

Appeared in: Numer. Funct. Anal. and Optimiz. **20** (1999), 779–803.

An Enclosure Method for the Solution of Linear ODEs with Polynomial Coefficients

Markus Neher

Institut für Angewandte Mathematik
Universität Karlsruhe
D-76128 Karlsruhe, Germany
E-mail: markus.neher@math.uni-karlsruhe.de

Abstract

The solution $y(x)$ of an initial value problem for a linear ODE with polynomial coefficients is represented as a power series. The remainder series is bounded by a geometric series, resulting in close bounds for y . The method has been implemented on a computer, where highly accurate results are attained with the staggered correction technique. For guaranteed enclosures on the computer, all roundoff errors have to be taken into account in the course of computation.

Key words: ordinary differential equations, initial value problems, enclosure methods, interval computations.

AMS Subject Classification: 65L05, 65L70, 65G10.

1 Introduction

The numerical solution of the initial value problem $y'(x) = f(x, y)$, $y(a) = y_a$, usually aims at finding the value of y at a given point $b > a$. Even if a continuous approximation of y on the interval $[a, b]$ is desired, then most often approximate values $v_j \approx y(x_j)$ are computed for a finite number of discrete points $x_j \in [a, b]$, and the continuous approximation is obtained by interpola-

tion. In particular, the Taylor series method, finite-difference methods, and Runge-Kutta methods work this way.

When the differential equation is replaced by a set of (linear or nonlinear) equations for discrete values $y(x_j)$, its numerical solution is subject to discretization errors. If the computation is carried out on a digital computer with a floating point number system, the result is also impaired by round-off errors stemming from the finite number set on the computer. In general, neither the discretization errors nor the roundoff errors are easily bounded. Unfortunately, many algorithms provide only error estimates, even though a guess on the error in computation is not essentially better than a guess on the solution alone.

To achieve certainty, the error must be bounded. For this purpose, enclosure methods have been developed. The introduction of computers facilitated the construction of new numerical algorithms with computable error bounds (see [4, 21, 22, 25] and the numerous references given there). Instead of approximations, these algorithms yield verified bounds of the solution (in our case, intervals $[\underline{y}_j, \bar{y}_j]$ that contain the respective values $y(x_j)$). Interval computations as described in [1, 18] have proved an especially useful tool in enclosure methods.

An interval computation based enclosure method for ordinary differential equations was developed by Moore [16, 17, 18] in 1965, the most popular one today is the method of Lohner [12, 13, 14], implemented in his computer program AWA. These methods (and some others, see [5, 6, 20] for more references) are based on the Taylor series method, with the additional computation of the truncation error of the series in every integration step. In [13], Lohner applied his method successfully to many linear and nonlinear problems.

Nevertheless, there are two drawbacks of these methods. First, they need an *a priori*-enclosure of the solution to start the computation. Usually, *a priori*-enclosures can only be secured for small step sizes $h_j := x_{j+1} - x_j$, so that these algorithms are sometimes slow compared to popular approximate schemes ([13, p. 41]; see [7, 24] for possible improvements).

The second problem faced with Moore's or Lohner's method (and their many variants) is known as wrapping effect, which may cause a dramatic increase of the diameters of the computed interval enclosures of the solution (see Moore [17, 18]). The wrapping effect usually becomes worse when the step sizes are being reduced.

In this paper, a new enclosure method is investigated to overcome these difficulties. The method is applicable to a class of initial value problems with a special structure. It is also based on Taylor series expansion, but it does not require an *a priori*-enclosure of the solution, and the numerical integration

can be carried out with very large step sizes so that the wrapping effect is of minor importance.

The paper is structured as follows: after the presentation of the problem in Section 2, the computation of an approximate solution and the error analysis with exact (real) arithmetic is discussed in Section 3, and an algorithm that computes bounds to the solution of the given problem is presented. In Section 4, general difficulties that arise with the implementation of the proposed algorithm on digital computers are exhibited. Necessary modifications of the algorithm are explained in Sections 5 and 6. In particular, in Section 6 the employment of interval computations is described. More practical aspects of the implementation of our method are then discussed in Section 7. In the last section, numerical examples are given.

2 Formulation of the problem

Consider the initial value problem

$$\begin{aligned} y^{(n)}(x) &= \sum_{i=0}^{n-1} p_i(x) y^{(i)}(x) + p(x), \\ y^{(i)}(0) &= y_{i0}, \quad i = 0, 1, \dots, n-1, \end{aligned} \tag{1}$$

where

$$p_i(x) = \sum_{j=0}^m b_{ij} x^j, \quad p(x) = \sum_{j=0}^m b_j x^j \quad (b_{ij}, b_j \in \mathbb{R}, m \in \mathbb{N}_0)$$

are polynomials of degree at most m . For some $h > 0$ we seek an enclosure of $y(h)$. Without loss of generality, the initial values are supposed to be given at $x = 0$. If they are given at some $x_0 \neq 0$ then let $u(x) := y(x - x_0)$ and reformulate the problem for u .

