no component of y_j is an endpoint of the corresponding component of $[\tilde{y}_j]$. If

$$y_j + \sum_{i=1}^{k-1} (t-t_j)^i f^{[i]}(y_j) + (t-t_j)^k f^{[k]}([\tilde{y}_j]) \subseteq [\tilde{y}_j] \tag{5.1.1}$$

for $t \in [t_j, t_{j+1}]$, it can be shown [13] that the problem $y'(t) = f(y)$, $y(t_j) = y_j$ has a unique solution

$$y(t; t_j, y_j) \in y_j + \sum_{i=1}^{k-1} (t-t_j)^i f^{[i]}(y_j) + (t-t_j)^k f^{[k]}([\tilde{y}_j]) \quad \text{for} \quad t \in [t_j, t_{j+1}].$$

For an interval $[y_j]$, the condition (5.1.1) translates to

$$[y_j] + \sum_{i=1}^{k-1} (t-t_j)^i f^{[i]}([y_j]) + (t-t_j)^k f^{[k]}([\tilde{y}_j]) \subseteq [\tilde{y}_j]. \tag{5.1.2}$$

To find the largest $t_{j+1} > t_j$ such that (5.1.2) holds for all $t \in [t_j, t_{j+1}]$, we have to compute rigorous lower bounds for the positive real roots of $2n$ algebraic equations, which are determined from (5.1.2). This task is not trivial to carry out.

However, since $t - t_j \in [0, h_j]$ for $t \in [t_j, t_{j+1}]$, if h_j is such that

$$[y_j] + \sum_{i=1}^{k-1} [0, h_j^i] f^{[i]}([y_j]) + [0, h_j^k] f^{[k]}([\tilde{y}_j]) \subseteq [\tilde{y}_j] \tag{5.1.3}$$

holds, then (5.1.2) holds for all $t \in [t_j, t_{j+1}]$. Verifying (5.1.3) is not difficult, and our validation procedure is based on (5.1.3). Given $[y_j]$ at t_j and a stepsize h_j, we want to find $[\tilde{y}_j]$ such that (5.1.3) is satisfied. Usually, h_j is predicted from the previous step. In the validation step, we try to verify existence and uniqueness with this h_j. If we cannot verify with h_j, we try to verify with a smaller stepsize than h_j.

Before we consider how to implement a method based on (5.1.3), we illustrate this approach with a few examples.

Consider

$$y' = y, \quad y(0) = 1 \tag{5.1.4}$$

and let $[\tilde{y}_0] = [1, 2]$. Then (5.1.3) on (5.1.4) with $[\tilde{y}_0] = [1, 2]$ gives

$$1 + [0, h_0] + [0, h_0^2/2] + \cdots + [0, h_0^{k-1}/(k-1)!] + [0, h_0^k/k!][1, 2] \subseteq [1, 2],$$

which is satisfied if

$$1 + h_0 + h_0^2/2 + \cdots + h_0^{k-1}/(k-1)! + 2h_0^k/k! \le 2. \tag{5.1.5}$$

For $k = 1$ and 3, (5.1.5) holds for $h_0 \le 0.5$ and $h_0 \le 0.63$, respectively.

Now, let $[\tilde{y}_0] = [1, 8]$. The inclusion (5.1.3) holds if

$$1 + h_0 + h_0^2/2 + \cdots + h_0^{k-1}/(k-1)! + 8h_0^k/k! \le 8. \tag{5.1.6}$$

For $k = 1$ and 3, (5.1.6) holds for $h_0 \le 0.875$ and $h_0 \le 1.48$, respectively.

Here, we can compute larger stepsizes with wider a priori bounds. With a variable stepsize control, we normally control the local excess per unit step (LEPUS), such that LEPUS is less than some tolerance (see Chapter 6). Depending on the tolerance, we can afford wider a priori bounds. For example, suppose that Algorithm II uses Taylor series of order $k = 15$. Then, LEPUS is given by $(h_0^{14}/15!)w([\tilde{y}_0])$. With $h_0 = 0.63$ and $[\tilde{y}_0] = [1, 2]$, LEPUS $\approx 1.2 \times 10^{-15}$, and with $h_0 = 1.48$ and $[\tilde{y}_0] = [1, 8]$, LEPUS $\approx 1.3 \times 10^{-9}$. If the tolerance is 10^{-8}, we can use $h_0 = 1.48$ and $[\tilde{y}_0] = [1, 8]$.

Consider

$$y' = -y, \quad y(0) = 1 \tag{5.1.7}$$

and let $[\tilde{y}_0] = [0.5, 1.5]$. For $k = 1$, we obtain from (5.1.3) the constant enclosure method:

$$1 + [0, h_0][-1.5, -0.5] \subseteq [0.5, 1.5],$$

from which we determine $h_0 \le 1/3$.

For $k = 2$, (5.1.3) becomes

$$1 - [0, h_0] + [0, h_0^2/2][0.5, 1.5] \subseteq [0.5, 1.5],$$

which is satisfied for $h_0 \le 0.5$.

For $k = 3$, (5.1.3) becomes

$$1 - [0, h_0] + [0, h_0^2/2] - [0, h_0^3/6][0.5, 1.5] \subseteq [0.5, 1.5],$$

which is satisfied for $h_0 \le 0.47$.

In this example, the maximum stepsize with $k = 3$ is smaller than with $k = 2$. The reason is that we ensure (5.1.1) by verifying (5.1.3). If we solve (5.1.1) directly, then we are often able to verify existence and uniqueness on larger intervals. For example, (5.1.1) for problem (5.1.7) and $k = 3$ reduces to

$$1 - t + t^2/2 - [0.5t^3/6, 1.5t^3/6] \subseteq [0.5, 1.5],$$

which holds for t satisfying

$$1 - t + t^2/2 - 1.5t^3/6 \ge 0.5 \quad \text{and} \quad 1 - t + t^2/2 - 0.5t^3/6 \le 1.5. \tag{5.1.8}$$

These inequalities are true for $t \le 0.63$. Note that the inequalities in (5.1.8) are more similar to stability conditions than Euler-type stepsize restrictions.

To summarize, by computing t_{j+1} such that (5.1.2) holds for all $t \in [t_j, t_{j+1}]$, we can often take larger stepsizes than with the constant enclosure method. The stepsize restriction imposed by (5.1.2) is more a "stability-type" than an Euler-type restriction. To implement (5.1.2) is more difficult than to implement (5.1.3). The latter often allows larger stepsizes than the constant enclosure method, although the stepsizes are generally smaller than the ones permitted by (5.1.2).

5.2 Guessing an Initial Enclosure

Suppose that we have computed $[\beta_j]$ such that

$$f^{[k]}([\tilde{y}_j]) = f^{[k]}\left([y_j] + \sum_{i=1}^{k-1}[0, h_j^i]f^{[i]}([y_j]) + [0, h_j^k][\beta_j]\right) \subseteq [\beta_j], \tag{5.2.1}$$

where

$$[\tilde{y}_j] = [y_j] + \sum_{i=1}^{k-1}[0, h_j^i]f^{[i]}([y_j]) + [0, h_j^k][\beta_j]. \tag{5.2.2}$$

Then, using (5.1.3) and (5.2.1),

$$\begin{aligned}
[y_j] + &\sum_{i=1}^{k-1}[0, h_j^i]f^{[i]}([y_j]) + [0, h_j^k]f^{[k]}([\tilde{y}_j]) \\
&\subseteq [y_j] + \sum_{i=1}^{k-1}[0, h_j^i]f^{[i]}([y_j]) + [0, h_j^k][\beta_j] \\
&= [\tilde{y}_j]\,.
\end{aligned}$$

Therefore, if $[\beta_j]$ is such that (5.2.1) holds, then (5.1.2) is satisfied, and there exists a unique solution

$$y(t; t_j, y_j) \in [y_j] + \sum_{i=1}^{k-1}(t - t_j)^i f^{[i]}([y_j]) + (t - t_j)^k f^{[k]}([\tilde{y}_j])$$

to the problem $y'(t) = f(y)$, $y(t_j) = y_j$, for any $y_j \in [y_j]$ and all $t \in [t_j, t_{j+1}]$.

How to compute an approximation for $[\beta_j]$

Let $y_j \in [y_j]$ and $t \in [0, h_j]$. Consider the nonlinear system of equations for β_j,

$$f^{[k]}\left(y_j + \sum_{i=1}^{k-1} t^i f^{[i]}(y_j) + t^k\beta_j\right) = \beta_j. \tag{5.2.3}$$

Ideally, we want to find an enclosure of the set of values for β_j such that (5.2.3) holds for all $y_j \in [y_j]$ and $t \in [0, h_j]$. In practice, computing such a set may be expensive.

Here, we suggest a simple method for computing an approximation to this set. From (5.2.3),

$$\begin{aligned}\beta_j &= f^{[k]}\left(y_j + \sum_{i=1}^{k-1} t^i f^{[i]}(y_j) + t^k\beta_j\right)\\ &\approx f^{[k]}(y_j) + \frac{\partial f^{[k]}(y_j)}{\partial y}\left(\sum_{i=1}^{k-1} t^i f^{[i]}(y_j) + t^k\beta_j\right)\\ &= f^{[k]}(y_j) + \frac{\partial f^{[k]}(y_j)}{\partial y}\sum_{i=1}^{k-1} t^i f^{[i]}(y_j) + \frac{\partial f^{[k]}(y_j)}{\partial y}t^k\beta_j\end{aligned} \tag{5.2.4}$$

and therefore,

$$\left(I - t^k\frac{\partial f^{[k]}(y_j)}{\partial y}\right)\beta_j \approx f^{[k]}(y_j) + \frac{\partial f^{[k]}(y_j)}{\partial y}\sum_{i=1}^{k-1} t^i f^{[i]}(y_j). \tag{5.2.5}$$

Since we are interested in computing an approximation to the set containing β_j, we can compute from (5.2.5),

$$[\beta_j] = \left(I + [0, h_j^k]\frac{\partial f^{[k]}([y_j])}{\partial y}\right)\left(f^{[k]}([y_j]) + \frac{\partial f^{[k]}([y_j])}{\partial y}\sum_{i=1}^{k-1}[0, h_j^i] f^{[i]}([y_j])\right).$$

Since $[\beta_j]$ is an approximation[1], in the algorithm that we describe in the next section, we inflate $[\beta_j]$ to reduce the likelihood of failure in (5.2.1).

In (5.2.4), we could have used the approximation

$$\begin{aligned}f^{[k]}\left(y_j + \sum_{i=1}^{k-1} t^i f^{[i]}(y_j) + t^k\beta_j\right) &\approx f^{[k]}\left(y_j + \sum_{i=1}^{k-1} t^i f^{[i]}(y_j)\right)\\ &\quad + \frac{\partial f^{[k]}}{\partial y}\left(y_j + \sum_{i=1}^{k-1} t^i f^{[i]}(y_j)\right)t^k\beta_j,\end{aligned} \tag{5.2.6}$$

which is perhaps a better approximation than (5.2.4). However, if we use (5.2.6), we have to generate the coefficients $\partial f^{[i]}/\partial y$ evaluated at $[y_j] + \sum_{i=1}^{k-1}[0, h_j^i] f^{[i]}([y_j])$, for $i = 1, \ldots, k$, while in (5.2.4), we need $f^{[k]}/\partial y$ evaluated at $[y_j]$, which coefficients can be reused in Algorithm II (see the next section).

[1]Note that $[\beta_j]$ is a guess for the enclosure of the kth Taylor coefficient, not a rigorous enclosure.

5.3 Algorithmic Description of the Validation Method

The method that we propose is described in Algorithm 5.1. Here, we explain some of the decisions we have made in designing it.

Input part If we use an ITS method with order k of the Taylor series, we have to compute the coefficients $f^{[i]}([y_j])$ and $\partial f^{[i]}([y_j])/\partial y$, for $i = 1, \ldots, k-1$, in Algorithm II. Therefore, we can use $f^{[i]}([y_j])$, for $i = 1, \ldots, k-1$, in Algorithm I without doing additional work to generate them. However, we have to compute $f^{[k]}([y_j])$ and $\partial f^{[k]}([y_j])/\partial y$.

If we use a (p, q) IHO method, we have to generate, in addition to the coefficients $f^{[i]}([y_j])$ for $i = 1, \ldots, q$, the coefficients $f^{[i]}([y_j])$ for $q = i+1, \ldots, k$ and $\partial f^{[k]}([y_j])/\partial y$.

Compute part In line 7, we inflate $[\beta_j]$. Since it is already an approximation to the enclosure of the kth Taylor coefficient on $[t_j, t_{j+1}]$, by inflating $[\beta_j]$, we hope to enclose this coefficient on $[t_j, t_{j+1}]$. We choose $\epsilon = 1$, but we can use other values instead. With $\epsilon = 1$, we add $[-|\beta_j|, |\beta_j|]$ to $[\beta_j]$. Since $[\beta_j]$ is multiplied by $[0, h_j^k]$, adding $[-|\beta_j|, |\beta_j|]$ to $[\beta_j]$ does not contribute significantly to the widths of the components of $[\tilde{y}_j^{(0)}]$.

If the condition in line 11 is satisfied, then we have verified existence and uniqueness with the computed $[\tilde{y}_j^0]$ in line 9. Otherwise, in line 15, we compute a new guess $[\tilde{y}_j^0]$ for the initial enclosure. Then, in the second while loop (line 18), we try to validate with order $s := l \leq k$. If we succeed, then in the third while loop (line 29), we try to improve the enclosure with the order s, with which we have verified existence and uniqueness. Otherwise, in line 38, we reduce the stepsize. If this is the second reduction, we start the computations from the beginning (line 4); otherwise, we repeat the while loop at line 18 with a smaller stepsize. The reason for starting the computations at line 4 after the second stepsize reduction is to try with a new guess for the a priori enclosure, before continuing with further stepsize reductions.

Algorithm 5.1 Validate existence and uniqueness with Taylor series.

INPUT:

$[y_j]$, h_j, k, $h_{\min}$, $\alpha = 0.8$, $\epsilon = 1$;
$\frac{\partial f^{[k]}([y_j])}{\partial y}$, $f^{[i]}([y_j])$, for $i = 1, \dots, (k-1)$.

COMPUTE:

`Verified` $:=$ **false** ;
while $h_j \geq h_{\min}$ **and not** `Verified` **do**
  $[v_j] := \sum_{i=1}^{k-1} [0, h_j^i] f^{[i]}([y_j])$;
  $[\beta_j] := \left(I + [0, h_j^k] \frac{\partial f^{[k]}([y_j])}{\partial y}\right)\left(f^{[k]}([y_j]) + \frac{\partial f^{[k]}([y_j])}{\partial y}[v_j]\right)$;
  $[\beta_j] := [\beta_j] + [-\epsilon, \epsilon] \cdot |[\beta_j]|$;
  $[u_j] := [y_j] + [v_j]$;
  $[\tilde{y}_j^{(0)}] := [u_j] + [0, h_j^k][\beta_j]$;
  Generate $f^{[i]}([\tilde{y}_j^{(0)}])$, for $i = 1, \dots, k$;
  if $f^{[k]}([\tilde{y}_j^{(0)}]) \subseteq [\beta_j]$ **then**
    $[\tilde{y}_j] := [u_j] + [0, h_j^k] f^{[k]}([\tilde{y}_j^{(0)}])$;
    break ;
  end-if
  $[\tilde{y}_j^{(0)}] := [u_j] + [0, h_j^k] f^{[k]}([\tilde{y}_j^{(0)}])$;
  Generate $f^{[i]}([\tilde{y}_j^{(0)}])$, for $i = 1, \dots, k$;
  `Reduced` $:= 0$;
  while not `Verified` **and** `Reduced` < 2 **do**
    for $l = 1$ **to** k **do**
      $[v_j] := [y_j] + \sum_{i=1}^{l-1} [0, h_j^i] f^{[i]}([y_j])$;
      $[\tilde{y}_j] := [v_j] + [0, h_j^l] f^{[l]}[\tilde{y}_j^{(0)}]$;
      if $[\tilde{y}_j] \subseteq [\tilde{y}_j^{(0)}]$ **then**
        `Verified` $:=$ **true** ; $s := l$;
        break ;
      end-if
    end-for

CONTINUED ON THE NEXT PAGE...

Algorithm 5.1 Continued

    if `Verified` **then**
        `Improving` := **true** ;
        while `Improving` **do**
            Generate $f^{[i]}([\tilde{y}_j])$, for $i = 1, \ldots, s$;
            $[\tilde{y}_j^{(0)}] := [v_j] + [0, h_j^s] f^{[s]}([\tilde{y}_j])$;
            if $\|w([\tilde{y}_j])\| / \|w([\tilde{y}_j^{(0)}])\| > 1.01$ **then**
                $[\tilde{y}_j] := [\tilde{y}_j^{(0)}]$;
            else
                `Improving` := **false** ;
            end
        end-while
    else
        $h_j := \alpha h_j$;
        `Reduced` := `Reduced` + 1;
    end-if
end-while
if $h_j < h_{\min}$ **then**
    print "Stepsize too small: cannot verify existence and uniqueness";
    exit ;
end-if
Compute $h_j^i f^{[i]}([\tilde{y}_j])$, for $i = 1, \ldots, k$.

OUTPUT:

$[\tilde{y}_j]$, h_j;
$h_j^i f^{[i]}([\tilde{y}_j])$, for $i = 1, \ldots, k$.

We do not halve the stepsize, but reduce it by multiplying by α, which we choose to be 0.8. As with ϵ, the value that we choose for α is somewhat arbitrary, but we want it to be closer to 1 than to 0.5. We have not thoroughly studied the influence of the choice for ϵ and α on the performance of Algorithm 5.1.

Chapter 6

Estimating and Controlling the Excess

In §3.2.4, we considered the local excess in one step of the ITS methods discussed in this thesis. The IHO method has the same sources of local excess as the ITS methods, but in the IHO method, we also enclose the solution of the nonlinear system (4.1.10). Since the excess that arises from enclosing the solution of this nonlinear system is usually small (see §4.3.3), we do not discuss the local excess in the IHO method.

In §6.1, we define local and global excess and discuss controlling the global and estimating the local excess. In §6.2, we propose a simple stepsize control based on controlling an approximation of the local excess.

6.1 Local and Global Excess

Let the set $\mathcal{U}_j$ be an enclosure of the solution at t_j. In this thesis, $\mathcal{U}_j$ is represented by an interval vector or a parallelepiped. We define local and global excess by

$$\epsilon_j = q(\mathcal{U}_j, y(t_j; t_{j-1}, \mathcal{U}_{j-1})) \quad \text{and} \tag{6.1.1}$$

$$\gamma_j = q(\mathcal{U}_j, y(t_j; t_0, [y_0])), \tag{6.1.2}$$

respectively [19, p. 87, p. 100], [71], where $q(\cdot,\cdot)$ is the Hausdorff distance between two sets given by (2.2.1).

6.1.1 Controlling the Global Excess

Similar to the standard numerical methods for IVPs for ODEs, our approach in VNODE is to allow the user to specify a tolerance *Tol*. Then the code tries to produce enclosures, at points t_j, such that

$$\gamma_j \approx C_j Tol \quad \text{for } j \geq 1, \tag{6.1.3}$$

where C_j is an unknown constant that depends on the problem and the length of the interval of integration, but not *Tol*. We try to achieve (6.1.3) by controlling the local excess per unit step (LEPUS) [71]. That is, we require

$$\epsilon_j \leq h_{j-1} Tol \tag{6.1.4}$$

on each step. Eijgenraam shows [19, p. 115] that

$$\gamma_j \leq \sum_{r=1}^{j} e^{\alpha(t_j - t_r)} \epsilon_r, \tag{6.1.5}$$

where α is a constant depending on the problem. This constant may be negative since the logarithmic norm is used in its definition [19, p. 46]. Using (6.1.4), we obtain from (6.1.5) that

$$\gamma_j \leq \begin{cases} e^{\alpha(t_j - t_1)}(t_j - t_0) Tol, & \text{if } \alpha > 0; \\ (t_j - t_0) Tol, & \text{if } \alpha \leq 0. \end{cases} \tag{6.1.6}$$

Therefore, by controlling LEPUS, we can obtain a bound for the global excess. In this sense, by reducing Tol, we should compute tighter bounds.