It is well known (cf. [3], Ch. III) that the solution of (1) can be written as a power series

$$y(x) := \sum_{k=0}^{\infty} a_k x^k, \tag{2}$$

and that the series converges for all $x \in \mathbb{R}$. The first n coefficients a_0, \dots, a_{n-1} are determined from the initial values for the solution. Differentiating (2), inserting the series and its derivatives into (1), and equating terms with equal powers of x yields recurrence formulas for the coefficients with index greater than $n - 1$. Let

$$P(k, i) := (k+1) \cdots (k+i), \quad P(k, 0) := 1 \quad \text{for } i \in \mathbb{N}, k \in \mathbb{N}_0,$$

then

$$a_k = \frac{y_{k0}}{k!} \quad \text{for } k = 0, \dots, n-1, \quad (3)$$

$$a_{k+n} = \sum_{i=0}^{n-1} \sum_{j=0}^k \frac{P(k-j, i) b_{ij}}{P(k, n)} a_{k+i-j} + \frac{b_k}{P(k, n)} \quad \text{for } k = 0, 1, \dots, m, \quad (4)$$

and

$$a_{k+n} = \sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) b_{ij}}{P(k, n)} a_{k+i-j} \quad \text{for } k > m. \quad (5)$$

For an approximate solution to (1), compute finitely many, say κ , coefficients a_k from (3), (4), and (5) and then use the truncated series

$$s(\kappa) := \sum_{k=0}^{\kappa-1} a_k h^k \quad (6)$$

as an approximation to $y(h)$. The approximation error is then the truncation error given by the infinite series

$$r(\kappa) := \sum_{k=\kappa}^{\infty} a_k h^k. \quad (7)$$

In the next section, we bound this series and thus procure an enclosure of $y(h)$.

3 Estimation of the truncation error

We first present two lemmas on some properties of $P(k, i)$. The proofs are given in the appendix.

Lemma 1 For $h, \omega > 0$, $k \geq nm$,

$$\sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) |b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n)} \quad (8)$$

is a strictly decreasing function of k .

Lemma 2 For $h, \omega > 0$, there is $\kappa = \kappa(h, \omega) \in \mathbb{N}$ so that the recess condition

$$\sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) |b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n)} \leq 1 \quad (9)$$

holds for all $k \geq \kappa$.

The truncation error $r(\kappa)$ given by (7) will now be bounded to yield a verified enclosure of $y(h)$. The basic idea of our approach is motivated by the fast convergence of (2).

$|r(\kappa)|$ is estimated by a geometric series: For $\omega \in (0, 1)$, let

$$c_k := a_k \left(\frac{h}{\omega}\right)^k.$$

Then

$$y(h) = \sum_{k=0}^{\kappa-1} a_k h^k + \sum_{k=\kappa}^{\infty} c_k \omega^k, \quad (10)$$

and if

$$|c_k| \leq C$$

can be proved for some constant $C > 0$ and all $k \geq \kappa$, then $r(\kappa)$ is bounded by

$$t(\kappa) := \sum_{k=\kappa}^{\infty} C \omega^k = \frac{C \omega^\kappa}{1 - \omega} \quad (11)$$

so that

$$|y(h) - s(\kappa)| \leq \frac{C \omega^\kappa}{1 - \omega}. \quad (12)$$

It is worth mentioning that enclosures of $y(x)$ for all $x \in [-h, h]$ follow immediately. From the definition of c_k and (10),

$$y(x) = \sum_{k=0}^{\kappa-1} a_k x^k + \sum_{k=\kappa}^{\infty} c_k \left(\frac{x}{h} \omega\right)^k, \quad x \in [-h, h].$$

Hence, if $|c_k| \leq C$ then

$$\left| y(x) - \sum_{k=0}^{\kappa-1} a_k x^k \right| \leq C \sum_{k=\kappa}^{\infty} \left(\frac{|x|}{h} \omega\right)^k = \frac{C \omega^\kappa}{h^{\kappa-1}} \cdot \frac{|x|^\kappa}{h - |x| \omega}$$

for all $x \in [-h, h]$. This error bound attains its maximum at the endpoints of the interval $[-h, h]$.

To compute a suitable C for the above estimation, multiply (5) with $\left(\frac{h}{\omega}\right)^{k+n}$ to obtain

$$c_{k+n} = \sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) b_{ij} \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n)} c_{k+i-j}$$

for $h, \omega > 0$, and $k > m$. Taking absolute values, we get

$$\begin{aligned} |c_{k+n}| &\leq \sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) |b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n)} |c_{k+i-j}| \\ &\leq \max_{\nu=-m}^{n-1} |c_{k+\nu}| \sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) |b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n)}. \end{aligned}$$

If the recess condition (9) is fulfilled for some $k = \kappa \geq mn$, then

$$|c_{\kappa+n}| \leq \max_{\nu=-m}^{n-1} |c_{\kappa+\nu}|,$$

and by induction (using Lemma 1)

$$|c_{\kappa+j}| \leq \max_{\nu=-m}^{n-1} |c_{\kappa+\nu}| =: C \quad \text{for all } j \geq n \quad (13)$$

follows. By construction, C depends on κ , h , and ω .

We have proved the main result of this paper:

Theorem 1 *For any $\omega \in (0, 1)$ there is $\kappa \in \mathbb{N}$ and $C \in \mathbb{R}$ so that*

$$\left| y(h) - \sum_{k=0}^{\kappa+n-1} a_k h^k \right| \leq \frac{C\omega^{\kappa+n}}{1-\omega}.$$

Note that $\omega^\kappa \rightarrow 0$ for $\kappa \rightarrow \infty$ and $C \rightarrow 0$ for $\kappa \rightarrow \infty$ and any fixed ω and h , so that the absolute approximation error can be made arbitrarily small by choosing a suitable κ . The same holds for the relative approximation error, if $y(h) \neq 0$.