6.1.2 Estimating the Local Excess

From §3.2.4, the local excess in an ITS method is given by

$$O(h_j \|w([y_j])\|^2) + O(h_j^{k+1}) + (\text{higher-order terms}). \tag{6.1.7}$$

To compute an estimate of the local excess, we have to determine the dominating term in (6.1.7). Obviously, if point initial conditions are specified, $w([y_0]) = 0$, the excess in each component of the computed solution at t_1 is at most $h_0^k \|w(f^{[k]}([\tilde{y}_0])\|$. Unfortunately, even if we start the integration with point initial conditions, $[y_j]$ is usually a non-degenerate interval vector ($\|w\left([y_j]\right)\| > 0$) on the second and all succeeding steps. If $\|w([y_j])\|$ is not small, then $h_j \|w([y_j])\|^2$ may be the dominating term in (6.1.7). Because of this term, the methods discussed in this thesis are first order methods if $h_j \|w([y_j])\|^2$ is dominating.

Eijgenraam shows an example for which the overestimation on each step is at least $ch_j \|w([y_j])\|^2$, for some constant $c > 0$ [19, pp. 127-128]. We discuss his example in the next subsection.

6.1.3 Worst Case Example

Consider the IVP problem

$$\begin{aligned} y_1' &= 0 \\ y_2' &= y_1^2 \end{aligned} \tag{6.1.8}$$

with initial conditions

$$y(0) \in [y_0] = \begin{pmatrix} [-\lambda, \lambda] \\ 0 \end{pmatrix}, \qquad 0 < \lambda \leq \lambda_0. \tag{6.1.9}$$

At $t = h_0 > 0$, the set of solutions of (6.1.8–6.1.9) is

$$y(h_0; 0, [y_0]) = \left\{ \begin{pmatrix} \mu \\ h_0 \mu^2 \end{pmatrix} \mid \mu \in [-\lambda, \lambda] \right\}$$

(see Figure 6.1). Suppose that we use a convex set $\mathcal{Y}_1$ to enclose $y(h_0; 0, [y_0])$. Since for each two points in a convex set, the line connecting them must be in the set too, we have

$$\frac{1}{2} \begin{pmatrix} -\lambda \\ h_0 \lambda^2 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} \lambda \\ h_0 \lambda^2 \end{pmatrix} = \begin{pmatrix} 0 \\ h_0 \lambda^2 \end{pmatrix} \in \mathcal{Y}_1.$$

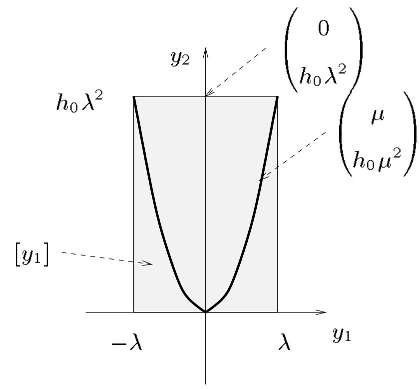


Figure 6.1: If we use an interval vector to enclose the solution, the overestimation measured in the Hausdorff distance is at least $ch_0\|w([y_0])\|^2$.

It can be shown that

$$\begin{aligned}
q\left(\mathcal{Y}_1, y(h_0;0,[y_0])\right) &\geq q\left(\begin{pmatrix} 0 \\ h_0\lambda^2 \end{pmatrix}, \left\{\begin{pmatrix} \mu \\ h_0\mu^2 \end{pmatrix} \mid \mu \in [-\lambda,\lambda]\right\}\right) \\
&\geq \min_{\mu\in[-\lambda,\lambda]} \|(-\mu, h_0\lambda^2 - h_0\mu^2)^T\| \\
&= \min_{0\leq\mu\leq\lambda} \max(\mu, h_0\lambda^2 - h_0\mu^2) \\
&= \frac{2}{1+\sqrt{1+4h_0^2\lambda^2}} h_0\lambda^2 \\
&\geq \frac{1}{2\left(1+\sqrt{1+4h_{\max}^2\lambda_0^2}\right)} h_0\|w([y_0])\|^2 \\
&= ch_0\|w([y_0])\|^2,
\end{aligned} \tag{6.1.10}$$

where $h_0 \leq h_{\max}$, and $h_{\max}$ is the maximum stepsize taken during the integration. Therefore, if we use a convex set to enclose the set of solutions at $t = h_0 > 0$, we have an overestimation that is at least $ch_0\|w([y_0])\|^2$, independently of the method used for computing the enclosing set.

If we use an interval vector to enclose the solution of (6.1.8–6.1.9) at h_0, the tightest enclosure we can have is

$$[y_1] = \begin{pmatrix} [-\lambda,\lambda] \\ h_0[0,\lambda^2] \end{pmatrix}, \tag{6.1.11}$$

which gives an overestimation of at least $ch_0\|w([y_0])\|^2$. The reason for this pessimistic result is that by using intervals, we treat each of the components of the solution of (6.1.8–6.1.9) independently, while the second component of the true solution depends on the first one.

The approach of Berz [7] uses multivariate high-order Taylor series with respect to time and the initial conditions to keep functional dependencies. As a result, his method is a higher order method with respect to the propagated enclosures. However, it requires arithmetic with Taylor polynomials, which involves significantly more work and memory.

In the example discussed, we cannot control LEPUS based on estimating the local excess in the Hausdorff distance. For example, if $h_{\max} = h_0 = 0.1$ and $\lambda_0 = 0.1$, then using (6.1.10), LEPUS at h_0 is

$$\frac{q\left(\mathcal{Y}_1, y(h_0; 0, [y_0])\right)}{h_0} \gtrsim 0.01,$$

independently of the stepsize h_0. Here, if the tolerance is small, an interval method may give up.

6.2 A Simple Stepsize Control

We assume that we solve problems with either point initial conditions or interval initial conditions with sufficiently small widths.

Let the enclosure of the remainder term on the jth step be given by

$$\gamma h_{j-1}^k f^{[k]}([\tilde{y}_{j-1}]),$$

where $\gamma > 0$ is a constant. ($\gamma = 1$ for ITS methods, and $\gamma < 1$ for the IHO method.) Then, we approximate the local excess on each step by

$$err_j = \gamma h_{j-1}^k \|w(f^{[k]}([\tilde{y}_{j-1}]))\|. \tag{6.2.1}$$

Given a tolerance Tol, we try to control LEPUS by requiring

$$err_j \le h_{j-1} Tol \tag{6.2.2}$$

on each step.

6.2.1 Predicting a Stepsize after an Accepted Step

Suppose that the jth step was accepted, and we predict a stepsize for the next step by

$$h_{j,0} = h_{j-1} \left(\frac{h_{j-1} Tol}{err_j} \right)^{1/(k-1)}. \tag{6.2.3}$$

Assuming that

$$\|w(f^{[k]}([\tilde{y}_{j-1}]))\| \approx \|w(f^{[k]}([\tilde{y}_j]))\|,$$

and using (6.2.1), we have for the excess with $h_{j,0}$,

$$\begin{aligned} err_{j+1,0} = \gamma h_{j,0}^k \|w(f^{[k]}([\tilde{y}_j]))\| &= \gamma h_{j,0} h_{j-1}^{k-1} \frac{h_{j-1} Tol}{err_j} \|w(f^{[k]}([\tilde{y}_j]))\| \\ &\approx h_{j,0} h_{j-1}^{k-1} \frac{\gamma h_{j-1} Tol}{err_j} \|w(f^{[k]}([\tilde{y}_{j-1}]))\| \\ &= h_{j,0} Tol \frac{\gamma h_{j-1}^k \|w(f^{[k]}([\tilde{y}_{j-1}]))\|}{err_j} \\ &\approx h_{j,0} Tol. \end{aligned}$$

In practice, we predict the stepsize by

$$h_{j,0} = 0.9\, h_{j-1} \left(\frac{0.5\, h_{j-1} Tol}{err_j} \right)^{1/(k-1)}, \tag{6.2.4}$$

where we aim at $0.5 Tol$ and choose a "safety" factor of 0.9.

Algorithm I may reduce the stepsize $h_{j,0}$. In which case, Algorithm II has to use a smaller stepsize $h_j \le h_{j,0}$.

6.2.2 Computing a Stepsize after a Rejected Step

If $err_j > h_{j-1} Tol$, we compute a new stepsize $h_{j-1,1}$ by using the equality

$$\gamma h_{j-1,1}^k \|w(f^{[k]}([\tilde{y}_{j-1}]))\| = h_{j-1,1} Tol, \tag{6.2.5}$$

from which we determine,

$$h_{j-1,1} = \left(\frac{Tol}{\gamma \|w(f^{[k]}([\tilde{y}_{j-1}]))\|}\right)^{1/(k-1)} = \left(\frac{h_{j-1}^k Tol}{\gamma h_{j-1}^k \|w(f^{[k]}([\tilde{y}_{j-1}]))\|}\right)^{1/(k-1)}$$
$$= h_{j-1}\left(\frac{h_{j-1}Tol}{err_j}\right)^{1/(k-1)}.$$

Therefore, if the stepsize is rejected, we compute a new stepsize by

$$h_{j-1,1} = h_{j-1}\left(\frac{h_{j-1}Tol}{err_j}\right)^{1/(k-1)}. \tag{6.2.6}$$

Since we reduce the stepsize, we can use the same $f^{[k]}([\tilde{y}_{j-1}])$: it is an enclosure on the interval $[0, h_{j-1}]$ and so must also be an enclosure over the smaller interval $[0, h_{j-1,1}]$. Therefore, we can compute a tight enclosure with $h_{j-1,1}$. Because of (6.2.5), we know that (6.2.2) holds.

Remark

If we want the inequality (6.2.2) to hold rigorously, we have to use directed roundings or interval arithmetic to compute (6.2.2) and (6.2.6). Otherwise, we can execute (6.2.2) and (6.2.6) in "regular" floating-point arithmetic.

Chapter 7

A Program Structure for Computing Validated Solutions

In this chapter, we describe a program structure for computing validated solutions of IVPs for ODEs. It combines algorithms for validating existence and uniqueness, computing a tight enclosure, and selecting a stepsize.

First, in §7.1, we specify the ODE problem. Then, in §7.2, we describe one step of an integration procedure. A problem can be integrated by a repeated execution of the code implementing one step. The structure that we propose in §7.2 is somewhat similar to the one discussed in [29].

7.1 Problem Specification

As in the classical methods, we have to specify the problem we want to integrate. A minimum set of parameters is:

n — number of equations;

f — function for computing the right side;

t_0 — initial point;

$[y_0]$ — initial condition at t_0;

T — end of the interval of integration; and

Tol — tolerance.

In addition to specifying the problem being integrated, we need functions for computing the Taylor coefficients $f^{[i]}$ and their Jacobians $\partial f^{[i]}/\partial y$, for $i > 0$. In VNODE, such functions are generated by an automatic differentiation package (see Appendix B).

7.2 One Step of a Validated Method

Our goal is to structure the integrator function such that parts of it can be replaced without changing the rest. In Algorithm 7.1, we show a general structure of a program for implementing one step of a validated method. Functionally, we divide our program into three modules, which are responsible for the following tasks.

MODULE 1: Validating existence and uniqueness and simultaneously computing an a priori enclosure of the solution on $[t_j, t_{j+1}]$ (Algorithm I).

MODULE 2: Tightening the enclosure at t_{j+1} (Algorithm II).

MODULE 3: Preparing for the next step, which includes estimating the excess, accepting or rejecting a step, and predicting a new order and stepsize.

At this stage, we do not have an order control strategy, but we include "order" in Module 3 to show where an order selection method would fit.

The VNODE package described in Appendix B implements the structure in Figure 7.1. Here, we briefly describe the modular structure.

Algorithm 7.1 One step of a validated method.

INPUT:

t_j, $h_{j,0}$, $h_{\min}$, $[y_j]$, $\hat{y}_j$, A_j, $[r_j]$;

Tol, k (p and q in an IHO method, $k = p + q + 1$).

COMPUTE:

MODULE 1:

Try to validate existence and uniqueness with $h_{j,0}$ and k.

if successful **then**

return $h_j \le h_{j,0}$, $[\tilde{y}_j]$, $[z_{j+1}] := h_j^k f^{[k]}([\tilde{y}_j])$;

end-if

while $h_j \ge h_{\min}$ **do**

MODULE 2:

Compute $[y_{j+1}]$, $\hat{y}_{j+1}$, A_{j+1}, and $[r_{j+1}]$.

MODULE 3:

Estimate the excess.

if the excess is acceptable **then**

select $h_{j+1,0}$ for the next step;

break ;

else

select new $h_{j,1} < h_j$;

$[z_{j+1}^1] := (h_{j,1}/h_j)^k [z_{j+1}]$;

$h_j := h_{j,1}$;

$[z_{j+1}] := [z_{j+1}^1]$;

end-if

end-while

if $h_j < h_{\min}$ **then**

print "Stepsize too small";

exit ;

end-if

OUTPUT:

t_{j+1}, $h_{j+1,0}$, $[y_{j+1}]$, $\hat{y}_{j+1}$, A_{j+1}, $[r_{j+1}]$.

Module 1

We try to validate existence and uniqueness with a stepsize $h_{j,0}$. If the validation is successful, we have as an output of this procedure a stepsize h_j, which can be smaller than $h_{j,0}$, an enclosure $[\tilde{y}_j]$ of the solution on $[t_j, t_j + h_j]$, and an enclosure of the kth Taylor coefficient multiplied by h_j^k, $[z_{j+1}] = h_j^k f^{[k]}([\tilde{y}_j])$. In our implementation, $h_j \leq h_{j,0}$ since $h_{j,0}$ is predicted such that the predicted error satisfies some tolerance. If the validation is unsuccessful, the code should print a message and exit the integration. For example, on the problem $y' = y^2$, $y(0) = 1$, a method for validating existence and uniqueness would normally start taking smaller and smaller stepsizes as t approaches 1. When the stepsize becomes smaller than a prescribed minimum, this method should stop and inform the user that it cannot continue (see Algorithm 5.1).

Remarks

1. If it is a first step, $h_{0,0}$ is a predicted initial stepsize; otherwise, $h_{j,0}$ is selected from the previous step. The algorithms for predicting an initial stepsize and selecting one after a successful step may differ.

2. It is convenient to return $h_j^k f^{[k]}([\tilde{y}_j])$ since it is used in computing $[y_{j+1}]$. In addition, if the stepsize is rejected, and we compute a new one $h_{j,1} < h_j$, we can make the excess term smaller by the scaling $[z_{j+1}^1] = (h_{j,1}/h_j)^k [z_{j+1}]$.

3. It is also convenient to have the terms $h_j^i f^{[i]}([\tilde{y}_j])$, for $i = 1, \ldots k$, available to Module 2. For example, the predictor and the corrector in the IHO method need $h_j^{q+1} f^{[q+1]}([\tilde{y}_j])$ and $h_j^k f^{[k]}([\tilde{y}_j])$, respectively.

Module 2

We use $[\tilde{y}_j]$ and $[z_{j+1}]$, from Module 1, to compute a tight enclosure of the solution at $t_{j+1} = t_j + h_j$. In the ITS methods, the local excess is approximated by $h_j^k \| w(f^{[k]}([\tilde{y}_j])) \|$.

In the IHO method, it is approximated by $\gamma h_j^k \| w(f^{[k]}([\tilde{y}_j])) \|$, for some constant γ, which depends on the p and q of the method. In the latter case, forming the term $\gamma h_j^k f^{[k]}([\tilde{y}_j])$ in this module is more convenient than in the first one.

Module 3

In VNODE, we estimate the local excess and compute new stepsizes as discussed in §6.2. Note that, if the step is not successful, we repeat the computation, but starting from Module 2. The reason is that we have already validated existence and uniqueness, and we have to compute an enclosure with a smaller stepsize. That is, we can use the output $h_j^k f^{[k]}([\tilde{y}_j])$ and scale it to $h_{j,1}^k f^{[k]}([\tilde{y}_j])$, where $h_{j,1} < h_j$.

Chapter 8

Numerical Results

With the numerical experiments described in this chapter, we study the behavior of our IHO method and compare it with the ITS methods.

In §8.1, we describe the tables shown in this chapter and introduce some assumptions.

In §8.2, we verify empirically that the order of the IHO method is $p+q+1$, which is the order of the truncation error. We also show empirically that the order of an ITS method with $k=p+q+1$ terms is k.

In §8.3, we examine the stability of these two methods on constant coefficient problems. Then, we compare the methods on nonlinear problems.

In §8.4, we compare the ITS and IHO methods again: first, by using a constant enclosure method in Algorithm I, and second, by using the Taylor series enclosure method (from Chapter 5). We also show that the Taylor series enclosure method enables larger stepsizes than the constant enclosure method.

8.1 Description of the Tables and Assumptions

Description of the Tables

We describe briefly some of the columns of the tables shown in this chapter.

h constant stepsize used during the integration.

Excess global excess at the end of the interval of integration.

If a point initial condition is specified, the global excess γ_j at a point t_j is measured by the norm of the width of the enclosure; that is,

$$\gamma_j = \|w([y_j])\|$$

$(j \geq 1)$. If an interval initial condition is given, and a closed form solution is known, then

$$\gamma_j = q\left([y_j], [y_j^{exact}]\right),$$

where $q\left([y_j], [y_j^{exact}]\right)$ is defined in (2.2.3). That is, we measure the global excess by the distance between $[y_j]$ and $[y_j^{exact}]$, where $[y_j^{exact}]$ is computed from the true solution. We assume that $[y_j^{exact}]$ is the tightest interval enclosure of the solution that can be obtained.

Time Total CPU time in seconds spent in Algorithm II. Since we are mainly interested in the performance of the methods implementing Algorithm II, we report only this time. Note that if the timing results are of order 10^{-2} or 10^{-3}, they may not be accurate. We have not measured the performance of Algorithm I because we have not yet optimized the Taylor series method from Chapter 5 to reuse the coefficients needed in Algorithm II.

H constant stepsize that is an input on each step to Algorithm I. This algorithm may reduce the stepsize in order to verify existence and uniqueness.

Steps number of steps used during the integration.

Assumptions

- We denote an ITS method with k terms by ITS(k) and an IHO method with parameters p and q by IHO(q, p). In all of the examples in this chapter, we use $p = q$ with $k = p + q + 1$. Thus, we compare methods with truncation errors of the same order.

- If necessary, both methods use Lohner's QR-factorization technique to reduce the wrapping effect. The ITS method with the QR-factorization is essentially Lohner's method.

- In the experiments with a variable stepsize control, we use Eijgenraam's method [19, pp. 129–136] for selecting initial stepsize.

- The implementation of the constant enclosure method (in Algorithm I) is as described in [19, pp. 59–67]. This implementation uses the Jacobian of f for computing an initial guess for the a priori enclosure.

- We compiled VNODE with the GNU C++ compiler version 2.7.2 on a Sun Ultra 2/2170 workstation with an 168 MHz UltraSPARC CPU. The underlying interval-arithmetic and automatic differentiation packages are PROFIL/BIAS [38] and FADBAD/TADIFF [5], [6], respectively (see §B.4.1 and §B.4.2).

8.2 Observed Orders

In this section, we determine empirically the error constants and orders of the ITS and IHO methods on a scalar (§8.2.1) and two-dimensional (§8.2.2) nonlinear problem. We have chosen nonlinear problems, because in the IHO method we have to enclose the solution of a nonlinear system, while we do not have to do that in the constant coefficient case. Our goal is to verify that the excess arising from solving such a system (see §4.3.3) does not reduce the order of the method.

For simplicity, we consider the case with point initial conditions. If $[\tilde{y}_j]$ is a good enclosure of $y(t; t_j, [y_j])$ on $[t_j, t_{j+1}]$, then the overestimation in $h_j^k f^{[k]}([\tilde{y}_j])$ is of order $O(h^{k+1})$ (see §3.2.4). Assuming that the computed intervals are small, the local excess in an ITS method and in an IHO method, with $k = p + q + 1$, is $O(h^{k+1})$ and the global excess should be $O(h^k)$.

For a given problem and method, we compute an error constant c and order r by a linear least squares fit determined from the conditions

$$\log(\text{Excess}_i) \approxeq \log c + r \log h_i, \quad i = 1, \ldots, s,$$

where Excess_i is the global excess at the endpoint obtained by integrating the problem with constant stepsizes h_i, $i = 1, 2, \ldots, s$.

Before we present our numerical results, we should note that the order of a validated method can be sensitive to the tightness of the a priori bounds. That is, for the same order of the truncation error of the underlying formula and ranges of stepsizes, depending on how tight the a priori bounds are, we may obtain different values for the order. For example, we may compute an order that is higher by two or three than that predicted theoretically.

To make the procedure for computing the order more deterministic, we can assume that we can obtain the tightest possible a priori bounds. Since in this thesis we use constant enclosures for the solution, the tightest constant enclosure on $[t_j, t_{j+1}]$ has components

$$[\min_{t \in [t_j, t_{j+1}],\ y_j \in [y_j]} y_i(t; t_j, y_j), \max_{t \in [t_j, t_{j+1}]\ y_j \in [y_j]} y_i(t; t_j, y_j)], \quad \text{for } i = 1, \ldots, n.$$

Note that in practice, it is normally difficult to compute such bounds. However, for the examples in the next two subsections, we use such optimal bounds.

8.2.1 Nonlinear Scalar Problem

We integrated

$$y' = -y^2, \quad y(0) = 1$$

on $[0, 12]$ with an ITS method with $k = 7$ and $k = 11$ and with an IHO method with $p = q = 3$ and $p = q = 5$, respectively ($k = p + q + 1$). We computed a priori enclosures of the solution on each step by the formula

$$[\tilde{y}_j] = [1/(h + 1/\underline{y}_j), \bar{y}_j],$$

which is obtained from the true solution, and used constant stepsizes $h = 0.1, 0.2, \ldots, 0.6$. Here and in §8.2.2, we select the endpoint, T, to be a multiple of the stepsizes. By computing a priori enclosures from the formula for the true solution, we eliminate the need to reduce the stepsize or to compute too wide (for this problem) a priori enclosures in Algorithm I.

Tables 8.1 and 8.2 show the excess at $T = 12$, the excess divided by h^7 and h^{11}, respectively, and the CPU time used in the ITS and IHO methods. By using a least squares fit, we have computed in Table 8.3 the error constants and orders corresponding to $k = 7$ and $k = 11$. For $k = 7$ and $k = 11$, we compute the base-10 logarithm of the data and plot in Figure 8.1 the excess versus the stepsize and the time versus the excess.

From Table 8.3, we observe higher orders for both methods than we would expect. Moreover, the observed orders of the IHO method on this example are bigger than the corresponding orders of the ITS method (for $k = 7$ and 11). From Figures 8.1(a) and 8.1(b), we see that for the same stepsizes, the IHO method produces enclosures that are of order 10^{-2} times tighter than the enclosures produced by the ITS method.

h	Excess		Excess/h^7		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
0.10	7.2×10^{-10}	2.3×10^{-11}	7.2×10^{-3}	2.3×10^{-4}	1.2×10^{-1}	1.2×10^{-1}
0.20	9.8×10^{-8}	3.6×10^{-9}	7.6×10^{-3}	2.8×10^{-4}	6.3×10^{-2}	6.4×10^{-2}
0.30	1.8×10^{-6}	7.3×10^{-8}	8.1×10^{-3}	3.4×10^{-4}	4.0×10^{-2}	4.0×10^{-2}
0.40	1.4×10^{-5}	6.9×10^{-7}	8.6×10^{-3}	4.2×10^{-4}	3.0×10^{-2}	3.1×10^{-2}
0.50	7.2×10^{-5}	4.3×10^{-6}	9.2×10^{-3}	5.5×10^{-4}	2.5×10^{-2}	2.4×10^{-2}
0.60	2.9×10^{-4}	2.1×10^{-5}	1.0×10^{-2}	7.4×10^{-4}	2.0×10^{-2}	2.1×10^{-2}

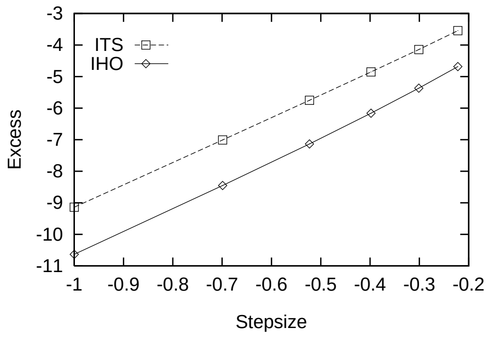
Table 8.1: ITS(7) and IHO(3,3) on $y' = -y^2$, $y(0) = 1$, $t \in [0, 12]$.

h	Excess		Excess/h^{11}		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
0.10	7.0×10^{-14}	1.1×10^{-15}	7.0×10^{-3}	1.1×10^{-4}	2.3×10^{-1}	1.9×10^{-1}
0.20	1.5×10^{-10}	5.4×10^{-13}	7.3×10^{-3}	2.6×10^{-5}	1.2×10^{-1}	9.6×10^{-2}
0.30	1.4×10^{-8}	8.2×10^{-11}	7.8×10^{-3}	4.6×10^{-5}	7.9×10^{-2}	6.4×10^{-2}
0.40	3.5×10^{-7}	3.9×10^{-9}	8.3×10^{-3}	9.3×10^{-5}	6.1×10^{-2}	4.8×10^{-2}
0.50	4.4×10^{-6}	9.2×10^{-8}	8.9×10^{-3}	1.9×10^{-4}	4.6×10^{-2}	3.8×10^{-2}
0.60	3.5×10^{-5}	1.3×10^{-6}	9.6×10^{-3}	3.6×10^{-4}	4.1×10^{-2}	3.3×10^{-2}

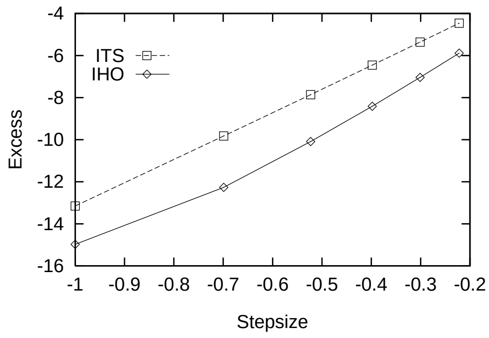
Table 8.2: ITS(11) and IHO(5,5) on $y' = -y^2$, $y(0) = 1$, $t \in [0, 12]$.

k	ch^r	
	ITS	IHO
7	$(1.04\times10^{-2}) \times h^{7.18}$	$(8.27\times10^{-4}) \times h^{7.61}$
11	$(9.94\times10^{-3})\times h^{11.17}$	$(2.41\times10^{-4})\times h^{11.76}$

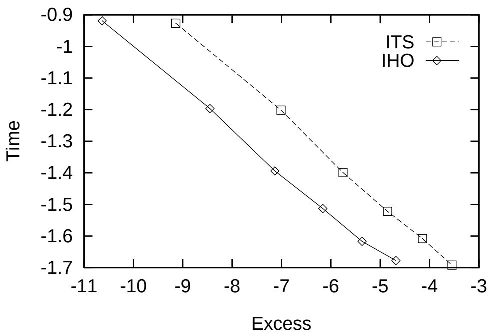
Table 8.3: Error constants and orders of the ITS and IHO methods on $y' = -y^2$, $y(0) = 1$, $t \in [0, 12]$. The excess and stepsizes used in the least squares fits are from Tables 8.1 ($k = 7$) and 8.2 ($k = 11$).



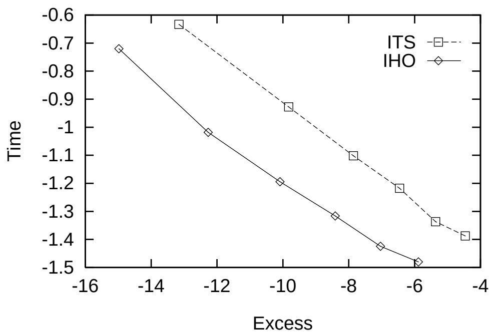
(a) ITS(7), IHO(3, 3)



(b) ITS(11), IHO(5, 5)



(c) ITS(7), IHO(3, 3)



(d) ITS(11), IHO(5, 5)

Figure 8.1: ITS and IHO on $y' = -y^2$, $y(0) = 1$, $t \in [0, 12]$.

8.2.2 Nonlinear Two-Dimensional Problem

We integrated

$$
\begin{aligned}
y_1' &= y_2 + y_1(1 - y_1^2 - y_2^2) \\
y_2' &= -y_1 + y_2(1 - y_1^2 - y_2^2)
\end{aligned} \tag{8.2.1}
$$

[66, p. 41] for

$$y(0) = (1, 1)^T, \quad t \in [0, 0.48] \tag{8.2.2}$$

with the ITS(7) and IHO(3, 3) methods. We used a constant enclosure method for validating existence and uniqueness of the solution and stepsizes $h = 0.1, 0.2, 0.4$, and 0.8. We could have computed a priori bounds from the formula for the true solution, but in this case, the constant enclosure method does not reduce the input stepsizes. Moreover, it produces tighter a priori bounds than if these bounds were computed by evaluating the formula for the true solution in interval arithmetic. Since the solution rotates in phase space, both methods use QR-factorization to reduce the wrapping effect.

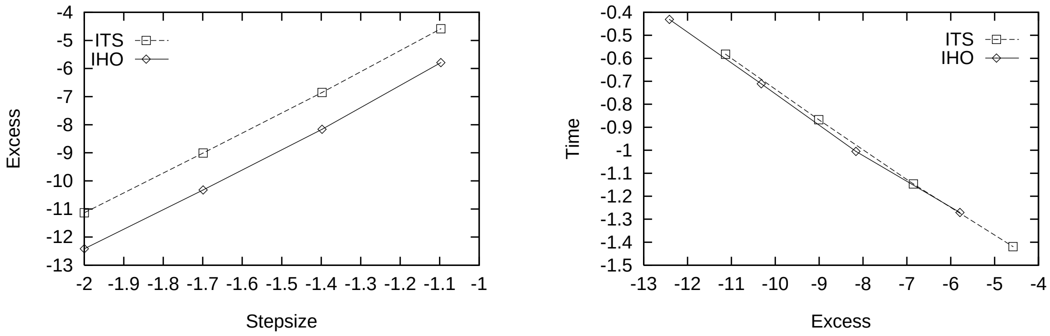
Table 8.4 shows the excess at $T = 0.48$, the excess divided by h^7, and the CPU time. In Figure 8.2, we plot (by first computing $\log 10$ of all relevant values) the excess versus the stepsize and the CPU time spent in these methods versus the excess. In Table 8.5, we have computed a least squares fit to the error constants and orders.

h	Excess		Excess/h^7		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
0.01	7.3×10^{-12}	3.9×10^{-13}	7.3×10^{2}	3.9×10^{1}	2.6×10^{-1}	3.7×10^{-1}
0.02	9.8×10^{-10}	4.8×10^{-11}	7.7×10^{2}	3.7×10^{1}	1.4×10^{-1}	1.9×10^{-1}
0.04	1.4×10^{-7}	6.9×10^{-9}	8.6×10^{2}	4.2×10^{1}	7.1×10^{-2}	9.9×10^{-2}
0.08	2.6×10^{-5}	1.6×10^{-6}	1.2×10^{3}	7.7×10^{1}	3.8×10^{-2}	5.4×10^{-2}

Table 8.4: ITS(7) and IHO(3, 3) on (8.2.1) with (8.2.2).

k	ch^r	
	ITS	IHO
7	$(2.12\times10^3)\times h^{7.25}$	$(1.44\times10^2)\times h^{7.32}$

Table 8.5: Error constant and order of the ITS and IHO methods on (8.2.1) with (8.2.2).



(a) (b)

Figure 8.2: ITS(7) and IHO(3, 3) on (8.2.1) with (8.2.2).

In this example, we have a behavior similar to the one from the previous example: slightly higher orders of both methods, than the expected $k = 7$, and slightly higher order of the IHO method than of the ITS method (see Table 8.5). In both examples, we computed tighter enclosures with the IHO method than with the ITS method, for the same stepsizes and orders of the truncations error.

For the same stepsizes, the IHO method is more expensive than the ITS method, but produces smaller excess (see Table 8.5). As a result, the IHO method is slightly less expensive for the same excess (see Figure 8.2(b)). As we shall see later, the ITS method can be less expensive for low orders if the work is measured per step.

8.3 Interval Hermite-Obreschkoff versus Interval Taylor Series Methods

8.3.1 Constant Coefficient Problems

Scalar Problem: Constant Stepsizes

We integrated

$$y' = -10y, \tag{8.3.1}$$

first with $y(0) = 1$ and then with $y(0) \in [0.9, 1.1]$ for $t \in [0, 10]$. (At $t = 10$, the true solution of (8.3.1) with $y(0) = 1$ is $e^{-100} \approx 3.7 \times 10^{-44}$.) To avoid possible stepsize reductions in Algorithm I, we computed a priori enclosures on each step by

$$[\tilde{y}_j] = [e^{-10h}\underline{y}_j, \bar{y}_j].$$

In Algorithm II, we used the ITS(17) and IHO(8,8) methods.

For constant stepsizes $0.2, 0.3, \ldots, 0.8$, Tables 8.6 and 8.7 show the excess at $T = 10$, the ratio of the excess of the IHO method to the excess of the ITS method, $\frac{\gamma_{8,8}}{Q_{8,8}(-10h)}$, and the CPU time spent in Algorithm II ($Q_{p,q}(z)$ is defined in (4.3.8)). We compute the base-10 logarithm of the data and plot in Figure 8.3 the excess versus the stepsize and the time versus the excess. We do not show the corresponding graphs for Table 8.7 since they are almost the same as in Figure 8.3.

Consider Table 8.6 and Figure 8.3(a). For "small" stepsizes, $h = 0.2, 0.3, 0.4$, the excess in the IHO method is approximately $\gamma_{8,8}/Q_{8,8}(-10h) \approx 10^{-5}$ times the excess in the ITS method, which confirms the theory in §4.3.1. As h increases beyond 0.4, the ITS method produces enclosures with rapidly increasing widths, while the IHO method computes good enclosures for those stepsizes.

h	Excess		Reductions		Time	
	ITS	IHO	IHO/ITS	$\frac{\gamma_{8,8}}{Q_{8,8}(-10h)}$	ITS	IHO
0.2	4.4×10^{-51}	1.3×10^{-55}	3.0×10^{-5}	3.0×10^{-5}	3.5×10^{-2}	2.1×10^{-2}
0.3	8.5×10^{-48}	1.6×10^{-52}	1.9×10^{-5}	1.9×10^{-5}	2.3×10^{-2}	1.4×10^{-2}
0.4	2.5×10^{-45}	2.9×10^{-50}	1.1×10^{-5}	1.2×10^{-5}	1.8×10^{-2}	1.1×10^{-2}
0.5	1.2×10^{-40}	1.8×10^{-48}	1.5×10^{-8}	7.8×10^{-6}	1.3×10^{-2}	8.8×10^{-3}
0.6	8.5×10^{-20}	5.9×10^{-47}	6.9×10^{-28}	5.2×10^{-6}	1.1×10^{-2}	7.0×10^{-3}
0.7	4.0×10^{-1}	1.3×10^{-45}	3.2×10^{-45}	3.5×10^{-6}	1.0×10^{-2}	6.3×10^{-3}
0.8	2.4×10^{10}	2.0×10^{-44}	8.4×10^{-55}	2.4×10^{-6}	8.9×10^{-3}	5.4×10^{-3}

Table 8.6: ITS(17) and IHO(8, 8) on $y' = -10y$, $y(0) = 1$, $t \in [0, 10]$.

h	Excess		Reductions		Time	
	ITS	IHO	IHO/ITS	$\frac{\gamma_{8,8}}{Q_{8,8}(-10h)}$	ITS	IHO
0.2	4.4×10^{-51}	1.2×10^{-55}	2.6×10^{-5}	3.0×10^{-5}	3.4×10^{-2}	2.1×10^{-2}
0.3	8.0×10^{-48}	1.5×10^{-52}	1.8×10^{-5}	1.9×10^{-5}	2.3×10^{-2}	1.4×10^{-2}
0.4	2.2×10^{-45}	2.8×10^{-50}	1.3×10^{-5}	1.2×10^{-5}	1.9×10^{-2}	1.2×10^{-2}
0.5	7.9×10^{-41}	1.9×10^{-48}	2.4×10^{-8}	7.8×10^{-6}	1.4×10^{-2}	8.3×10^{-3}
0.6	5.4×10^{-20}	6.2×10^{-47}	1.1×10^{-27}	5.2×10^{-6}	1.1×10^{-2}	7.0×10^{-3}
0.7	2.5×10^{-1}	1.4×10^{-45}	5.5×10^{-45}	3.5×10^{-6}	1.0×10^{-2}	6.3×10^{-3}
0.8	1.5×10^{10}	2.2×10^{-44}	1.5×10^{-54}	2.4×10^{-6}	8.7×10^{-3}	5.4×10^{-3}

Table 8.7: ITS(17) and IHO(8, 8) on $y' = -10y$, $y(0) \in [0.9, 1.1]$, $t \in [0, 10]$.

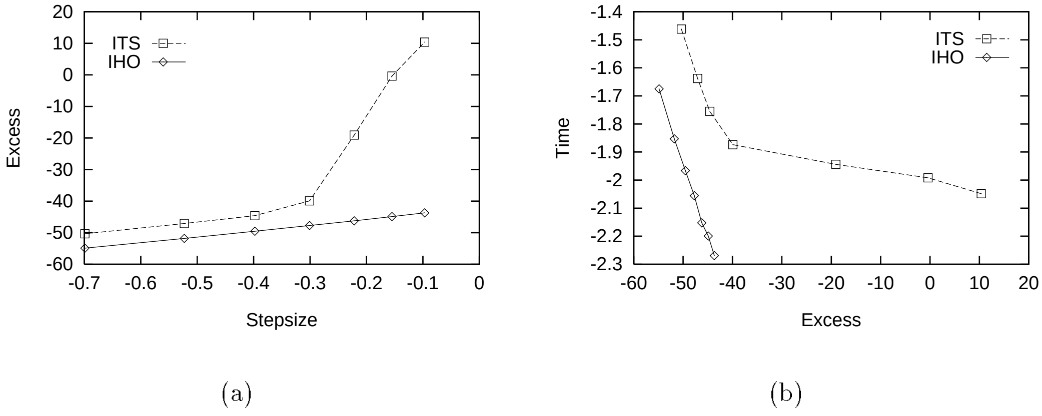


Figure 8.3: ITS(17) and IHO(8, 8) on $y' = -10y$, $y(0) = 1$, $t \in [0, 10]$.

Variable Stepsizes

We integrated (8.3.1) with $y(0) = 1$ for $t \in [0, 100]$ with the stepsize selection scheme from §6.2. We used an absolute tolerance of 10^{-10}. In Figure 8.4, we plot the stepsizes against the step number for the two methods. With the ITS method, the solver takes a small stepsize in the last step to hit the endpoint exactly.