The following Algorithm 1 computes an enclosure of $y(h)$ according to Theorem 1. Input data are the coefficients b_{ij}, b_k of the polynomials in the differential equation, the initial values y_{i0} , the argument h , and two parameters $\varepsilon_{\text{rel}}, \varepsilon_{\text{abs}} > 0$ that control the accuracy of the computed enclosure of $y(h)$. The iteration is stopped (after finitely many steps, since $t(k) \rightarrow 0$) when the relative or the absolute error is less than ε_{rel} or ε_{abs} , respectively.

Algorithm 1

Enclosure method for ODEs with polynomial coefficients

1. For $k = 0, \dots, n - 1$:
 Compute a_k from (3).
2. For $k = 0, \dots, nm - 1$:
 Compute a_{k+n} from (4) and (5).
3. Choose $\omega \in (0, 1)$.
4. For $k = nm, nm + 1, \dots$:
 - (a) If the recess condition (9) is not fulfilled then goto (e).
 - (b) Let $C := \max_{\nu=-m}^{n-1} |a_{k+\nu} \left(\frac{h}{\omega}\right)^{k+\nu}|$.
 - (c) Compute $s(k+n) := \sum_{\nu=0}^{k+n-1} a_{\nu} h^{\nu}$, $t(k+n) := \frac{C \omega^{k+n}}{1-\omega}$.
 - (d) If $\left|\frac{t(k+n)}{s(k+n)}\right| \leq \varepsilon_{\text{rel}}$ or $|t(k+n)| \leq \varepsilon_{\text{abs}}$ then goto 5.
 - (e) Compute a_{k+n} from (5).
5. Terminate with $y(h) \in s(k+n) + [-t(k+n), t(k+n)]$.

4 Use of a floating point arithmetic

On a digital computer, a straightforward implementation of Algorithm 1, that is, replacing all arithmetic operations between real numbers by corresponding approximate operations between machine representable numbers, may fail because of roundoff errors.

To illustrate some of the difficulties arising with the use of a floating point arithmetic, we consider the model problem

$$y'' = y, \quad y(0) = 1, \quad y'(0) = -1.$$

Its exact solution is $y(h) = e^{-h}$, or rather

$$y(h) = \sum_{k=0}^{\infty} \frac{(-h)^k}{k!}.$$

Rounded to 4 decimal digits, $y(10) \approx 4.540 \times 10^{-5}$. However, when this value is computed via the series, it is the result of a summation of numbers with

alternating signs, that themselves are of much larger size. For $h = 10$, the summands with largest absolute value are $\frac{(-h)^9}{9!}$ and $\frac{(-h)^{10}}{10!} \approx 2.756 \times 10^3$, and the computation of $y(10)$ with 4 decimal digits of accuracy in the result requires an arithmetic of at least 12 decimal digits and the summation of at least 40 summands of the series (since $\frac{10^{40}}{40!} \approx 1.226 \times 10^{-8}$). For $h = 1000$, similar calculations show that the length of the mantissa must be at least 871, and thousands of summands $a_k h^k$ have to be computed to guarantee 4 decimal digits of accuracy in the result ($y(1000) \approx 5.076 \times 10^{-435}$, $\frac{1000^{1000}}{1000!} \approx 2.485 \times 10^{432}$).

From the above example it is obvious that the evaluation of (6) is the crucial element of a practical implementation of Algorithm 1 on a digital computer. In some cases, the employment of a multiple precision arithmetic is sufficient to procure an accurate result, even if the summands of (6) cancel out each other. In our implementation of Algorithm 1, a multiple precision arithmetic that is based on the staggered correction technique is used. This will be described in detail in Section 7.

However, examples where the range of the summands of (6) is not contained in the set of machine representable numbers on the computer are easily conceived. If such a numerical overflow occurs, the interval $[0, h]$ must be divided into a number of short enough subintervals $I_j = [x_{j-1}, x_j]$, $j = 1, \dots, J$, so that Algorithm 1 can be used on each subinterval with step length $h_j = x_j - x_{j-1}$ instead of h (note that for $h \rightarrow 0$, the partial sum (6) is more and more well-behaved with respect to cancellation). In this case, on each subinterval starting values not only for the solution $y(x)$ of (1), but also for its first $n - 1$ derivatives are needed. How these values are gained will be shown in Section 5.

The splitting of the integration domain into subintervals poses another problem. When the solution of (1) is unknown, exact values of the function or of its derivatives at intermediate points are in general neither available nor machine representable. Instead, bounds of y and its derivatives at intermediate points x_j must be used to continue the integration. Hence, the initial values for all but the first subintervals consist of intervals instead of real numbers. The treatment of interval initial values will be the subject of Section 6.

5 Enclosures of derivatives

Differentiating (2), we have

$$y^{(l)}(h) = \sum_{k=l}^{\infty} P(k-l, l) a_k h^{k-l},$$

where the a_k are again defined by (3) and the recurrence formulas (4) and (5). For $k \geq l$, let

$$d_{kl} := P(k-l, l) a_k \left(\frac{h}{\omega}\right)^{k-l},$$

then for $\kappa > l$

$$y^{(l)}(h) = \sum_{k=l}^{\kappa-1} P(k-l, l) a_k h^{k-l} + \sum_{k=\kappa}^{\infty} d_{kl} \omega^{k-l}.$$

From (4) and (5), recurrence relations for the d_{kl} can be derived. For $1 \leq l \leq n-1$ and $k \geq l+m$,

$$d_{k+n,l} = \sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) b_{ij} \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n-l) P(k+i-j-l, l)} d_{k+i-j,l}.$$

With the same reasoning as in the proof of Lemma 2, we deduce

Corollary 1 *For $h, \omega > 0$, $1 \leq l \leq n-1$, there is a $\kappa \in \mathbb{N}$, so that*

$$\sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) |b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n-l) P(k+i-j-l, l)} \leq 1$$

for all $k \geq \kappa$.