The ITS method is asymptotically unstable for stepsizes h such that

$$|T_{16}(-10h)| + \frac{(10h)^{17}}{17!} > 1$$

(see §4.3.1). For $h = 0.695$,

$$|T_{16}(-10h)| + \frac{(10h)^{17}}{17!} \approx 0.996.$$

For the IHO method, the stepsize oscillates around 1.875, which is about 2.7 times bigger than 0.695, the stepsize limit for the ITS method. For $h = 1.875$,

$$|R_{8,8}(-10h)| + \frac{\gamma_{8,8}}{|Q_{8,8}(-10h)|}\frac{(10h)^{17}}{17!} \approx 0.996.$$

Although the IHO method permits larger stepsizes, they are still limited by its local error term. This observation confirms the theory in §4.3.1.

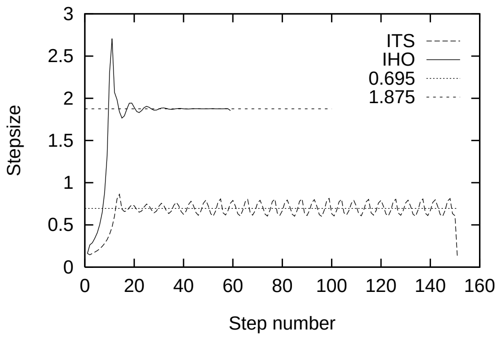


Figure 8.4: ITS(17) and IHO(8, 8) on $y' = -10y$, $y(0) = 1$, $t \in [0, 100]$, variable stepsize control with $Tol = 10^{-10}$.

Two-Dimensional Problem

We compare the ITS(17) and IHO(8, 8) methods on the system

$$y' = By = \begin{pmatrix} 1 & -2 \\ 3 & -4 \end{pmatrix} y \tag{8.3.2}$$

[44]. This system is interesting because the solution components tend to zero rapidly, but still, we have to deal with the wrapping effect. For example, if

$$y(0) = (1, -1)^T, \tag{8.3.3}$$

then the true solution of (8.3.2–8.3.3) is given by

$$y_1(t) = 5e^{-t} - 4e^{-2t}$$

$$y_2(t) = 5e^{-t} - 6e^{-2t}.$$

As t increases, both $y_1(t)$ and $y_2(t)$ become approximately $5e^{-t}$. In the phase plane, the solution becomes almost parallel to the line $y_2 = y_1$. If the solution is enclosed by a parallelepiped, as in the direct method (see §3.2.2), there is a large overestimation, which increases with the steps taken.

Constant Stepsizes We integrated (8.3.2) on $[0, 50]$ first with an initial condition $y(0) = (1, -1)^T$ and then with an initial condition $y(0) \in ([0.9, 1.1], [-0.1, 0.1])^T$. We used constant stepsizes $h = 1.2, 1.4, \ldots, 3.4$ and computed a priori enclosures of the solution on each step by

$$[\tilde{y}_j] = \begin{pmatrix} 3e^{-[0,h]} - 2e^{-2[0,h]} & -2e^{-[0,h]} + 2e^{-2[0,h]} \\ 3e^{-[0,h]} - 3e^{-2[0,h]} & -2e^{-[0,h]} + 3e^{-2[0,h]} \end{pmatrix} [y_j], \tag{8.3.4}$$

which is obtained from the true solution

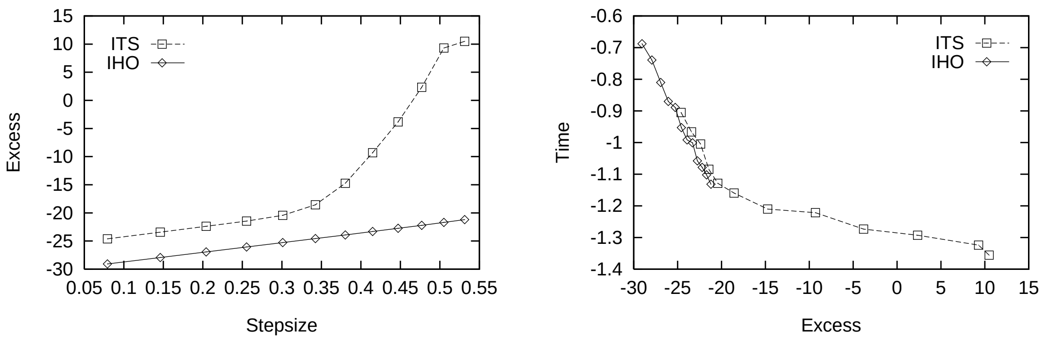
$$y(t) = \begin{pmatrix} e^{-t} & 2e^{-2t} \\ e^{-t} & 3e^{-2t} \end{pmatrix} \begin{pmatrix} 3 & -2 \\ -1 & 1 \end{pmatrix} y(0).$$

The results are shown in Tables 8.8 and 8.9. Corresponding to Table 8.8, we compute the base-10 logarithm of the data and plot in Figure 8.5 the excess versus the stepsize and the CPU time spent in Algorithm II versus the excess. Since the results for Table 8.9 are similar, we do not show the corresponding graphs.

In the ITS method, the widths of the computed enclosures increase rapidly with h for $h \geq 2.8$. We would expect a blow up to occur for stepsizes not much smaller than 3.66, which is determined from the condition $|T_{16}(-2h)| < 1$ (The eigenvalues of B are -1 and -2.). The reason for this "early" blow up is that the a priori enclosures are not tight enough, and as a result, the local excess in the ITS method is not as small as it should be. However, the IHO method produces good enclosures for all stepsizes considered ($h = 1.2, 1.4, \ldots, 3.4$). From the fourth column in Table 8.8, the excess in the IHO method is (roughly) at least 10^{-5} times the excess in the ITS method (see also §4.3.2, (4.3.25)).

h	Excess		Reductions		Time	
	ITS	IHO	IHO/ITS	$Q(hB)$	ITS	IHO
1.2	2.5×10^{-25}	8.6×10^{-30}	3.5×10^{-5}	1.2×10^{-4}	1.2×10^{-1}	2.1×10^{-1}
1.4	3.7×10^{-24}	1.2×10^{-28}	3.1×10^{-5}	1.1×10^{-4}	1.1×10^{-1}	1.8×10^{-1}
1.6	4.1×10^{-23}	1.2×10^{-27}	2.8×10^{-5}	1.1×10^{-4}	9.9×10^{-2}	1.5×10^{-1}
1.8	3.6×10^{-22}	8.6×10^{-27}	2.4×10^{-5}	1.1×10^{-4}	8.2×10^{-2}	1.3×10^{-1}
2.0	3.7×10^{-21}	5.3×10^{-26}	1.4×10^{-5}	1.0×10^{-4}	7.4×10^{-2}	1.3×10^{-1}
2.2	2.7×10^{-19}	2.7×10^{-25}	1.0×10^{-6}	9.4×10^{-5}	6.9×10^{-2}	1.1×10^{-1}
2.4	1.8×10^{-15}	1.2×10^{-24}	6.7×10^{-10}	8.9×10^{-5}	6.2×10^{-2}	1.0×10^{-1}
2.6	5.0×10^{-10}	5.0×10^{-24}	1.0×10^{-14}	8.3×10^{-5}	6.0×10^{-2}	1.0×10^{-1}
2.8	1.5×10^{-4}	1.8×10^{-23}	1.2×10^{-19}	7.8×10^{-5}	5.3×10^{-2}	8.8×10^{-2}
3.0	2.2×10^{2}	6.2×10^{-23}	2.9×10^{-25}	7.3×10^{-5}	5.1×10^{-2}	8.3×10^{-2}
3.2	2.0×10^{9}	2.0×10^{-22}	1.0×10^{-31}	6.8×10^{-5}	4.7×10^{-2}	7.9×10^{-2}
3.4	3.0×10^{10}	6.4×10^{-22}	2.1×10^{-32}	6.3×10^{-5}	4.4×10^{-2}	7.4×10^{-2}

Table 8.8: ITS(17) and IHO(8, 8) on (8.3.2), $y(0) = (1, -1)^T$, $t \in [0, 50]$,
$Q(hB) = \gamma_{8,8} \| \big(Q_{8,8}(hB)\big)^{-1} \|$.



(a) (b)

Figure 8.5: ITS(17) and IHO(8, 8) on (8.3.2), $y(0) = (1, -1)^T$, $t \in [0, 50]$.

h	Excess		Reductions		Time	
	ITS	IHO	IHO/ITS	$Q(hB)$	ITS	IHO
1.2	1.4×10^{-25}	4.7×10^{-30}	3.5×10^{-5}	1.2×10^{-4}	1.2×10^{-1}	2.0×10^{-1}
1.4	2.1×10^{-24}	6.5×10^{-29}	3.1×10^{-5}	1.1×10^{-4}	1.1×10^{-1}	1.9×10^{-1}
1.6	2.2×10^{-23}	6.4×10^{-28}	2.8×10^{-5}	1.1×10^{-4}	9.5×10^{-2}	1.5×10^{-1}
1.8	2.0×10^{-22}	4.7×10^{-27}	2.4×10^{-5}	1.1×10^{-4}	8.3×10^{-2}	1.3×10^{-1}
2.0	2.0×10^{-21}	2.9×10^{-26}	1.4×10^{-5}	1.0×10^{-4}	7.6×10^{-2}	1.3×10^{-1}
2.2	1.5×10^{-19}	1.5×10^{-25}	1.0×10^{-6}	9.4×10^{-5}	6.9×10^{-2}	1.1×10^{-1}
2.4	1.0×10^{-15}	6.7×10^{-25}	6.7×10^{-10}	8.9×10^{-5}	6.3×10^{-2}	1.0×10^{-1}
2.6	2.7×10^{-10}	2.8×10^{-24}	1.0×10^{-14}	8.3×10^{-5}	6.0×10^{-2}	9.9×10^{-2}
2.8	8.3×10^{-5}	9.9×10^{-24}	1.2×10^{-19}	7.8×10^{-5}	5.3×10^{-2}	8.6×10^{-2}
3.0	1.2×10^{2}	3.4×10^{-23}	2.9×10^{-25}	7.3×10^{-5}	5.0×10^{-2}	8.1×10^{-2}
3.2	1.1×10^{9}	1.1×10^{-22}	1.0×10^{-31}	6.8×10^{-5}	4.8×10^{-2}	7.8×10^{-2}
3.4	1.7×10^{10}	3.5×10^{-22}	2.1×10^{-32}	6.3×10^{-5}	4.5×10^{-2}	7.2×10^{-2}

Table 8.9: ITS(17) and IHO(8, 8) on (8.3.2), $y(0) \in ([0.9, 1.1], [-0.1, 0.1])^T$, $t \in [0, 50]$, $Q(hB) = \gamma_{8,8} \| \big(Q_{8,8}(hB)\big)^{-1} \|$.

Variable Stepsizes We integrated (8.3.2) with $y(0) = (1, -1)^T$ for $t \in [0, 400]$ with a variable step control with $Tol = 10^{-11}$. In Figures 8.6(a-d), we show how the stepsize depends on how the a priori bounds are computed and whether they are intersected with the tight bounds.

If the a priori enclosures are computed from (8.3.4), then these enclosures are normally wider than the tight bounds. Since the intersection of the a priori and tight enclosures produces intervals that are the same (or almost the same) as the tight bounds, the stepsize shows similar behavior, whether or not these bounds are intersected; see Figures 8.6(a-b). Because of the additional stability restriction from the remainder term, the stepsize in the ITS cannot reach the value 3.66 and oscillates around 2.9. Recall that 3.66 is determined from $|T_{k-1}(-2h)| < 1$. The stepsize in the IHO method is restricted mainly by the associated formula for the truncation error. If the a priori bounds are computed by

$$[\tilde{y}_j] = \begin{cases} \begin{pmatrix} 3e^{-[0,h]} - 2e^{-2[0,h]} & -2e^{-[0,h]} + 2e^{-2[0,h]} \\ 3e^{-[0,h]} - 3e^{-2[0,h]} & -2e^{-[0,h]} + 3e^{-2[0,h]} \end{pmatrix} [y_j], & \text{if } t \le 1 \\ [0, \bar{y}_j] & \text{otherwise}, \end{cases} \tag{8.3.5}$$

we observe a different behavior; see Figures 8.6(c-d). With (8.3.5), we compute tighter a priori bounds for $t > 1$ than with (8.3.4).

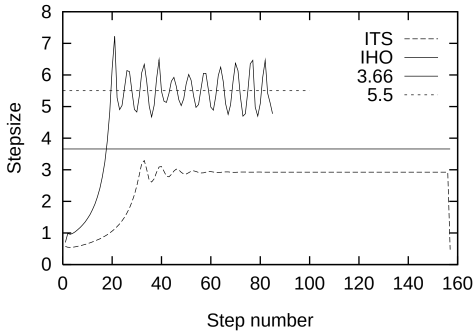
In Figure 8.6(c), the stepsize in the ITS method oscillates slightly below 3.66. In a standard method with a stability restriction on the stepsize of 3.66, we would expect these oscillations to occur at about 3.66, but here, they are shifted down because of the restriction on the stepsize from the remainder term. In this figure, the oscillations of the stepsizes in the ITS method and the IHO method occur at larger values of h than in Figure 8.6(a). The reason is that we compute tighter bounds for the truncation error.

In Figure 8.6(d), the stepsize in the ITS method reaches a value greater than 3.66 and then stays at this value. Taking stepsizes outside the stability region of $T_{k-1}(hB)$ seems strange, but this phenomenon can be explained as follows.

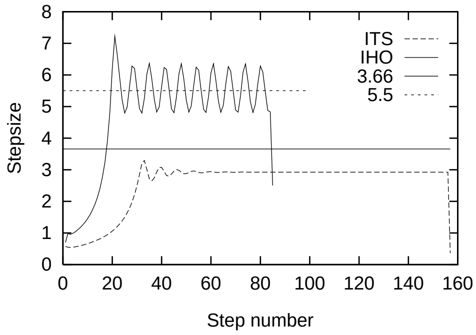
As the stepsize increases towards 3.66, the ITS method becomes unstable for some $h_j < 3.66$. Suppose that the solver has accepted $[y_j]$ at $t_j > 1$ and computes $[y_{j+1}]$ with the ITS method and such h_j. Because of instability, $w([y_{j+1}]) > w([y_j])$. Since the true solution components tend to zero as t increases, we can assume that $[y_{j+1}]$ contains $(0,0)^T$. Then the tight and a priori bounds are intersected, and the solver accepts

$$[y^1_{j+1}] = [y_{j+1}] \cap [\tilde{y}_j] = [y_{j+1}] \cap [0, \bar{y}_j] = [0, \bar{y}_j].$$

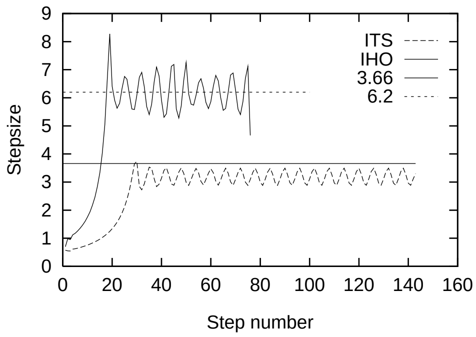
For the next step, it determines a stepsize so that $[0, \bar{y}_j]$ satisfies the tolerance requirement. In our example, such a stepsize is greater than 3.66. The ITS produces again a tight bound that is wider than the a priori one, which is again $[0, \bar{y}_j]$. Thus, the solver



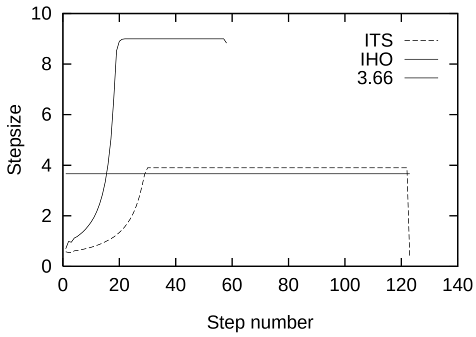
(a) Tight and a priori enclosures are not intersected; a priori enclosures computed from (8.3.4).



(b) Tight and a priori enclosures are intersected; a priori enclosures computed from (8.3.4).



(c) Tight and a priori enclosures are not intersected; a priori enclosures computed from (8.3.5).



(d) Tight and a priori enclosures are intersected; a priori enclosures computed from (8.3.5).

Figure 8.6: ITS(17) and IHO(8, 8) on (8.3.2), $y(0) = (-1, 1)$, $t \in [0, 400]$, variable stepsize control with $Tol = 10^{-11}$.

keeps taking the same stepsize and accepting $[0, \bar{y}_j]$, which satisfies the accuracy requirement. The situation is similar with the IHO method, except that the stepsize reaches a much larger value and stays at it.

It is important to note on this example that although Algorithm II becomes unstable, the integration essentially continues with Algorithm I. Here, we knew how to compute good a priori bounds in Algorithm I for large stepsizes, but this is rarely the case.

8.3.2 Nonlinear Problems

Example 1 We integrated [39]

$$y' = t(1-y) + (1-t)e^{-t}, \tag{8.3.6}$$

with constant stepsizes $0.2, 0.3, 0.4$, and 0.5. In autonomous form, this equation is

$$\begin{aligned} y_1' &= 1 \\ y_2' &= y_1(1-y_2) + (1-y_1)e^{-y_1}. \end{aligned}$$

We computed a priori enclosures from

$$\begin{aligned} [\tilde{y}_{j,1}] &= [y_{j,1}] + [0, h] \\ [\tilde{y}_{j,2}] &= 1 - e^{-[\tilde{y}_{j,1}]} + e^{-[\tilde{y}_{j,1}^2]/2} \frac{[y_{j,2}] - 1 + e^{-t_j}}{e^{-t_j^2/2}}, \end{aligned}$$

which is determined from the true solution

$$y(t) = 1 - e^{-t} + \frac{e^{-t^2/2}(y_j - 1 + e^{-t_j})}{e^{-t_j^2/2}}, \quad \text{where } y(t_j) = y_j,$$

and used the ITS(17) and IHO(8, 8) methods with the direct method, described in §3.2.2 (without the QR-factorization, described in §3.2.5).

As can be seen from the results in Tables 8.10 and 8.11 and Figure 8.7, for the same stepsizes, the IHO method produces much better enclosures in less time than the ITS method. In these tables, we have also shown the maximum excess during the integration.

In Table 8.12, we show results produced with the ITS(17) and IHO(8, 8) methods with the QR-factorization and without rearranging the columns of the transformation matrix (see §3.2.5). It is interesting to note that in this case, the solver computed wider bounds than the ones reported in Table 8.11, which are obtained without QR-factorization. The reason is that by computing the interval vectors $[r_j]$, $j \geq 1$ (see §3.2.5 and §4.1.3), the initial excess in the second component of the solution is introduced into the first one. Then, the excess in the first component propagates, as we integrate towards the endpoint.

h	Excess		Max Excess		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
0.2	5.3×10^{-14}	4.4×10^{-16}	6.5×10^{-14}	2.2×10^{-15}	1.4×10^{0}	1.1×10^{0}
0.3	1.2×10^{-12}	3.3×10^{-16}	7.9×10^{-11}	1.8×10^{-15}	9.2×10^{-1}	7.6×10^{-1}
0.4	4.5×10^{-8}	4.5×10^{-14}	4.5×10^{-8}	1.1×10^{-13}	7.1×10^{-1}	5.7×10^{-1}
0.5	2.1×10^{-6}	1.3×10^{-12}	2.1×10^{-6}	3.5×10^{-12}	5.5×10^{-1}	4.5×10^{-1}

Table 8.10: ITS(17) and IHO(8, 8) on $y' = t(1 - y) + (1 - t)e^{-t}$, $y(0) = 1$, $t \in [0, 20]$.

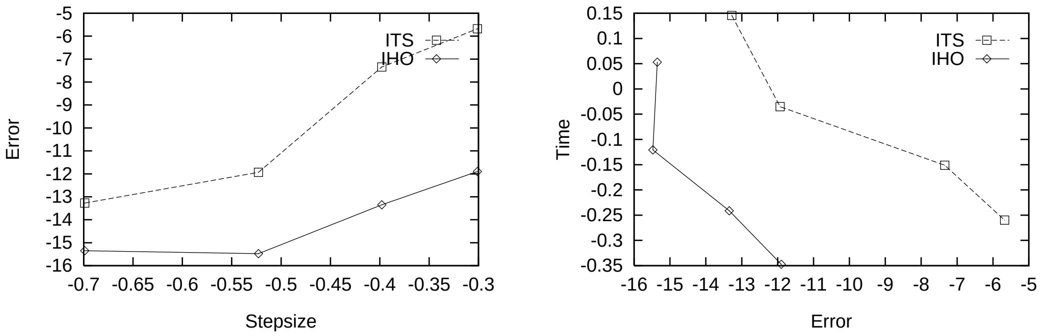


Figure 8.7: ITS(17) and IHO(8, 8) on $y' = t(1 - y) + (1 - t)e^{-t}$, $y(0) = 1$, $t \in [0, 20]$.

h	Excess		Max Excess		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
0.2	3.3×10^{-14}	1.1×10^{-16}	5.0×10^{-14}	1.6×10^{-15}	1.4×10^{0}	1.1×10^{0}
0.3	1.0×10^{-12}	1.1×10^{-16}	7.4×10^{-11}	1.1×10^{-15}	9.3×10^{-1}	7.5×10^{-1}
0.4	4.5×10^{-8}	2.2×10^{-14}	4.5×10^{-8}	7.6×10^{-14}	7.1×10^{-1}	5.7×10^{-1}
0.5	2.1×10^{-6}	6.3×10^{-13}	2.1×10^{-6}	2.6×10^{-12}	5.5×10^{-1}	4.5×10^{-1}

Table 8.11: ITS(17) and IHO(8,8) on $y' = t(1-y) + (1-t)e^{-t}$, $y(0) \in [0.999, 1.001]$, $t \in [0, 20]$.

h	Excess		Max Excess		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
0.2	8.3×10^{-4}	8.3×10^{-4}	8.3×10^{-4}	8.3×10^{-4}	1.4×10^{0}	1.2×10^{0}
0.3	8.1×10^{-4}	8.1×10^{-4}	8.1×10^{-4}	8.1×10^{-4}	9.4×10^{-1}	8.1×10^{-1}
0.4	1.6×10^{-3}	8.3×10^{-4}	1.6×10^{-3}	8.3×10^{-4}	7.2×10^{-1}	6.1×10^{-1}
0.5	$1.0 \times 10^{+7}$	8.4×10^{-4}	$1.0 \times 10^{+7}$	8.4×10^{-4}	5.6×10^{-1}	4.8×10^{-1}

Table 8.12: ITS(17) and IHO(8,8) on $y' = t(1-y) + (1-t)e^{-t}$, $y(0) \in [0.999, 1.001]$, $t \in [0, 20]$, QR-factorization.

Example 2 Two-Body Problem

We integrated the two-body problem

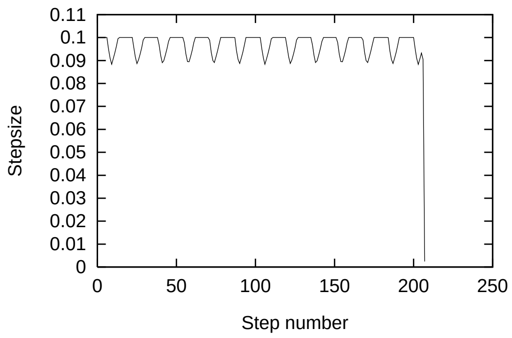
$$
\begin{aligned}
y_1' &= y_3 \\
y_2' &= y_4 \\
y_3' &= -\frac{y_1}{(y_1^2 + y_2^2)^{3/2}} \\
y_4' &= -\frac{y_2}{(y_1^2 + y_2^2)^{3/2}}, \\
y(0) &= (1, 0, 0, 1)^T, \quad t \in [0, 20]
\end{aligned}
$$

with the ITS(17) and IHO(8, 8) methods. We used a constant enclosure method in Algorithm I and input stepsizes 0.1, 0.15, 0.2 to this method. For input stepsize 0.1 to the validation procedure, the IHO method produces slightly better enclosures for slightly less work, Table 8.13. However, when the stepsize is 0.15 or 0.2, the excess of the IHO method is significantly smaller than the excess of the ITS method.

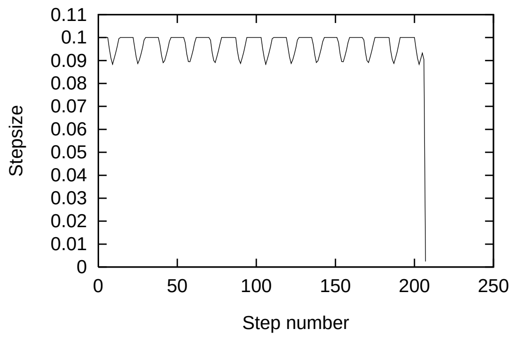
H	Steps		Excess		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
0.10	207	207	5.2×10^{-10}	7.3×10^{-11}	6.1	4.7
0.15	154	154	5.4×10^{-7}	2.0×10^{-10}	4.6	3.7
0.20	161	161	6.6×10^{-6}	3.5×10^{-9}	4.9	3.7

Table 8.13: ITS(17) and IHO(8, 8) on the two-body problem, constant enclosure method.

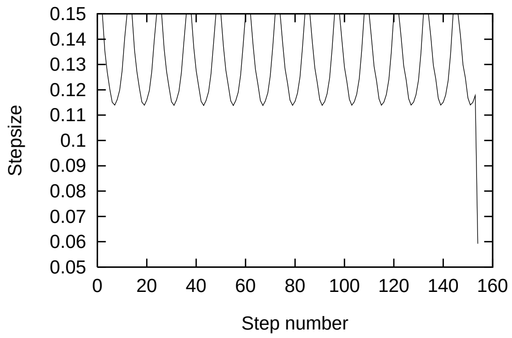
Since Algorithm I usually reduces the input stepsize, in Figure 8.8, we plot the stepsize against the step number. The stepsizes reductions at the end of the plots occurs because the solver takes a small stepsize to hit the endpoint (in time) exactly.



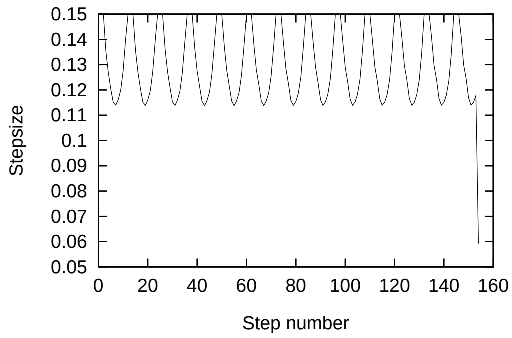
(a) Input stepsize 0.1, ITS



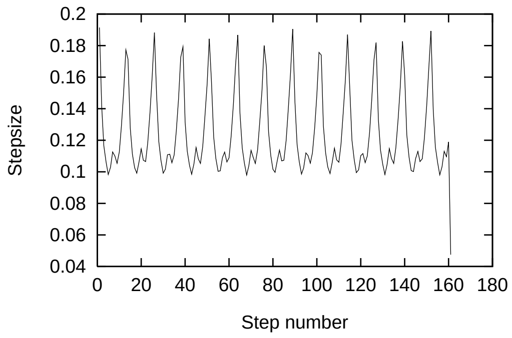
(b) Input stepsize 0.1, IHO



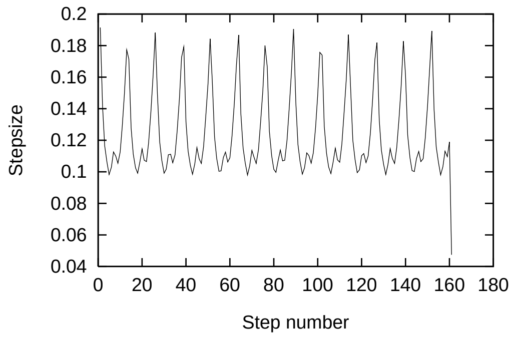
(c) Input stepsize 0.15, ITS



(d) Input stepsize 0.15, IHO



(e) Input stepsize 0.2, ITS



(f) Input stepsize 0.2, IHO

Figure 8.8: ITS(17) and IHO(8, 8) on the two-body problem, constant enclosure method.

Example 3 Lorenz system

We integrated

$$
\begin{aligned}
y_1' &= \sigma(y_2 - y_1) \\
y_2' &= y_1(\rho - y_3) - y_2 \\
y_3' &= y_1 y_2 - \beta y_3, \\
y(0) &= (15, 15, 36)^T, \qquad t \in [0, 10],
\end{aligned}
$$

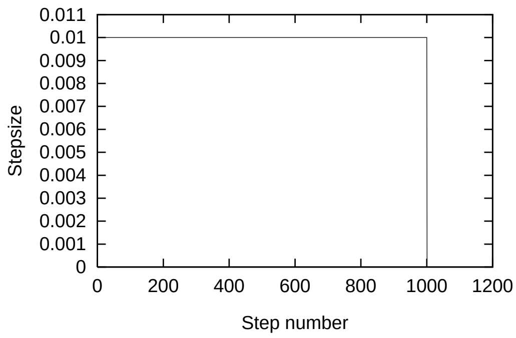
where $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$, with the ITS(17) and IHO$(8,8)$ methods and used a constant enclosure method in Algorithm I. The input stepsizes for Algorithm I are $0.01, 0.05, 0.1$. The results are shown in Table 8.14, and the stepsizes versus step number are shown in Figure 8.9. As in the two-body problem, the IHO method produces tighter enclosures in less time, than the ITS method.

H	Steps		Excess		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
0.01	1001	1001	1.0×10^{-6}	7.6×10^{-7}	1.4×10^{1}	1.2×10^{1}
0.05	286	286	2.0×10^{-1}	1.6×10^{-2}	4.0×10^{0}	3.4×10^{0}
0.10	282	282	2.9×10^{-1}	2.4×10^{-2}	4.0×10^{0}	3.4×10^{0}

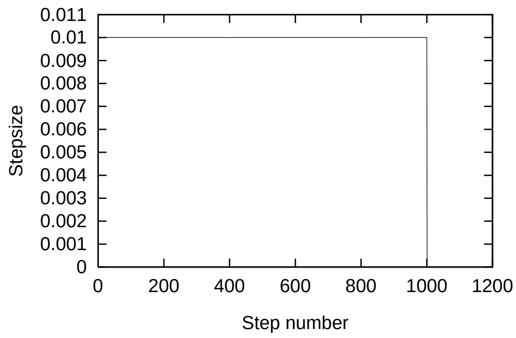
Table 8.14: ITS(17) and IHO$(8,8)$ on the Lorenz system, constant enclosure method.

We also tried the IHO method with the a priori bounds from Algorithm I as an input to the corrector, instead of computing bounds with the predictor from §4.2.2. With $H = 0.01$ and $T = 0.8$, the excess at $T = 0.8$ was 26.8. Therefore, if we want to eliminate the predictor step, we have to perform at least one more step of the corrector, which is more expensive than the predictor.

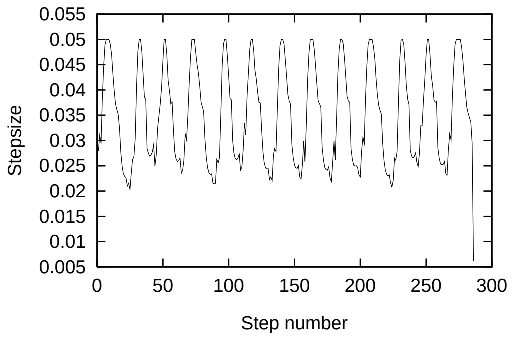
In the next two examples, we compare the ITS and IHO methods with a variable stepsize control (see §6.2) and our version of a Taylor series method for validating existence and uniqueness of the solution (see Chapter 5).



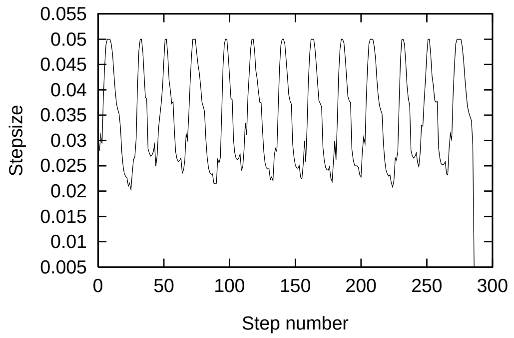
(a) Input stepsize 0.01, ITS



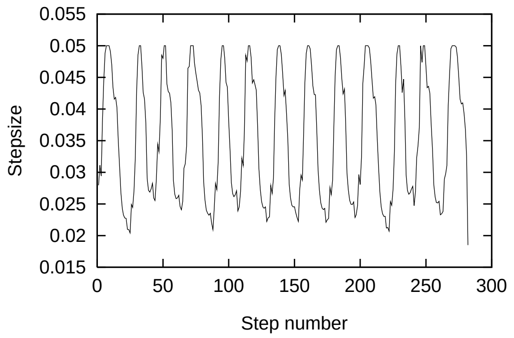
(b) Input stepsize 0.01, IHO



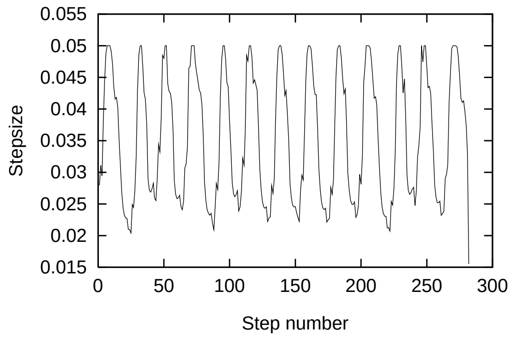
(c) Input stepsize 0.05, ITS



(d) Input stepsize 0.05, IHO



(e) Input stepsize 0.1, ITS



(f) Input stepsize 0.1, IHO

Figure 8.9: ITS(17) and IHO(8, 8) on the Lorenz system, constant enclosure method.

Example 4 Van der Pol's equation

We integrated Van der Pol's equation, written as a system,

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \mu(1-y_1^2)y_2 - y_1, \end{aligned} \tag{8.3.7}$$

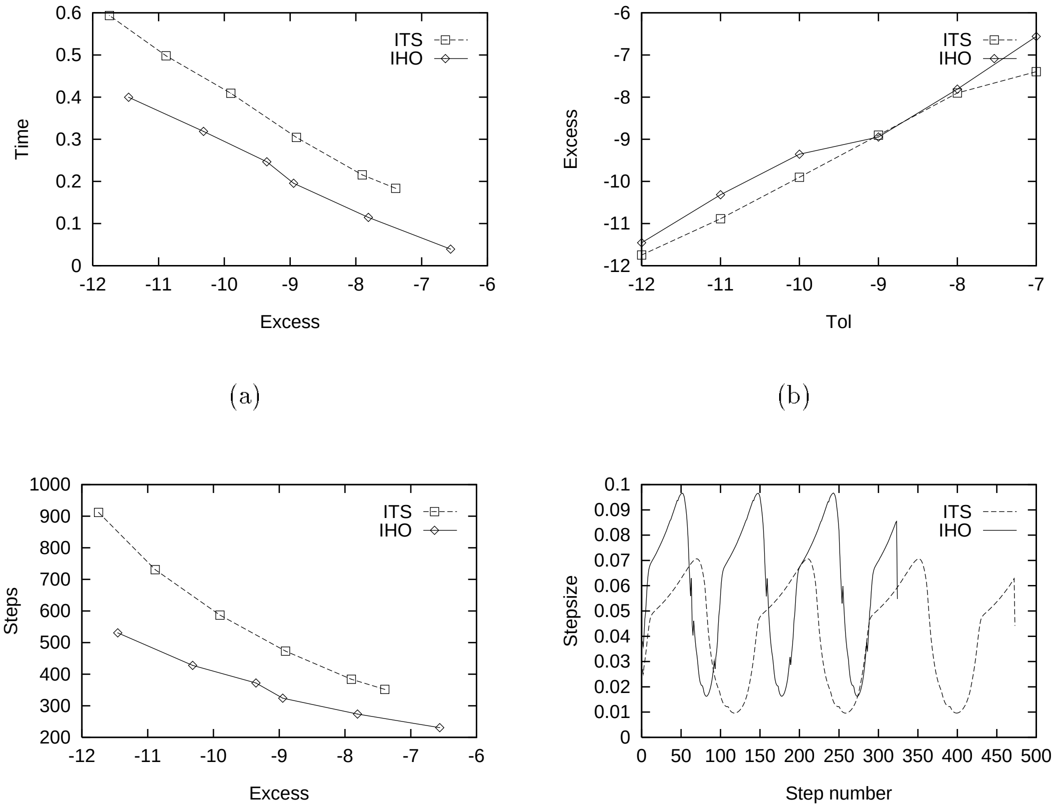
with

$$y(0) = (2,0)^T, \tag{8.3.8}$$

for $t \in [0,20]$, where $\mu = 5$. We used the ITS(11) and IHO(5, 5) methods and tolerances $10^{-7}, 10^{-8}, \ldots, 10^{-12}$.

From Table 8.15 and Figure 8.10, we see that, for approximately the same excess, VNODE using the IHO method took fewer steps than it did using the ITS method, thus saving computation time. In Figure 8.10, we plot the logarithms of the excess, time, and tolerance. In Figure 8.10(d), the stepsize corresponding to the IHO method is not as smooth as the one corresponding to the ITS method. In the regions where the stepsize is not smooth, the Taylor series method for validation could not verify existence and uniqueness with the supplied stepsizes, but verified with reduced stepsizes. Note also that we control the local excess per unit step and report the global excess in Table 8.15. Thus the global excess can be larger than the tolerance.

We also integrated (8.3.7–8.3.8) on $[0, 0.1]$ with an input stepsize of 0.01 to Algorithm I. We used orders $k = 3, 7, 11, 17, 25, 31, 37, 43$, and 49 for the ITS method and $p = q = (k-1)/2$ for the IHO method. Algorithm I did not reduce the input stepsize. As a result, the solver could take the same number of steps with the ITS and IHO methods. In Figure 8.11, we plot the logarithm of the CPU time against the logarithm of the order for these two methods. Although on this problem, the IHO method is more expensive for "low" orders, including $k = 11$, we still have savings in time (for the same excess) due to the fewer steps taken.



(a) (b)

(c) (d) $Tol = 10^{-9}$

Figure 8.10: ITS(11) and IHO(5,5) on Van der Pol's equation, Taylor series for validation, variable stepsize control with $Tol = 10^{-7}, 10^{-8}, \ldots, 10^{-12}$.

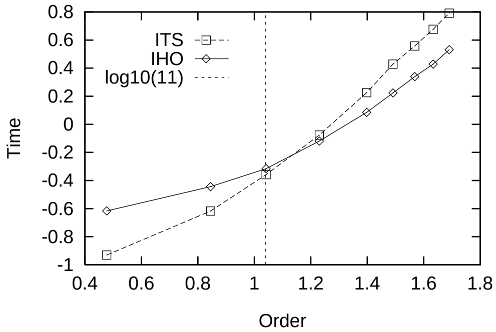


Figure 8.11: ITS and IHO with orders $3, 7, 11, 17, 25, 31, 37, 43$, and 49 on Van der Pol's equation.