Hence, the estimation of $\sum_{k=\kappa}^{\infty} d_{kl} \omega^{k-l}$ can be carried out as for $r(\kappa)$, so that bounds for derivatives of y can be computed the same way as bounds for y itself.

6 Interval initial values

To facilitate the understanding for the reader who is not familiar with the concept of interval computations, we first introduce the interval notation and some basic properties used in the following. For a detailed introduction to interval computation see [1].

Real bounded and closed intervals are denoted by $[a] = [\underline{a}, \bar{a}]$, $[b] = [\underline{b}, \bar{b}]$, etc. Real numbers a are identified with point intervals $a = [a, a]$. The same notation is used for interval vectors, e.g. $[a] = ([a_i])$. The space of m -dimensional interval vectors is denoted by \mathbb{IR}^m . For $m = 1$, \mathbb{IR} is written instead of \mathbb{IR}^1 .

Real (m, m) -matrices are denoted by $A = (a_{ij})$, the corresponding interval matrices by $[A] = ([a_{ij}])$.

The basic arithmetic operations between intervals are defined by

$$[\underline{a}, \bar{a}] * [\underline{b}, \bar{b}] = \{ a * b \mid a \in [\underline{a}, \bar{a}], b \in [\underline{b}, \bar{b}] \}, \quad * \in \{+, -, \cdot, /\}.$$

The same concept applies to operations between interval vectors or interval matrices. A fundamental property of interval computations is the inclusion monotonicity of the basic arithmetic operations. For $* \in \{+, -, \cdot, /\}$,

$$[\underline{a}, \bar{a}] \subseteq [\underline{c}, \bar{c}], [\underline{b}, \bar{b}] \subseteq [\underline{d}, \bar{d}] \Rightarrow [\underline{a}, \bar{a}] * [\underline{b}, \bar{b}] \subseteq [\underline{c}, \bar{c}] * [\underline{d}, \bar{d}]$$

([1, Theorem 1.5]).

The midpoint $\frac{1}{2}(\underline{a} + \bar{a})$ of an interval $[a]$ is denoted by $\text{mid}([a])$, the diameter of $[\underline{a}, \bar{a}]$ by

$$d([\underline{a}, \bar{a}]) = \bar{a} - \underline{a}.$$

The diameter of a sum of two intervals $[a]$ and $[b]$ is the sum of the diameters of $[a]$ and $[b]$. The diameter of the product of a real number a and an interval $[b]$ is given by ([1, Theorem 2.9])

$$d(a \cdot [b]) = |a| \cdot d([b]). \quad (14)$$

Intervals in equations usually take the place of parameters whose exact value is not known. For example, if in (1) interval initial values $[y_{io}]$ are given instead of real numbers y_{io} , then (1) is interpreted as the set of all initial value problems that are obtained for all possible choices of $y_{io} \in [y_{io}]$, $i = 0, \dots, n-1$. In other words, interval initial values define the family of all solutions of the differential equation in (1) that run through a given n -dimensional interval in the phase plane.

When dealing with interval initial values in (1), our goal is an interval that contains $y(h)$ for all $y_{io} \in [y_{io}]$, $i = 0, \dots, n-1$. Such an interval could be computed with Algorithm 1 in a straightforward manner, if all computations were performed with interval arithmetic operations. However, from (14) it follows that one should not insert intervals into recurrence formulas like (5). For example, if enclosures of $a_k h^k$ were computed according to

$$[a_k h^k] := \frac{[y_{k0}]}{k!} h^k \quad \text{for } k = 0, \dots, n-1,$$

$$[a_{k+n} h^{k+n}] := \sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) b_{ij} h^{n-i+j}}{P(k, n)} [a_{k+i-j} h^{k+i-j}] \quad \text{for } k = 0, 1, \dots,$$

then due to (14),

$$d([a_{k+n} h^{k+n}]) = \sum_{i=0}^{n-1} \sum_{j=0}^m \left| \frac{P(k-j, i) b_{ij} h^{n-i+j}}{P(k, n)} \right| d([a_{k+i-j} h^{k+i-j}]),$$

so that the diameters $d([a_{k+n} h^{k+n}])$ would grow as long as

$$\sum_{i=0}^{n-1} \sum_{j=0}^m \left| \frac{P(k-j, i) b_{ij} h^{n-i+j}}{P(k, n)} \right| > 1,$$

and the diameter of $\sum_{k=0}^{\kappa-1} [a_k h^k]$ would become too large for a practical enclosure of $s(\kappa)$.

Since the differential equation in (1) is linear, the mean value method (cf. [20]) is especially simple to apply. In the case of interval initial values, instead of computing enclosures of the individual summands, an enclosure of a fundamental system of the differential equation is computed and the enclosure of the interval initial value problem is then built of linear combinations of the latter.