Tol	Excess		Steps		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
10^{-7}	4.0×10^{-8}	2.8×10^{-7}	352	231	1.5	1.1
10^{-8}	1.3×10^{-8}	1.6×10^{-8}	384	274	1.6	1.3
10^{-9}	1.2×10^{-9}	1.1×10^{-9}	473	324	2.0	1.6
10^{-10}	1.3×10^{-10}	4.4×10^{-10}	587	372	2.6	1.8
10^{-11}	1.3×10^{-11}	4.8×10^{-11}	731	428	3.1	2.1
10^{-12}	1.8×10^{-12}	3.5×10^{-12}	912	531	3.9	2.5

Table 8.15: ITS(11) and IHO(5, 5) on Van der Pol's equation, Taylor series for validation, variable stepsize control.

Example 5 Stiff DETEST Problem D1

We integrated the Stiff DETEST problem D1 [21],

$$\begin{aligned} y_1' &= 0.2(y_2 - y_1) \\ y_2' &= 10y_1 - (60 - 0.125y_3)y_2 + 0.125y_3 \\ y_3' &= 1, \end{aligned} \tag{8.3.9}$$

with

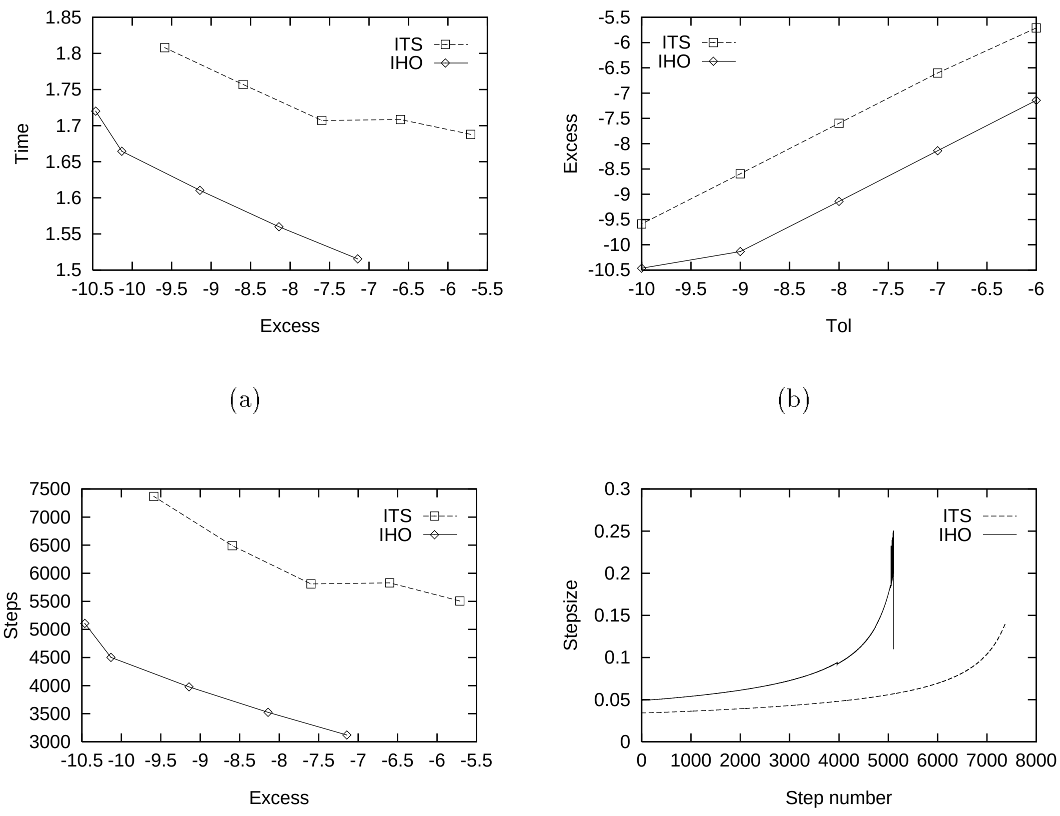
$$y(0) = (0,0,0)^T, \quad \text{for} \quad t \in [0, 400]. \tag{8.3.10}$$

Here, we used the ITS(17) and IHO(8, 8) methods, Taylor series for validation, and a variable stepsize control with tolerances $10^{-6}, 10^{-7}, \ldots, 10^{-10}$.

With the IHO method, we computed tighter bounds with fewer stepsizes, than with the ITS method; see Table 8.16 and Figure 8.12. The reduction in the stepsize on the last step for the IHO method seen in Figure 8.12(d) is a result of our program reducing the stepsize to hit the endpoint exactly.

Tol	Excess		Steps		Time	
	ITS	IHO	ITS	IHO	ITS	IHO
10^{-6}	1.9×10^{-6}	7.2×10^{-8}	5506	3122	4.9×10^{1}	3.3×10^{1}
10^{-7}	2.5×10^{-7}	7.2×10^{-9}	5829	3524	5.1×10^{1}	3.6×10^{1}
10^{-8}	2.5×10^{-8}	7.2×10^{-10}	5811	3977	5.1×10^{1}	4.1×10^{1}
10^{-9}	2.5×10^{-9}	7.4×10^{-11}	6492	4502	5.7×10^{1}	4.6×10^{1}
10^{-10}	2.6×10^{-10}	3.4×10^{-11}	7367	5107	6.4×10^{1}	5.2×10^{1}

Table 8.16: ITS(17) and IHO(8,8) on Stiff DETEST D1, Taylor series for validation, variable stepsize control.



(a) (b)

(c) (d) $Tol = 10^{-10}$

Figure 8.12: ITS(17) and IHO(8,8) on Stiff DETEST D1, Taylor series for validation, variable stepsize control with $Tol = 10^{-6}, 10^{-7}, \ldots, 10^{-10}$.

8.4 Taylor Series versus Constant Enclosure Method

We integrated the following problems, which we denote by P1, P2, P3, and P4,

P1: (8.3.6) with $y(0) = 1$, for $t \in [0, 20]$;

P2: $y_1' = y_2$, $y_2' = -y_1$, with $y(0) = (0, 1)^T$, for $t \in [0, 100]$;

P3: (8.3.2) with $y(0) = (1, -1)^T$, for $t \in [0, 50]$; and

P4: (8.3.9) with $y(0) = (0, 0, 0)^T$, for $t \in [0, 50]$.

For all of these tests, we used order $k = 17$ for the ITS method and $p = q = 8$ for the IHO method, and LEPUS error control with $Tol = 10^{-10}$.

Tables 8.17 and 8.18 show the number of steps taken by VNODE, when Algorithm I uses a constant enclosure (CE) method (see §3.1) and our Taylor series enclosure (TSE) method (see Chapter 5), and the corresponding excess and times. The results in Table 8.17 are produced with the ITS method, and the results in Table 8.18 are produced with the IHO method. In Figures 8.13 and 8.14, we plot the stepsizes against the step number.

From Table 8.17, we see that if we use a Taylor series enclosure method, we have a significant reduction in the number of steps (with $Tol = 10^{-10}$). Furthermore, from the obtained excess, we see that with a Taylor series enclosure method the stepsize is controlled from the accuracy requirements. In the constant enclosure method, we achieve more accuracy than we have asked for, implying that the stepsize was controlled from Algorithm I in that case. We should note, though, that the TSE method may still reduce the stepsizes determined from the stepsize control mechanism.

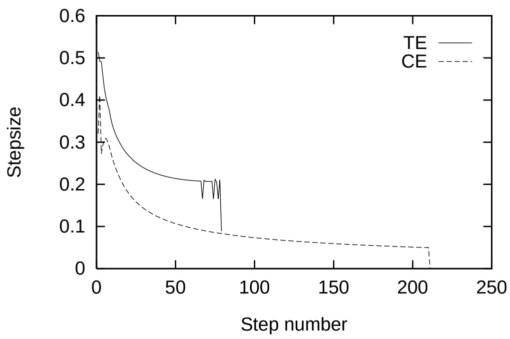
In Table 8.18, we see a further reduction in the number of steps with the TSE method, while the number of steps with the CE method remains the same (except for a slight difference for P1). Note also that with the IHO method, we generally compute smaller enclosures in less time; cf. Tables 8.17 and 8.18.

Problem	Steps		Excess		Time	
	TSE	CE	TSE	CE	TSE	CE
P1	79	211	4.7×10^{-13}	1.6×10^{-15}	1.1×10^{0}	2.9×10^{0}
P2	64	161	1.2×10^{-9}	1.0×10^{-13}	9.8×10^{-2}	2.4×10^{-1}
P3	91	368	4.2×10^{-12}	1.1×10^{-33}	2.8×10^{-1}	1.0×10^{0}
P4	1402	3354	2.0×10^{-11}	3.3×10^{-14}	1.3×10^{1}	2.7×10^{1}

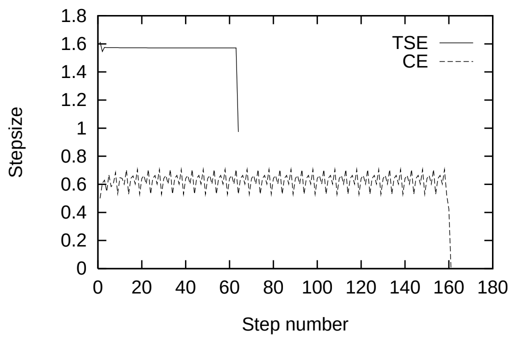
Table 8.17: TSE and CE methods, ITS method, variable stepsize control with $Tol = 10^{-10}$.

Problem	Steps		Excess		Time	
	TSE	CE	TSE	CE	TSE	CE
P1	60	209	2.3×10^{-15}	4.4×10^{-16}	6.8×10^{-1}	2.3×10^{0}
P2	40	161	1.4×10^{-9}	1.4×10^{-13}	1.7×10^{-1}	8.1×10^{-1}
P3	62	368	8.9×10^{-18}	9.1×10^{-34}	3.0×10^{-1}	1.8×10^{0}
P4	976	3354	6.1×10^{-13}	2.1×10^{-14}	1.0×10^{1}	3.3×10^{1}

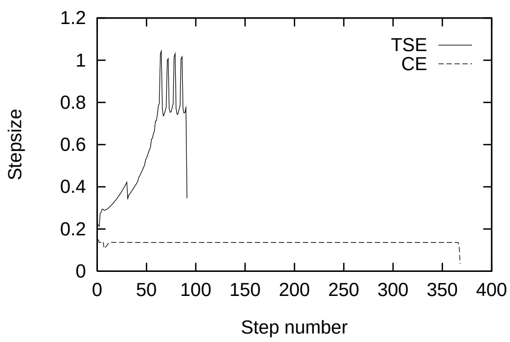
Table 8.18: TSE and CE methods, IHO method, variable stepsize control with $Tol = 10^{-10}$.



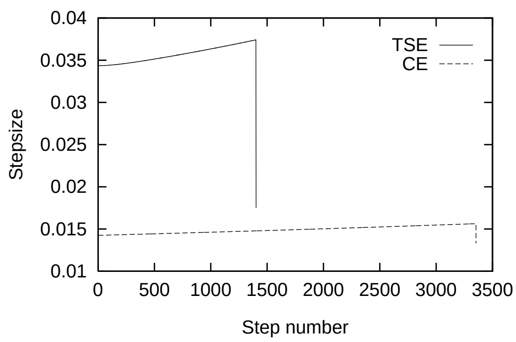
(a) P1



(b) P2

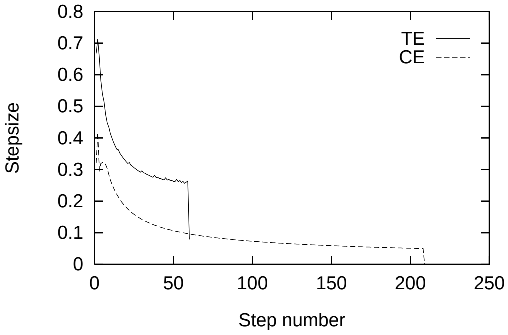


(c) P3

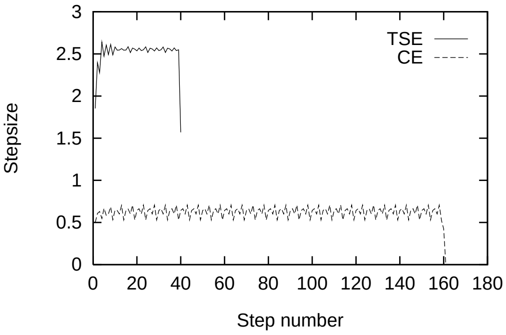


(d) P4

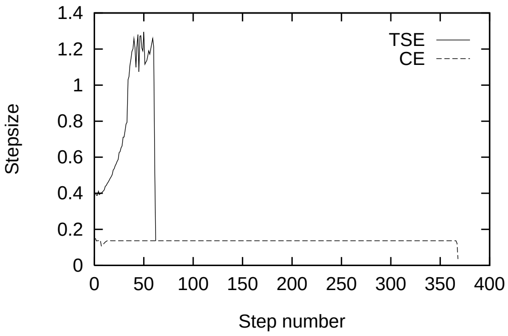
Figure 8.13: TSE and CE methods, ITS method, variable stepsize control with $Tol = 10^{-10}$.



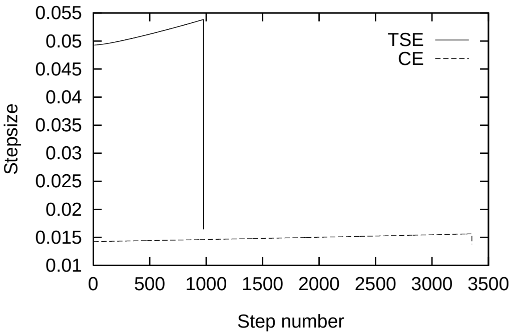
(a) P1



(b) P2



(c) P3



(d) P4

Figure 8.14: TSE and CE methods, IHO method, variable stepsize control with $Tol = 10^{-10}$.

Chapter 9

Conclusions and Directions for Further Research

We have developed and studied an interval Hermite-Obreschkoff method for computing rigorous bounds on the solution of an IVP for an ODE. Compared to interval Taylor series methods with the same order and stepsize, our method has a smaller truncation error, better stability, and is usually less expensive for ODEs for which the right side contains many terms. Although Taylor series methods can be considered as a special case of the more general Hermite-Obreschkoff methods, we have developed a different approach (from Taylor series) to compute bounds for the solution of an IVP for an ODE.

While our study was not directed towards producing an interval method for stiff problems, we have shown that an interval version of a scheme suitable for stiff problems (in traditional numerical methods) may still have a restriction on the stepsize. To obtain an interval method without stepsize limitations, we need to find a scheme with a stable formula not only for advancing the step but also for the truncation error.

We proposed a Taylor series method for validating existence and uniqueness of the solution. This method was designed to ameliorate the stepsize restriction imposed by Algorithm I, but we have not tried to produce an algorithm that always verifies existence and uniqueness (if possible) with the supplied stepsize. Further work is necessary to produce a very good implementation of Algorithm I. Such an implementation can be considered as an optimization problem: maximize the step length, subject to a tolerance restriction.

Our stepsize control mechanism is relatively simple. It worked well for our tests, but we have not performed a thorough empirical investigation. Further studies may be necessary. New developments on stepsize selection for standard and validated ODE methods might be appropriate for considerations in a validated solver; see for example [26] and [36].

There has not been a comprehensive study of order control heuristics. Eijgenraam [19, pp. 129–136] describes the only order selection scheme known to the author. Some insights into the problem of order control are given in [50, pp. 100–118] and [70]. To develop an order control strategy based on the amount of work per step, we need to estimate this work. Obtaining a theoretical bound for the number of arithmetic operations in generating Taylor coefficients for the solution is not difficult, but obtaining a reasonably accurate formula for the number of arithmetic operations in generating their Jacobians is more complex. These Jacobians can be computed by a forward (TADIFF) or a reverse mode (IADOL-C) of automatic differentiation [58], sparsity may or may not be exploited, and different packages may implement the same method differently; for example, with a tape in ADOL-C or using only the main memory as in TADIFF. In addition to estimating the number of floating-point operations, the time spent on memory operations may be nonnegligible.

As the area of validated ODE solving develops, we will need a methodology for assessing validated methods. A part of such a methodology should be an estimate of the amount of work. It may be possible to express it as a number of function and Jacobian evaluations. Then, we may compare validated methods in a framework similar to DETEST [30] or Stiff DETEST [21].

Appendix A

Number of Operations for Generating Taylor Coefficients

We obtain formulas for the number of arithmetic operations for generating one Taylor coefficient and k Taylor coefficients for the solution to $y' = f(y)$, $y(t_0) = y_0$. For simplicity, we assume that the code list of f contains only arithmetic operations. Let N_1, N_2 and N_3 be, respectively, the number of additions (we count subtractions as additions), multiplications, and divisions in the code list of f. If we have computed the Taylor coefficients $(y)_1$, $(y)_2$, ... , $(y)_i$, we can compute the $(i+1)$-st coefficient from

$$f^{[i+1]}(y) \equiv (y)_{i+1} = \frac{1}{i+1} \left(f(y)\right)_i , \tag{A.1}$$

where $(f(y))_i$ is the ith Taylor coefficient of $f(y)$ (see §2.4). The number of arithmetic operations required for computing $(f(y))_i$, using $(y)_0$, $(y)_1$, ... , $(y)_i$, are calculated in Table A.1. If $\mathrm{Ops}\,(g)$ denotes the number of arithmetic operations for computing some function g, then from Table A.1, the number of arithmetic operations to compute $(f(y))_i$ is

$$\begin{aligned} \mathrm{Ops}\,((f(y))_i) &= 2(N_2 + N_3)i + N_1 + N_2 + N_3 \\ &= 2c_f N i + N, \end{aligned} \tag{A.2}$$

Op.	#	Formula	Number of		
			$\pm$	$*$	$/$
$\pm$	N_1	$(u \pm v)_i = (u)_i \pm (v)_i$	N_1	–	–
$*$	N_2	$(uv)_i = \sum_{r=0}^{i} (u)_r (v)_{i-r}$	$N_2 i$	$N_2 i + N_2$	–
$/$	N_3	$\left(\frac{u}{v}\right)_i = \frac{1}{v}\left\{(u)_i - \sum_{r=1}^{i} (v)_r \left(\frac{u}{v}\right)_{i-r}\right\}$	$N_3 i$	$N_3 i$	N_3

Table A.1: Number of additions, multiplications, and divisions for computing $(f(y))_i$.

where $N = N_1 + N_2 + N_3$ and $c_f = (N_2 + N_3)/N$. Because of (A.1), (A.2) also gives the number of operations for computing $f^{[i+1]}(y) = (y)_{i+1}$.[1] Therefore,

$$\begin{aligned} \text{Ops}\left(f^{[i]}(y)\right) &= 2c_f N(i-1) + N = 2c_f Ni + (1 - 2c_f)N \\ &= 2c_f Ni + O(N). \end{aligned} \tag{A.3}$$

The total number of arithmetic operations to compute $k \geq 1$ Taylor coefficients, $(y)_1$, $(y)_2$, ..., $(y)_k$, can be obtained by summing the number of arithmetic operations to compute $(f(y))_i$ for $i = 0, \ldots, k-1$:

$$\begin{aligned} \sum_{i=0}^{k-1} \text{Ops}\left((f(y))_i\right) &= \sum_{i=0}^{k-1} (2c_f Ni + N) \\ &= 2c_f \frac{(k-1)k}{2} N + kN = c_f(k-1)kN + kN \\ &= c_f Nk^2 + O(Nk). \end{aligned} \tag{A.4}$$

[1] We do not count the multiplication $\frac{1}{i+1} \times (f(y))_i$.

Appendix B

A Validated Object-Oriented Solver

B.1 Objectives

Our primary goal is to provide a program environment that will assist researchers in the numerical study and comparison of schemes and heuristics used in computing validated solutions of IVPs for ODEs. The VNODE (Validated Numerical ODE) package that we are developing is intended to be *a uniform implementation of a generic validated solver for IVPs for ODEs.* Uniform means that the design and implementation of VNODE follow well-defined patterns. As a result, implementing, modifying, and using methods can be done systematically. Generic means that the user can construct solvers by choosing appropriate methods from sets of methods. This property enables us to isolate and compare methods implementing the same part of a solver. For example, we can assemble two solvers that differ only in the module implementing Algorithm II. Then, the difference in the numerical results obtained by executing the two solvers will indicate the difference in the performance of Algorithm II. Since we would like to investigate algorithms, being able to isolate them is an important feature of such an environment.

We list and briefly explain some of the goals we have tried to achieve with the design of VNODE. Provided that a validated method for IVPs for ODEs is implemented correctly, the reliability issue does not exist: if a validated solver returns an enclosure of the solution, then the solution is guaranteed to exist within the computed bounds.

Modularity The solver should be organized as a set of modules with well-defined interfaces. The implementation of each module should be hidden, but if necessary, the user should be able to modify the implementation.

Flexibility Since we require well-defined interfaces, we should be able to replace a method, inside a solver, without affecting the rest of it. Furthermore, we should be able to add methods following the established structure and without modifying the existing code.

Efficiency The methods incorporated in VNODE do not have theoretical limits. However, these methods require the computation of high-order Taylor coefficients and Jacobians of Taylor coefficients. As a result, the efficiency of a validated solver is determined mainly by the efficiency of the underlying automatic differentiation package. Other factors that contribute to the performance are: the efficiency of the interval-arithmetic package, the programming language, and the actual implementation of the methods. To achieve flexibility, we may need to repeat the same calculations in two parts of a solver. For example, to separate Algorithm I and Algorithm II, we may need to generate the same Taylor coefficients in both algorithms. However, the repetition of such computations should be avoided.

Since VNODE is to be used for comparing and assessing methods, it has to contain the existing ones. Moreover, VNODE should support rapid prototyping.

B.2 Background

The area of computing validated solutions of IVPs for ODEs is not as developed as the area of computing approximate solutions. Some of the difficulties that arise in interval methods are discussed in Chapter 3 and [52]. With respect to the tools involved, a validated solver is inherently more complex than a classical ODE solver. In addition to an interval-arithmetic package, a major component of a validated solver is the module for automatic generation of interval Taylor coefficients (see §B.4).

Currently, there are three available packages for computing guaranteed bounds on the solution of an IVP for an ODE: AWA [44], ADIODES [69] and COSY INFINITY [8]. We briefly summarize each in turn.

AWA is an implementation of Lohner's method (§3.2.5) and the constant enclosure approach (§3.1). This package is written in Pascal-XSC [37], an extension of Pascal for scientific computing.

ADIODES is a C++ implementation of a solver using the constant enclosure method in Algorithm I and Lohner's method in Algorithm II. The stepsizes in both ADIODES and AWA is restricted to Euler steps by Algorithm I.

COSY INFINITY is a Fortran-based code for study and design of beam physics systems. The method used for verified integration of ODEs is based on high-order Taylor polynomials with respect to time and the initial conditions. The wrapping effect is reduced by establishing functional dependency between initial and final conditions (see [7]). For that purpose, the computations are carried out with Taylor polynomials with real floating-point coefficients and a guaranteed error bound for the remainder term. Thus, the arithmetic operations and standard functions are executed with such Taylor polynomials as operands. Although the approach described in [7] reduces the wrapping effect substantially, working with polynomials is significantly more expensive than working with intervals.

B.3 Object-Oriented Concepts

Since our goal is to build a flexible, easy-to-use, and easy-to-extend package, we have chosen an object-oriented approach in designing VNODE. This is not the first object-oriented design of an ODE solver. The Godess project [57] offers a generic ODE solver that implements traditional methods for IVPs for ODEs. Another successful package is Diffpack [43], which is devised for solving partial differential equations. In [43], there is also an example of how to construct an object-oriented ODE solver.

In this section, we review some object-oriented concepts supported in C++. A good discussion of object-oriented concepts, analysis, and design can be found in [11]. An excellent book on advanced C++ styles and idioms is [12]. A study of nonprocedural paradigms for numerical analysis, including object-oriented ones, is presented in [72].

Data Abstraction

In the object model, a software system can be viewed as a collection of objects that interact with each other to achieve a desired functionality. An object is an instance of a class, which defines the structure and behavior of its objects. By grouping data and methods inside a class and specifying its interface, we achieve encapsulation, separating the interface from the implementation. Hence, the user can change the data representation and the implementation of a method[1] (or methods) of a class without modifying the software that uses it. By encapsulating data, we can avoid function calls with long parameter lists, which are intrinsic to procedural languages like Fortran 77. A class can encapsulate data or algorithms, or both.

[1]We use *method* in two different contexts: to denote a member function of a class or a method in VNODE.

Inheritance and Polymorphism

Inheritance and polymorphism are powerful features of object-oriented languages. Inheritance allows code reuse: the derived class can use the data and functions of its base class(es). Polymorphism serves to apply a given function to different types of objects. Often polymorphism and inheritance are used with abstract classes. An abstract class defines abstract operations, which are implemented in its subclasses; it has no instances and an object of such a class cannot be created.

Operator Overloading

Operator overloading allows the operators of the language to be overloaded for user defined types. To program interval operations without explicit function calls, we have to use a language that supports operator overloading. Without it, programming interval-arithmetic expressions is cumbersome. Both C++ and Fortran 90 provide operator overloading. This feature is used to build interval-arithmetic libraries like PROFIL/BIAS [38] (C++) and INTLIB (Fortran 90) [34].

B.4 Choice of Language: C++ versus Fortran 90

We have chosen C++ [20] over Fortran 90 [47] to implement VNODE. Procedural languages like C or Fortran 77 can be used to implement an object-oriented design [3]. However, using a language that supports object-oriented programming usually reduces the effort for implementing object-oriented software. Our choice was determined by the following considerations, listed in order of importance:

1. availability of software for automatic generation of interval Taylor coefficients;

2. performance and built-in functions of the available interval-arithmetic packages;

3. support of object-oriented concepts; and

4. efficiency.

In this section, we discuss each in turn.

B.4.1 Software for Automatic Generation of Interval Taylor Coefficients

Although packages for automatic differentiation (AD) are available (see for example [33] and [79]), to date, only two free packages for automatic generation of interval Taylor coefficients for the solution of an ODE and the Jacobians of these coefficients are known to the author. These are the FADBAD/TADIFF [5], [6] and IADOL-C [31] packages. They are written in C++ and implement AD through operator overloading.

TADIFF and FADBAD are two different packages. TADIFF can generate Taylor coefficients with respect to time. Then, FADBAD can be used to compute Jacobians of Taylor coefficients by applying the forward mode of AD [58] to these coefficients. FADBAD and TADIFF are not optimized to handle large and sparse systems. Also, they perform all the work in the main memory.

The IADOL-C package is an extension of ADOL-C [25] that allows generic data types. ADOL-C can compute Taylor coefficients by using the forward mode and their Jacobians by applying the reverse mode [67] to these coefficients. The basic data type of ADOL-C is double. To use a new data type in IADOL-C, the user has to overload the arithmetic and comparison operations and the standard functions for that data type. Then, using IADOL-C is essentially the same as using ADOL-C. Since IADOL-C replaces only the double data type of ADOL-C, IADOL-C inherits all the functionality of ADOL-C. However, it was reported that the operator overloading, in IADOL-C, for a basic data type incurs about a three times speed penalty over ADOL-C [31]. This appears to be a phenomenon of the C++ compilers rather than the AD package [31].

The ADOL-C package records the computation graph on a so-called tape. This tape is stored in the main memory, but, when necessary, is paged to disk. When generating Jacobians of Taylor coefficients, ADOL-C exploits the sparsity structure of the Jacobian of the function for computing the right side. Since optimization techniques are used in ADOL-C, we expect the interval version, IADOL-C, to perform better than FADBAD/TADIFF on large and complex problems. But, still, FADBAD/TADIFF should perform well on small to medium-sized problems.

Currently, VNODE is configured with FADBAD/TADIFF, but we have also used IADOL-C. VNODE with these AD packages is based on the INTERVAL data type from the PROFIL/BIAS package, which we discuss in §B.4.2 and §B.4.3.

B.4.2 Interval Arithmetic Packages

The most popular and free interval-arithmetic packages are PROFIL/BIAS [38], written in C++, and INTLIB [35], written in Fortran 77 and available with a Fortran 90 interface [34]. The Fortran 90 version of INTLIB uses operator overloading. For references and comments on other available packages, see for example [34] or [38]. Recently, an interval extension of the Gnu Fortran compiler was reported [65], where intervals are supported as an intrinsic data type.

PROFIL/BIAS seems to be the fastest interval package. In comparison with other such packages, including INTLIB, PROFIL/BIAS is about one order of magnitude faster [38]. Also, PROFIL/BIAS is easy-to-use, and provides matrix and vector operations and essential routines, for example, guaranteed linear equation solvers and optimization routines. For efficiency, it uses the rounding mode of the processor on the machines on which it is installed. Portability is provided by isolating the machine dependent code in small assembler files, which are distributed with the package.

B.4.3 Efficiency

Compared to Fortran, C++ has been criticized for its poor performance for scientific computing. Here, we discuss an important performance problem: the pairwise evaluation of arithmetic expression with arguments of array types (e.g., matrices and vectors). More detailed treatment of this and other problems can be found in [62], [75], and [76].

In C++, executing overloaded arithmetic operations between array data types creates temporaries, which can introduce a significant overhead, particularly for small objects. For example, if `A`, `B`, `C`, and `D` are vectors, the evaluation of the expression

```
D = A + B + C
```

creates two temporaries: one to hold the result of `A + B`, and another to hold the result of `(A + B) + C`. Furthermore, this execution introduces three loops. Clearly, it would be better to compute this sum in one loop without temporaries. In Fortran 90, mathematical arrays are represented as elementary types and optimization is possible at the compiler level.

Because of better optimizing compilers and template techniques [74], [76], C++ is becoming more competitive for scientific computing. A good technique for reducing the overhead in the pairwise evaluation of expressions involving arrays is to use expression templates [74]. The expression template technique is based on performing compile-time transformation of the code using templates. With this technique, expressions containing vectors and matrices can be evaluated in a single pass without allocating temporaries. For example, with expression templates, it is possible to achieve a loop fusion [74], allowing the above sum to be evaluated in a single loop:

```
for ( int i = 1; i <= N; i++ )
        D(i) = A(i) + B(i) + C(i);
```

However, executing this loop in interval arithmetic may not be the best solution for the following reason. Each interval addition in this loop involves two changes of the rounding mode. In modern RISC architectures, rounding mode switches cost nearly the same or even more than floating-point operations [38], [65]. The approach of PROFIL/BIAS is to minimize these switches. Suppose that we want to compute in PROFIL/BIAS

```
C = A + B,
```

where `A`, `B`, and `C` are vectors of the same dimensions. If we denote the components of `A`, `B`, and `C` by a_i, b_i, and c_i, respectively, PROFIL/BIAS changes the rounding mode downwards and computes $\underline{c}_i = \underline{a}_i + \underline{b}_i$, for $i = 1, 2, \ldots, n$. Then, this package changes the rounding mode upwards and computes $\bar{c}_i = \bar{a}_i + \bar{b}_i$, for $i = 1, 2, \ldots n$. Therefore, the result of `A + B` is computed with two rounding mode switches. However, PROFIL/BIAS still creates temporaries.

B.4.4 Support of Object-Oriented Concepts

C++ is a fully object-oriented language, while Fortran 90 is not, because it does not support inheritance and polymorphism. The features of C++ (e.g., data abstraction, operator overloading, inheritance, and polymorphism) allow the goals in §B.1 to be achieved in a relatively simple way. Inheritance and polymorphism can be simulated in Fortran 90 [17], but this is cumbersome.

B.5 The VNODE package

B.5.1 Structure

From an object-oriented perspective, it is useful to think of a numerical problem as an object containing all the information necessary to compute its solution. Also, we can think of a particular method, or a solver, as an object containing the necessary data and functions to perform the integration of a given problem. Then, we can compute a solution by "applying" a method object to a problem object. Most functions in VNODE have

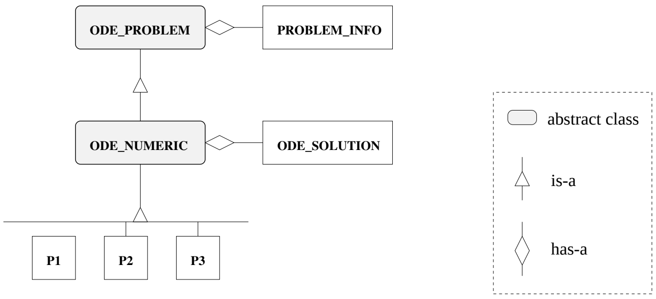


Figure B.1: Problem classes.

such objects as parameters. The description of the numerical problem and the methods in VNODE are implemented as classes in C++.

The problem classes are shown in Figure B.1, and the method classes are shown in Figure B.2. A box in Figures B.1 and B.2 denotes a class; the rounded, filled boxes denote abstract classes. Each of them declares one or more virtual functions, which are not defined in the corresponding abstract class, but must be defined in the derived classes. The lines with $\triangle$ indicate an *is-a* relationship, which can be interpreted as a derived class or as a specialization of a base class; the lines with $\Diamond$ indicate a *has-a* relationship. It is realized either by a complete containment of an object `Y` within another object `X` or by a pointer from `X` to `Y`. The notation in these figures is similar to that suggested in [64].

In the next two subsections, we list the problem and method classes and provide brief explanations. Here, we do not discuss the classes for generating Taylor coefficients in VNODE. A detailed description of VNODE will be given in the documentation of the code at `http://www.cs.toronto.edu/NA`.

Problem Classes Class `ODE_PROBLEM` specifies the mathematical problem, that is, t_0, $[y_0]$, T, and a pointer to a function to compute the right side of the ODE. It also contains a pointer to a class `PROBLEM_INFO`. It indicates, for example, if the problem is constant coefficient, scalar, has a closed form solution, or has a point initial condition. Such information is useful since the solver can determine from it which part of the code to

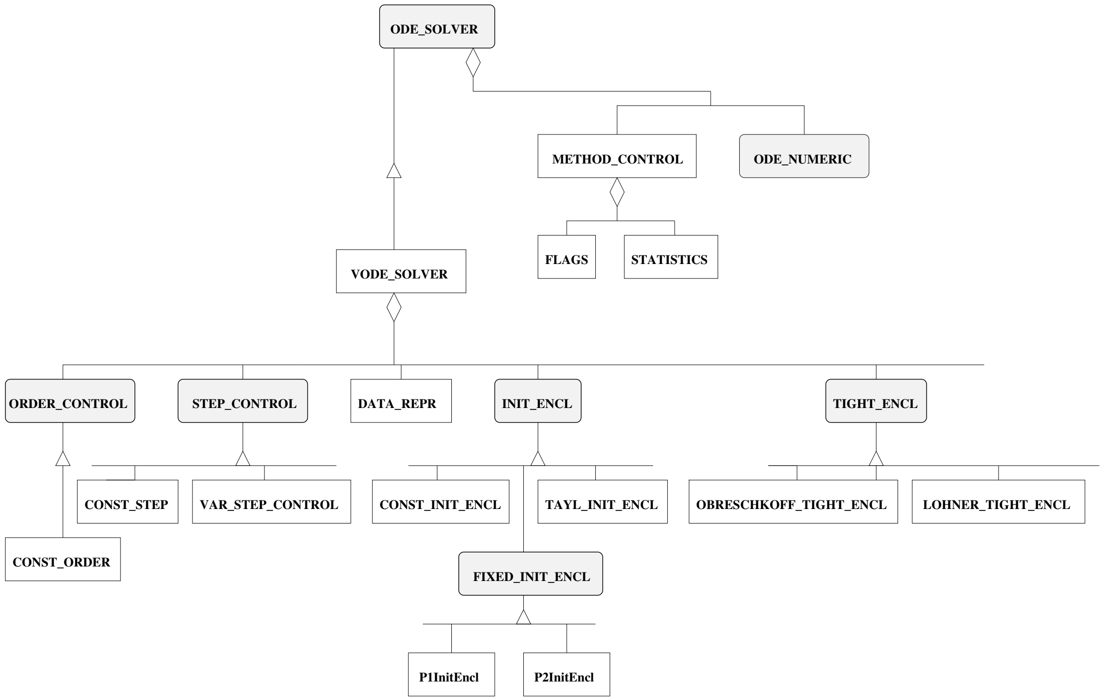


Figure B.2: Method classes.

execute.

`ODE_NUMERIC` specifies the numerical problem. This class contains data such as absolute and relative[2] error tolerances, and a pointer to a class `ODE_NUMERIC` representing a solution. The user-defined problems, `P1`, `P2`, and `P3` in Figure B.1 are derived from this class. New problems can be added by deriving them from `ODE_NUMERIC`.

`ODE_SOLUTION` contains the last obtained a priori and tight enclosures of the solution and the value of t where the tight enclosure is computed. `ODE_SOLUTION` contains also a pointer to a file that stores information from the preceding steps (e.g., enclosures of the solution and stepsizes).

Method Classes Class `ODE_SOLVER` is a general description of a solver that "solves" an `ODE_NUMERIC` problem. `ODE_SOLVER` declares the pure virtual function `Integrate`. Its definition is not provided in this class. As a result, instances of `ODE_SOLVER` cannot be created. This class also contains the class `METHOD_CONTROL`, which includes different flags (encapsulated in `FLAGS`) and statistics collected during the integration (encapsulated in `STATISTICS`).

Class `VODE_SOLVER` implements a general validated solver by defining the `Integrate` function. We have divided this solver into four abstract methods: for selecting an order, selecting a stepsize, and computing initial and tight enclosures of the solution. These methods are realized by the abstract classes `ORDER_CONTROL`, `STEP_CONTROL`, `INIT_ENCL`, and `TIGHT_ENCL`, respectively. Their purpose is to provide general interfaces to particular methods. A new method can be added by deriving it from these abstract classes. `Integrate` performs the integrations by calling objects that are instances of classes derived from `ORDER_CONTROL`, `STEP_CONTROL`, `INIT_ENCL`, and `TIGHT_ENCL`.

`ORDER_CONTROL` has only one derived class, `CONST_ORDER`, whose role is to return a constant order. Currently, VNODE does not implement variable-order methods.

[2]How to specify and interpret relative error tolerance will be discussed in the documentation of VNODE.

For selecting a stepsize, `CONST_STEP` returns a constant stepsize on each step, and `VAR_STEP_CONTROL` implements the stepsize selection scheme from §6.2.

There are two methods for validating existence and uniqueness of the solution in VNODE: a constant enclosure method (`CONST_INIT_ENCL`) and a Taylor series method (`TAYL_INIT_ENCL`). The purpose of the `FIXED_INIT_ENCL` class is to compute a priori enclosures of the solution from the formula for the true solution, if the problem has a closed form solution. This class turns out to be helpful when we want to isolate the influence of Algorithm I, because this algorithm often reduces the input stepsize.

There are two methods for computing a tight enclosure of the solution: an interval Hermite-Obreschkoff method (`OBRESCHKOFF_TIGHT_ENCL`) and Lohner's method (`LOHNER_TIGHT_ENCL`). The `VODE_SOLVER` class has also a pointer to `DATA_REPR`, which is responsible for generating and storing Taylor coefficients and their Jacobians.