For $\nu = 0, 1, \dots, n-1$, let $u_\nu(x)$ denote the solution of the initial value problem

$$\begin{aligned} u_\nu^{(n)} &= \sum_{i=0}^{n-1} p_i(x) u_\nu^{(i)}, \\ u_\nu^{(\mu)}(0) &= \delta_{\mu\nu}, \quad \mu = 0, 1, \dots, n-1, \end{aligned}$$

where $\delta_{\mu\nu}$ is the Kronecker symbol. Further let $u_n(x)$ be the solution of the initial value problem

$$\begin{aligned} u_n^{(n)} &= \sum_{i=0}^{n-1} p_i(x) u_n^{(i)} + p(x), \\ u_n^{(\mu)}(0) &= \text{mid}([y_{\mu 0}]), \quad \mu = 0, 1, \dots, n-1, \end{aligned}$$

and let finally $[u_\nu]$ denote the enclosure of $u_\nu(h)$ (as it is delivered by Algorithm 1), and define the interval vector $[u] = ([u_\nu])_{\nu=0}^n$. Then for any $y_{i0} \in [y_{i0}]$, $i = 0, \dots, n-1$, the value $y(h)$ of the solution of (1) is contained in the scalar product of two interval vectors:

$$y(h) \in [u]^T \cdot \begin{pmatrix} [y_{00}] - \text{mid}([y_{00}]) \\ \vdots \\ [y_{n-1,0}] - \text{mid}([y_{n-1,0}]) \\ 1 \end{pmatrix}.$$

Thus, by solving $n+1$ initial value problems with real initial values instead of only one with interval initial values, a tight enclosure of $y(h)$ is acquired.

7 Implementation

We now describe the practical implementation of Algorithm 1 on a digital computer, where only a limited number set S of machine representable numbers is available.

To ensure validated results, it is assumed that all computations are performed with a reliable interval arithmetic on the computer, as it was described by Kulisch and Miranker [11]. Let IS denote the set of machine intervals (intervals with machine representable bounds), then the result of any computation in IS must include the result of the corresponding calculation in \mathbb{IR} (in other words, all roundoff errors must be enclosed in the result of any computation). Programming languages like PASCAL-XSC, C-XSC, or FORTRAN-XSC (cf. [9, 10, 26]) have a built-in machine interval arithmetic of this kind and thus enable the automatic control of roundoff errors on the computer. With such a computer arithmetic, Algorithm 1 is performed with machine intervals instead of real numbers, and all arithmetic operations between real numbers are replaced by their corresponding machine interval extensions.

As another important feature of the computer arithmetic, a precise scalar product is required, with which the computation of scalar products of (real or interval) vectors is performed with one single rounding. While the interval arithmetic is used to enclose the roundoff errors, the use of the precise scalar product ensures that the roundoff errors are being kept sufficiently small.

In our model problem in Section 3, we have shown that the summands of (6) may cover a much larger number range than the built-in single precision of a standard computer. To avoid cancellation and thus loss of accuracy in the result, the summation in (6) must then be carried out with multiple precision. Unfortunately, the necessary precision in the computation is generally not known before. If the chosen mantissa length is too small, then the computation will provide a result of too less accuracy. If it is chosen too large, then the computation may be unnecessarily lavish and slow. Hence, instead of a fixed multiple precision number format, we rather use the staggered correction technique (cf. [15, 23]) to store machine numbers in multiple precision format of variable mantissa length. The advantage of the staggered correction technique over the multiple precision arithmetic with a fixed longreal number format is that with staggered correction, the length of the mantissa can be adapted to the necessary accuracy while the computation is performed.

The staggered correction technique uses several machine numbers to store a real number with higher precision. A staggered correction format of length

L of a real number x is given by

$$x = \sum_{l=0}^L g_l, \quad g_l \in S. \quad (15)$$

For higher precision of x , the numbers g_l must have exponents of different size. If b is the base of the number system, M is the length of the mantissa and if the mantissas do not overlap, that is if

$$|g_l| > b^M |g_{l+1}|$$

holds in (15), then the precision is at least $(L + 1)M$ mantissa digits. In an interval staggered correction format of an interval $[x]$, g_L is replaced by a machine interval $[g_L]$.

In the staggered correction implementation of Algorithm 1, the enclosure of the numbers $a_k h^k$ is stored as a sum of intervals

$$a_k h^k \in \sum_{l=0}^L [g_{kl}],$$

where $[g_{k0}], [g_{k1}], \dots, [g_{k,L-1}]$ are machine representable point intervals. To compute them, a midpoint function $\text{mid}: IS \rightarrow S$ with $\text{mid}([x])$ the machine number next to the midpoint of the interval $[x]$ is used.

Algorithm 2	
Computation of $a_k h^k$ with staggered correction ($k > m$)	
1.	$[g_{k+n,0}] := \frac{1}{P(k,n)} \left(\sum_{i=0}^{n-1} \sum_{j=0}^m P(k-j,i) b_{ij} h^{n-i+j} \sum_{l=0}^L [g_{k+i-j,l}] \right)$
2.	For $\mu = 1, \dots, L$: <ul style="list-style-type: none"> (a) $[g_{k+n,\mu-1}] := \text{mid}([g_{k+n,\mu-1}])$ (b) $[z] := \sum_{i=0}^{n-1} \sum_{j=0}^m P(k-j,i) b_{ij} h^{n-i+j} \sum_{l=0}^L [g_{k+i-j,l}] - P(k,n) \sum_{\nu=0}^{\mu-1} [g_{k+n,\nu}]$ (c) $[g_{k+n,\mu}] := \frac{[z]}{P(k,n)}$

For fixed k , the $[g_{kl}]$ are computed according to the above Algorithm 2. It is most important that the computation of $[z]$ in Step 2(b) of the algorithm is performed as a precise scalar product. This can be done if the products $P(k,n)$ and $P(k-j,i) b_{ij} h^{n-i+j}$ are stored exactly on the computer in one or more

machine numbers, using the staggered correction technique. If the polynomial coefficients b_{ij} , b_k or the argument h are not exactly representable as a finite sum of machine numbers, they must be enclosed in machine intervals.

The computation of the summands $a_k h^k$ according to Algorithm 1 can be started with $L = 0$, i.e. single precision. When computing subsequent summands, L is incremented whenever a loss in accuracy of $a_k h^k$ is observed during the computation. In our numerical examples, L was increased whenever the accuracy of the next summand $a_k h^k$ in the iteration, or of $s(k+n)$ in Step 4(c), fell below single precision. In Section 3 it has been shown that with real arithmetic, Algorithm 1 terminates after finitely many iterations for any $\omega \in (0, 1)$, because the bound $t(\kappa)$ for the truncation error $r(k)$ tends to zero. The same is true for practical computations, provided that reasonable error bounds ε_{abs} and ε_{rel} are used (reasonable means, that ε_{abs} must be at least twice as large as the smallest positive machine representable number), that no overflow occurs (the absolute values of the numbers $a_k (\frac{h}{\omega})^k$ must not exceed the greatest representable number on the computer), and that roundoff errors of any arithmetic operation only affect the last digit of the mantissa. $t(\kappa)$ tends to zero for $\kappa \rightarrow \infty$, so under the above conditions, even when computed with roundoff errors, it becomes smaller than ε_{abs} . The accuracy of the bounds for $y(h)$ then depends on the diameter of the interval sum $\sum_{k,l} [g_{kl}] \ni s(\kappa+n)$ in Step 4(c) of Algorithm 1.

If the diameters of the enclosing intervals of the summands $a_k h^k$ grow due to error propagation and thus make the computation of a tight enclosure of $s(\kappa+n)$ impossible, then one must split the integration domain into subintervals, like in the case of overflow.

Splitting of the integration domain

In our numerical examples, the recess condition (9) was used to bisect the interval of integration recursively. Starting with $[0, h]$, the recess condition was tested for $\omega = 1$ and suitable values of $k = \kappa$. This test was performed before executing any other step of Algorithm 1. If (9) was fulfilled, then Algorithm 1 was carried out, otherwise the interval was bisected and the same test was applied to both subintervals.

It was already mentioned in Section 6, that after splitting the integration domain $[0, h]$ into J subintervals $I_j = [x_{j-1}, x_j]$, $j = 1, \dots, J$, then in each subinterval I_j , $j > 1$, interval initial values appear. Following the arguments of Section 6, on each subinterval, real initial value problems should be used to compute tight enclosures of $y(x_j)$.

In the case of a homogeneous differential equation, n initial value problems

$$u_\nu^{(n)} = \sum_{i=0}^{n-1} p_i(x) u_\nu^{(i)}, \quad x \in I_j,$$

$$u_\nu^{(\mu)}(x_{j-1}) = \delta_{\mu\nu}, \quad \mu, \nu = 0, 1, \dots, n-1,$$

replace one interval initial value problem. If interval matrices $[A_j] = ([a_{\mu\nu}^j])$ are built of intervals $[a_{\mu\nu}^j] \ni u_\nu^{(\mu)}(x_j)$ for $j = 1, \dots, J$ then $y(h)$ is contained in the first component of the matrices-vector product

$$[A_J]B_J \cdot \left(B_J^{-1}[A_{J-1}]B_{J-1} \cdot \left(\dots \left(B_3^{-1}[A_2]B_2 \left(B_2^{-1}[A_1] \cdot \begin{pmatrix} y^{(0)} \\ \vdots \\ y^{(n-1)}(0) \end{pmatrix} \dots \right) \right) \right), \quad (16)$$

where the B_j are arbitrary nonsingular matrices. Using $n+1$ ivps on each subinterval, a similar representation can be derived for nonhomogeneous ODEs.

The matrices B_j are used to rule out the wrapping effect that appears when intermediate results of successive matrix-vector multiplications are stored in interval vectors. Lohner [12, 13, 14] discussed several choices for the matrices B_j . In our numerical examples, we used $B_j = \text{mid}([A_{j-1}]B_{j-1})$. Lohner called this the "parallel epiped evaluation" of (16).

Choice of ω

For an early regression of the $c_k = a_k \left(\frac{h}{\omega}\right)^k$, ω should be chosen near 1. On the other hand, for the smallest $k =: k_{\min}$ that fulfills the recess condition (9) for a particular value of $\omega \approx 1$, the error bound $t(k+n)$ in Step 4(c) of Algorithm 1 is typically very large. Most often, the iteration terminates not before $k \gg k_{\min}$.

For a fixed $k > k_{\min}$, $t(k+n)$ can be regarded as a function of ω , and instead of $\omega \approx 1$ one had better use ω such that $t(k+n)$ becomes minimal, provided that (9) still holds for k and ω .