B.5.2 An Example Illustrating the Use of VNODE

Suppose that we want to compare two solvers that differ only in the method implementing Algorithm II. In addition, we want to compare them with a constant enclosure method and then with a Taylor series enclosure method in Algorithm I. Here, we show and discuss part of the VNODE code that can be employed for this study. As an example of an ODE, we use Van der Pol's equation, written as a system,

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \mu(1-y_1^2)y_2 - y_1. \end{aligned} \tag{B.5.1}$$

In a traditional ODE solver, we provide a function for computing the right side. In a validated solver, we have to provide also functions for generating Taylor coefficients and their Jacobians. Since we use an AD package for generating such coefficients, we have to specify a function for computing the right side of (B.5.1) for this package. We write the template function

```
template <class YTYPE> void VDPtemplate(YTYPE *yp, const YTYPE *y)
{
   yp[0] =  y[1];
   yp[1] =  MU*(1-sqr(y[0]))*y[1] - y[0];
}
```

which is used by FADBAD/TADIFF and IADOL-C to store the computation graph, and by VNODE to create a function for computing the right side. Then we derive a class VDP from ODE_NUMERIC. Since the details about the declaration of VDP are not essential to understand our example, we omit this declaration.

Figure B.3 shows a typical use of VNODE classes. First, we create an ODE_NUMERIC object[3], VDP, and load the initial condition, the interval of integration, and tolerance by calling the function LoadProblemParam (Part A). For testing, it is convenient to have a function that supplies different sets of data depending on the parameter to this function.

Then, we create methods and return pointers to them (Part B), as described below. ITS and IHO are pointers to objects for computing enclosures using Lohner's and the IHO methods, respectively. InitEncl is a pointer to an object for validating existence and uniqueness of the solution with the constant enclosure method; StepControl refers to an object that implements a variable stepsize control; and OrderControl points to an an object that provides a constant value for the order.

The purpose of class TAYLOR_EXPANSION is to generate and store Taylor coefficients and their Jacobians. It is a template class, for which instances are created by specifying a class for generating Taylor coefficients and a class for generating Jacobians of Taylor coefficients. Here, we create such an instance with parameters VDPTaylGenODE and VDPTaylGenVar, which are classes[4] for generating Taylor coefficients and their Jacobians for (B.5.1).

[3] In Figure B.3, Ptr stands for pointer in PtrODENumeric, PtrTightEncl, etc.
[4] We do not describe these classes here.

In part C, we create two solvers, `SolverITS` and `SolverIHO` and integrate the problem by calling the `Integrate` function on these solvers.[5] Note that they differ only in the method for computing a tight enclosure of the solution. Thus, we can isolate and compare the two methods implementing Algorithm II.

Now, in part D, we want to replace the constant enclosure method for validating existence and uniqueness of the solution with a Taylor series method and repeat the same integrations. We create an instance of `TAYL_INIT_ENCL` by

```
InitEncl =
      new TAYL_INIT_ENCL(ODE->Size,new VDPTaylGenODE,new VDPTaylGenVAR);
```

set it by calling the `SetInitEncl` function, and integrate.

We explain how class `INIT_ENCL` works; the same idea is used in the other abstract classes. `INIT_ENCL` is an abstract class since it contains the pure virtual function

```
virtual void Validate( ... ) = 0;
```

(for simplicity, we leave out the parameters). Each of the derived classes of `INIT_ENCL` must declare a function with the same name and parameters and specify the body of the function. In `Integrate`, there is a call to `Validate`. During execution, depending on the object set, the appropriate `Validate` function will be called. We use *dynamic* or *late* binding: the function that is called is determined by the type of object during the execution of the program. In our example, the method for validating existence and uniqueness is replaced, but the integrator function is not changed. If the user wants to implement his/her own Algorithm I, he/she has to define a derived class of `INIT_ENCL` and an associate `Validate` function.

[5] We omit the details about extracting data after an integration.

```
// ...
// A. Create the ODE problem.
PtrODENumeric ODE = new VDP;
ODE->LoadProblemParam(1);

// B. Create the methods.
int K, P, Q;
K = 11;            // order
P = Q = (K-1)/2;

PtrTightEncl ITS = new LOHNER_TIGHT_ENCL(K);
PtrTightEncl IHO = new OBRESCHKOFF_TIGHT_ENCL(P,Q);

PtrInitEncl  InitEncl  = new CONST_INIT_ENCL(ODE->Size, new VDPTaylGenVAR);
PtrStepCtrl  StepCtrl  = new VAR_STEP_CONTROL(ODE->Size);
PtrOrderCtrl OrderCtrl = new CONST_ORDER(K);
PtrDataRepr  DataRepr  = new TAYLOR_EXPANSION<VDPTaylGenODE,VDPTaylGenVAR>;

// Part C. Create the solvers and integrate.
PtrVODESolver SolverITS = new
   VODE_SOLVER(ODE, DataRepr, OrderCtrl, StepCtrl, InitEncl, ITS);

PtrVODESolver SolverIHO = new
   VODE_SOLVER(ODE, DataRepr, OrderCtrl, StepCtrl, InitEncl, IHO);

SolverITS->Integrate();
SolverIHO->Integrate();

// Part D. Replace the method implementing Algorithm I and integrate.
InitEncl =
   new TAYL_INIT_ENCL(ODE->Size, new VDPTaylGenODE, new VDPTaylGenVAR);

SolverITS->SetInitEncl(InitEncl);
SolverIHO->SetInitEncl(InitEncl);

SolverITS->Integrate();
SolverIHO->Integrate();
// ...
```

Figure B.3: The test code.

Bibliography

[1] E. Adams, D. Cordes, and R. Lohner. Enclosure of solutions of ordinary initial value problems and applications. In E. Adams, R. Ansorge, Chr. Großmann, and H. G. Roos, editors, *Discretization in Differential Equations and Enclosures*, pages 9–28. Akademie-Verlag, Berlin, 1987.

[2] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.

[3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Parallelism in object-oriented numerical software libraries. In Erlend Arge, Are Magnus Bruaset, and Hans Petter Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser, Boston, 1997. See http://www.mcs.anl.gov/petsc/.

[4] H. Bauch and W. Kimmel. Solving ordinary initial value problems with guaranteed bounds. *Z. angew. Math. Mech.*, 69:T110–T112, 1989.

[5] Claus Bendsten and Ole Stauning. FADBAD, a flexible C++ package for automatic differentiation using the forward and backward methods. Technical Report 1996-x5-94, Department of Mathematical Modelling, Technical University of Denmark, DK-2800, Lyngby, Denmark, August 1996. FADBAD is available at http://www.imm.dtu.dk/fadbad.html.

[6] Claus Bendsten and Ole Stauning. TADIFF, a flexible C++ package for automatic differentiation using Taylor series. Technical Report 1997-x5-94, Department of

Mathematical Modelling, Technical University of Denmark, DK-2800, Lyngby, Denmark, April 1997. TADIFF is available at http://www.imm.dtu.dk/fadbad.html.

[7] M. Berz and K. Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4:361–369, 1998.

[8] Martin Berz. COSY INFINITY version 8 reference manual. Technical Report MSUCL–1088, National Superconducting Cyclotron Lab., Michigan State University, East Lansing, Mich., 1997. COSY INFINITY is available at http://www.beamtheory.nscl.msu.edu/cosy/.

[9] Martin Berz, Christian Bischof, and George F. Corliss, editors. *Computational Differentiation: Techniques, Applications, and Tools*. SIAM, Philadelphia, Penn., 1996.

[10] G. Birkhoff and R. S. Varga. Discretization errors for well-set Cauchy problems: I. *J. Math. and Phys.*, 44:1–23, 1965.

[11] Grady Booch. *Object-Oriented Analysis and Design*. The Benjamin/Cummings Publishing Company Inc., Rational, Santa Clara, California, 2nd edition, 1994.

[12] James O. Coplien. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley, AT&T Bell Laboratories, 1992.

[13] G. F. Corliss and R. Rihm. Validating an a priori enclosure using high-order Taylor series. In G. Alefeld and A. Frommer, editors, *Scientific Computing, Computer Arithmetic, and Validated Numerics*, pages 228–238. Akademie Verlag, Berlin, 1996.

[14] George F. Corliss. Survey of interval algorithms for ordinary differential equations. *Appl. Math. Comput.*, 31:112–120, 1989.

[15] George F. Corliss. Guaranteed error bounds for ordinary differential equations. In M. Ainsworth, J. Levesley, W. A. Light, and M. Marletta, editors, *Theory of Numer-*

ics in Ordinary and Partial Differential Equations, pages 1–76. Oxford University Press, 1995.

[16] G. Darboux. Sur les dèveloppements en série des fonctions d'une seule variable. *J. des Mathématique pures et appl.*, pages 291–312, 1876. 3ème série, t. II.

[17] Viktor K. Decyk, Charles D. Norton, and Boleslaw K. Szymanski. Expressing object-oriented concepts in Fortran 90. *ACM Fortran Forum*, 16(1):13–18, April 1997.

[18] B. L. Ehle. On Padé approximations to the exponential function and A-stable methods for the numerical solution of initial value problems. *SIAM J. Math. Anal.*, 4:671–680, 1973.

[19] P. Eijgenraam. *The Solution of Initial Value Problems Using Interval Arithmetic.* Mathematical Centre Tracts No. 144. Stichting Mathematisch Centrum, Amsterdam, 1981.

[20] Margaret A. Ellis and B. Stroustrup. *The Annotated C++ Reference Manual.* Addison–Wesley, 1990.

[21] W. H. Enright, T. E. Hull, and B. Lindberg. Comparing numerical methods for stiff systems of ODEs. *BIT*, 15:10–48, 1975.

[22] Andreas Griewank. ODE solving via automatic differentiation and rational prediction. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1995*, volume 344 of *Pitman Research Notes in Mathematics Series.* Addison-Wesley Longman Ltd, 1995.

[23] Andreas Griewank and George F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application.* SIAM, Philadelphia, Penn., 1991.

[24] Andreas Griewank, George F. Corliss, Petra Henneberger, Gabriella Kirlinger, Florain A. Potra, and Hans J. Stetter. High-order stiff ODE solvers via automatic differentiation and rational prediction. In *Lecture Notes in Comput. Sci., 1196*, pages 114–125. Springer, Berlin, 1997.

[25] Andreas Griewank, David Juedes, and Jean Utke. ADOL-C, a package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Software*, 22(2):131–167, June 1996.

[26] Kjell Gustafsson, Michael Lundh, and Gustaf Söderlind. A PI stepsize control for the numerical solution of ordinary differential equations. *BIT*, 28(2):270–287, 1988.

[27] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems.* Springer-Verlag, 2nd revised edition, 1991.

[28] Ch. Hermite. Extrait d'une lettre de M. Ch. Hermite à M. Borchardt sur la formule d'interpolation de Lagrange. *J. de Crelle*, 84(70):70, 1878. Oeuvres, tome III, p. 432–443.

[29] T. E. Hull and W. H. Enright. A structure for programs that solve ordinary differential equations. Technical Report 66, Department of Computer Science, University of Toronto, May 1974.

[30] T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgwick. Comparing numerical methods for ordinary differential equations. *SIAM J. on Numerical Analysis*, 9(4):603–637, December 1972.

[31] Ronald Van Iwaarden. IADOL-C, personal communications, 1997. IADOL-C is available through the author. E-mail vaniwaar@metsci.com.

[32] L. W. Jackson. Interval arithmetic error–bounding algorithms. *SIAM J. Numer. Anal.*, 12(2):223–238, 1975.

[33] David Juedes. A taxonomy of automatic differentiation tools. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 315–329. SIAM, Philadelphia, Penn., 1991.

[34] R. B. Kearfott. INTERVAL_ARITHMETIC: A Fortran 90 module for an interval data type. *ACM Trans. Math. Software*, 22(4):385–392, 1996.

[35] R. B. Kearfott, M. Dawande, K. Du, and C. Hu. Algorithm 737: INTLIB: A portable Fortran 77 interval standard function library. *ACM Trans. Math. Softw.*, 20(4):447–459, December 1995.

[36] Monika Kerbl. Stepsize strategies for inclusion algorithms for ODE's. In E. Kaucher, S. M. Markov, and G. Mayer, editors, *Computer Arithmetic, Scientific Computation, and Mathematical Modelling*, IMACS Annals on Computing and Appl. Math. 12. J.C. Baltzer, Basel, 1991.

[37] Rudi Klatte, Ulrich Kulisch, Michael Neaga, Dietmar Ratz, and Christian Ullrich. *Pascal–XSC: Language Reference with Examples*. Springer-Verlag, Berlin, 1992.

[38] O. Knüppel. PROFIL/BIAS – a fast interval library. *Computing*, 53(3–4):277–287, 1994. PROFIL/BIAS is available at http://www.ti3.tu-harburg.de/Software/PROFIL/Profil.texinfo_1.html.

[39] Fred T. Krogh. On testing a subroutine for the numerical integration of ordinary differential equations. *J. Assoc. Comput. Mach.*, 20(4):545–562, October 1973.

[40] F. Krückeberg. Ordinary differential equations. In Eldon Hansen, editor, *Topics in Interval Analysis*, pages 91–97. Clarendon Press, Oxford, 1969.

[41] Ulrich W. Kulisch and Willard L. Miranker. *Computer Arithmetic in Theory and Practice*. Academic Press, New York, 1981.

[42] J. D. Lambert. *Computational Methods in Ordinary Differential Equations.* John Wiley & Sons, 1977.

[43] Hans Petter Langtangen. Diffpack. Technical report, SINTEF, Oslo, Norway, June 1996. See http://www.oslo.sintef.no/diffpack/reports/.

[44] Rudolf J. Lohner. *Einschließung der Lösung gewöhnlicher Anfangs– und Randwertaufgaben und Anwendungen.* PhD thesis, Universität Karlsruhe, 1988. AWA is available at ftp://iamk4515.mathematik.uni-karlsruhe.de/pub/awa/.

[45] Rudolf J. Lohner. Step size and order control in the verified solution of IVP with ODE's, 1995. SciCADE'95 International Conference on Scientific Computation and Differential Equations, Stanford University, Calif., March 28 – April 1, 1995.

[46] Rudolph J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. In Edgar W. Kaucher, Ulrich W. Kulisch, and Christian Ullrich, editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, pages 255–286. Wiley-Teubner Series in Computer Science, Stuttgart, 1987.

[47] M. Metcalf and J. Reid. *Fortran 90 Explained.* Oxford University Press, Oxford, England, 1990.

[48] Ramon E. Moore. The automatic analysis and control of error in digital computation based on the use of interval numbers. In Louis B. Rall, editor, *Error in Digital Computation, Vol. I*, pages 61–130. Wiley, New York, 1965.

[49] Ramon E. Moore. Automatic local coordinate transformations to reduce the growth of error bounds in interval computation of solutions of ordinary differential equations. In Louis B. Rall, editor, *Error in Digital Computation, Vol. II*, pages 103–140. Wiley, New York, 1965.

[50] Ramon E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1966.

[51] Ramon E. Moore. A survey of interval methods for differential equations. In *Proceedings of the 23rd Conference on Decision and Control (Las Vegas, 1984)*, pages 1529–1535. IEEE, 1984.

[52] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. Comp.*, (To appear). Available at http://www.cs.toronto.edu/NA/reports.html.

[53] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.

[54] K. L. E. Nickel. Using interval methods for the numerical solution of ODE's. *Z. angew. Math. Mech.*, 66:513–523, 1986.

[55] N. Obreschkoff. Neue Quadraturformeln. *Abh. Preuss. Akad. Wiss. Math. Nat. Kl.*, 4, 1940.

[56] N. Obreschkoff. Sur le quadrature mecaniques. *Spisanie Bulgar. Akad. Nauk. (Journal of the Bulgarian Academy of Sciences)*, 65:191–289, 1942.

[57] Hans Olson. Documentation of the structure of Godess. Technical report, Computer Science, Lund Institute of Technology, S-221 00 Lund, Sweden, November 1995.

[58] Louis B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1981.

[59] A. Ralston. *A First Course in Numerical Analysis*. McGraw-Hill, New York, 2nd edition, 1978.

[60] Robert Rihm. Interval methods for initial value problems in ODEs. In Jürgen Herzberger, editor, *Topics in Validated Computations: Proceedings of the IMACS-GAMM International Workshop on Validated Computations, University of Olden-*

burg, Elsevier Studies in Computational Mathematics, pages 173–207. Elsevier, Amsterdam, New York, 1994.

[61] Robert Rihm. On a class of enclosure methods for initial value problems. *Computing*, 53:369–377, 1994.

[62] Arch D. Robinson. C++ gets faster for scientific computing. *Computers in Physics*, 10(5):458–462, Sep/Oct 1996.

[63] J. Rohn. NP-hardness results for linear algebraic problems with interval data. In J. Herzberger, editor, *Topics in Validated Computations*, volume 5 of *Studies in Computational Mathematics*, pages 463–471, North-Holland, Amsterdam, 1994.

[64] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, New York, 1991.

[65] Michael J. Schulte, Vitaly Zelov, Ahmet Akkas, and James Craig Burley. The interval-enhanced GNU Fortran compiler. *Reliable Computing*, (Submitted), October 1998.

[66] Lawrence F. Shampine. *Numerical solution of ordinary differential equations.* Chapman & Hall, New York, 1994.

[67] B. Speelpenning. *Compiling Fast Partial Derivatives of Functions Given by Algorithms.* PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, January 1980.

[68] H. Spreuer and E. Adams. On the existence and the verified determination of homoclinic and heteroclinic orbits of the origin for the Lorenz system. *Computing Suppl.*, 9:233–246, 1993.

[69] Ole Stauning. Automatic validation of numerical solutions. Technical Report IMM-PHD-1997-36, IMM, Lyngby, Denmark, October 1997. ADIODES is available at http://www.imm.dtu.dk/~os/ADIODES.tar.gz.

[70] Hans J. Stetter. Algorithms for the inclusion of solutions of ordinary initial value problems. In Jaramír Vosmanský and Milŏs Zlámal, editors, *Equadiff 6: Proceedings of the International Conference on Differential Equations and Their Applications (Brno, 1985)*, volume 1192 of *Lecture Notes in Mathematics*, pages 85–94. Springer Verlag, Berlin, 1986.

[71] Hans J. Stetter. Validated solution of initial value problems for ODEs. In Christian Ullrich, editor, *Computer Arithmetic and Self-Validating Numerical Methods*, pages 171–187. Academic Press, New York, 1990.

[72] Stephen J. Sullivan and Benjamon G. Zorn. Numerical analysis using nonprocedural paradigms. *ACM TOMS*, 21(3):267–298, Sept 1995.

[73] R. S. Varga. On higher order stable implicit methods for solving parabolic differential equations. *J. Math. and Phys.*, 40:220–231, 1961.

[74] T. Veldhuizen. Expression templates. *C++ Report*, 7(5):26–31, June 1995.

[75] T. Veldhuizen. Scientific computing: C++ versus Fortran. *Dr. Dobb's Journal*, 34, November 1997.

[76] T. L. Veldhuizen and M. E. Jernigan. Will C++ be faster than Fortran? In *Proceedings of the 1st International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'97)*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[77] G. Wanner. On the integration of stiff differential equations. Technical report, Université de Genéve, Section de Mathematique, 1211 Genéve 24th, Suisse, October 1976.

[78] G. Wanner. On the integration of stiff differential equations. In *Proceedings of the Colloquium on Numerical Analysis*, volume 37 of *Internat. Ser. Numer. Math.*, pages 209–226, Basel, 1977. Birkhäuser.

[79] Wenhong Yang and George Corliss. Bibliography of computational differentiation. In Martin Berz, Christian H. Bischof, George F. Corliss, and Andreas Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 393–418. SIAM, Philadelphia, Penn., 1996.