Substituting c_ν by $a_\nu \left(\frac{h}{\omega}\right)^\nu$ in (13), we obtain

$$C = \max_{\nu=-m}^{n-1} |a_{\kappa+\nu} \left(\frac{h}{\omega}\right)^{\kappa+\nu}|.$$

If (9) holds for $k = \kappa$ and ω , instead of (12) we have

$$|y(h) - s(\kappa+n)| \leq t(\kappa+n) = \max_{\nu=-m}^{n-1} |a_{\kappa+\nu} h^{\kappa+\nu}| \frac{1}{\omega^\nu (1-\omega)}. \quad (17)$$

If $\nu \leq 0$ then

$$|a_{\kappa+\nu}h^{\kappa+\nu}| \frac{1}{\omega^\nu(1-\omega)} \quad (18)$$

is monotonically increasing with $\omega \in (0, 1)$. If $\nu > 0$ then the minimum of (18) occurs at $\omega = \frac{\nu}{\nu+1}$. However, since $\{a_k h^k\}$ is a null sequence it is reasonable to assume that the maximum in (17) usually occurs for some $\nu \leq 0$. To minimize $t(\kappa + n)$, ω should then be chosen as small as possible, if only (9) still holds.

In our computer program, ω is updated in every iteration step so that (9) remains barely fulfilled. As soon as $k \geq mn$ is large enough to ensure that

$$\sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) |b_{ij}| h^{n-i+j}}{P(k, n)} < 0.98,$$

ω is determined from

$$\sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) |b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n)} \approx 0.98, \quad (19)$$

where the latter is maintained by a Newton correction of ω whenever k is incremented (observe that ω defined from (19) is a strictly decreasing function of k). This scheme worked very well in practice.

8 Numerical Examples

The numerical examples were computed on two PCs, one with a 233 MHz Pentium II processor and a Windows 95 operating system, and one with a 166 MHz Pentium processor and a Linux operating system. The programming was done in PASCAL-XSC with the standard IEEE arithmetic of 16 decimal digits. The numerical results on both computers were identical.

The results of our computer program according to Algorithm 1 (performed with the staggered correction arithmetic as described in Algorithm 2) are compared with results computed with Lohner's program AWA [12, 13, 14]. The comparison is a little inappropriate, because AWA is a general purpose algorithm that can handle almost any differential equation (as long as all defining functions are smooth enough) and does not make use of the particular structure of the given differential equation, like our method does. On the other hand, the comparison shows that even though AWA is a comprehensive and reliable program, it may be worth looking for alternative enclosure methods in special cases.

In the implementation of our method we focused on accuracy, not speed. Due to the data structure used in the computer program, the numbers $a_k h^k$ are

stored with at least double precision. Hence, in all of our examples, the results of our program are more accurate than the results computed with AWA. For some problems (see Example 1), large step sizes (and, consequently, multiple precision data formats) appear the only means to obtain enclosures at all.

Comparing the computation times is difficult, because the performance of both programs is very sensitive to choices of parameters like the order of the Taylor expansion in AWA, or the number κ that is used in the bisection test of the integration domain in our method. For a simple problem like Example 3, AWA seems to work faster than our program. However, considering that the enclosures of Example 3 with our program were computed with double precision and are much more accurate than the enclosures that were computed with AWA, the gain of AWA in computation time is not impressive.

A single integration step of our method is only expensive if many of the summands $a_k h^k$ must be computed with high precision. In this case, a single large integration step may replace hundreds of small integration steps that are enforced when using a single precision data format, and our method can be superior with respect to not only accuracy but also speed (as can be observed in Example 2).

For the computations with our method, we list the values of the computed enclosures and the values of L in the staggered correction format and of κ in the partial sum $s(\kappa)$, for which the enclosures were gained. Other parameters like ε_{abs} are only shown when appropriate. In all examples, $\varepsilon_{\text{rel}} = 1.0\text{E-}16$ was used.

With AWA, we computed the intersection of "interval evaluation" and "QR decomposition" (see [12, 13, 14]), which usually gives the most accurate results. Besides the enclosures of the solution, the respective degrees of the Taylor expansions and the number of integration steps are presented.

We also show the computation times (in seconds) that were measured with the Windows 95 system. Similar relations of the respective computation times were also observed with the Linux computer.

The complete code of our computer program for Algorithm 1 is available on the internet from

<http://www.uni-karlsruhe.de/~ae16/pivp.html>

Example 1: $y'' = y$, $y(0) = 1$, $y'(0) = -1$.

Exact Solution: $y(x) = e^{-x}$.

The purpose of this example is to trace the decaying solution of $y'' = y$ over a long range. Table I shows that even for values of h as large as 300, 16 decimal digits of accuracy in the result could be reached with our method. All enclosures were obtained with only one integration step.

TABLE I
Alternative runs of Algorithm 1 with different integration domains

h	$[y](h)$	L	κ	Time
10	4.539 992 976 248 $48\frac{7}{4}$ E-005	2	59	0.05
15	3.059 023 205 018 $25\frac{9}{7}$ E-007	2	78	0.06
20	2.061 153 622 438 $55\frac{9}{7}$ E-009	3	97	0.11
40	4.248 354 255 291 $5\frac{91}{88}$ E-018	4	170	0.39
100	3.720 075 976 020 $83\frac{7}{5}$ E-044	7	386	2.47
200	1.383 896 526 736 $73\frac{8}{7}$ E-087	12	745	12.74
300	5.148 200 222 412 $01\frac{6}{2}$ E-131	18	1104	37.19

With AWA, tight enclosures of $y(h)$ can only be computed for small values of h . When h becomes larger, the computed enclosures widen, until the method finally breaks down. In Table II, results that were obtained for a degree of the Taylor expansion of 20 are given. Using higher degrees (even up to 50) yielded longer computation times, but not better enclosures.

TABLE II
Alternative runs of AWA

h	$[y](h)$	Degree	No. of steps	Time
10	4.539 99 $\frac{32}{28}$ E-05	20	33	0.11
15	3.0 $\frac{61}{57}$ E-07	20	50	0.17
20	$\frac{3.136}{-2.724}$ E-08	20	69	0.28
40	$\frac{-1.422}{-1.422}$ E+01	20	101	0.39
46.241...	$\frac{-7.299}{-7.299}$ E+03	20	110	0.44

Despite the simplicity of the differential equation, the problem appears to be delicate. Lohner's algorithm fails because the first integration step of his method supplies interval initial values for the second and subsequent integration steps. These interval initial value problems also contain solutions of $y'' = y$ that do not decay but grow exponentially. Since these solutions must be enclosed together with e^{-x} in subsequent integration steps, the diameters of the computed enclosures grow until numerical overflow on the computer occurs.

Example 2:
$$y^{(4)} = (x^2 + 10x + 26) y^{(3)} + (-20x - 99.5) y''$$

$$+ (x^2 + 10x + 25) y' + (-2x^2 - 4x + 29.5) y$$

$$y(0) = 5, y'(0) = 4, y''(0) = 3, y^{(3)}(0) = 2.$$

Exact solution: $y(x) = (5 - x) e^x$.

The solution of this example has a zero at $x = 5$, which can be detected nicely with our method. As is also indicated in Table II, for achieving higher precision in the resulting enclosure from Algorithm 1, it is not necessary to increase the number of summands in $s(\kappa)$ but to raise L and perform all computations with higher precision.

TABLE III
Alternative runs of Algorithm 1 with different error bounds

h	ε_{abs}	$[y](h)$	L	κ	Time
1	—	1.087 312 731 383 61 ₇ ⁹ E+01	2	58	0.11
1.25	—	1.308 878 609 048 19 ₀ ¹ E+01	2	72	0.17
1.5	—	1.568 591 174 618 32 ₂ ³ E+01	2	88	0.22
4	—	5.459 815 003 314 42 ₂ ⁶ E+01	7	364	2.25
5	1.0E-050	$\begin{matrix} 4.951 \\ -4.951 \end{matrix}$ E-065	12	548	5.28
5	1.0E-100	$\begin{matrix} 4.945 \\ -4.945 \end{matrix}$ E-114	15	548	6.70
5	1.0E-150	$\begin{matrix} 1.890 \\ -1.890 \end{matrix}$ E-162	18	548	8.74

Like in Example 1, AWA breaks down early. Also, the computation times for our method are much better here.

TABLE IV
Alternative runs of AWA

h	$[y](h)$	Degree	No. of steps	Time
1	1.087 312 $\frac{78}{68}$ E+01	15	100	2.80
1	1.087 312 $\frac{77}{68}$ E+01	20	61	2.42
1	1.087 312 $\frac{78}{67}$ E+01	30	55	3.74
1.25	1.30 $\frac{91}{86}$ E+01	20	81	3.13
1.5	$\begin{matrix} 34.960 \\ -6.183 \end{matrix}$	20	101	3.95

Example 3: $y'' = -xy$, $y(0) = 1$, $y'(0) = 0$.

Exact Solution: Linear combination of Airy functions $\text{Ai}(-x)$ and $\text{Bi}(-x)$.

The Airy functions $\text{Ai}(x)$ and $\text{Bi}(x)$ are special solutions of the differential equation $y'' = xy$ (see [2] for the definition and general properties of Airy functions). Like $\text{Ai}(-x)$ and $\text{Bi}(-x)$, the solution of the above initial value

problem oscillates more rapidly with increasing x . To make Algorithm 1 applicable, the integration domain had to be split into subintervals, as worked out in Section 7. We show the respective values of κ_{test} , that were used in the bisection test of the recess condition.

TABLE V
Alternative runs of Algorithm 1 with different integration domains

h	κ_{test}	$[y](h)$	No. of steps	L_{max}	κ_{max}	Time
100	30	2.686 659 923 58 $\frac{812}{799}$ E-01	32	2	110	8.90
100	100	2.686 659 923 588 0 $\frac{84}{33}$ E-01	11	4	292	16.31
500	30	2.523 976 408 6 $\frac{738}{663}$ E-02	376	2	112	100.02
500	100	2.523 976 408 6 $\frac{712}{689}$ E-02	106	4	294	197.08
1000	30	1.112 457 368 6 $\frac{667}{514}$ E-02	946	2	115	281.50
1000	100	1.112 457 368 6 $\frac{615}{567}$ E-02	323	4	305	529.81

For this well-behaved IVP, the solution can also be traced over a large integration domain with AWA. Since our program uses at least double precision, our results are more precise than those of AWA. On the other hand, the optimal degree of the Taylor expansion in AWA yields faster results. However, this is only true if the expansion degree is chosen almost optimal. Otherwise the computations simultaneously lose precision and last much longer.

TABLE VI
Alternative runs of AWA

h	$[y](h)$	Degree	No. of steps	Time
100	2.686 659 9 $\frac{29}{18}$ E-01	20	1036	7.42
500	2.523 976 $\frac{461}{356}$ E-02	15	16568	87.28
500	2.523 976 $\frac{466}{351}$ E-02	20	9845	70.69
500	2.523 976 $\frac{534}{283}$ E-02	22	24863	196.85
500	2.523 976 $\frac{635}{183}$ E-02	25	57354	532.78
1000	1.112 457 $\frac{453}{284}$ E-02	20	27197	194.55
1000	1.112 45 $\frac{802}{672}$ E-02	22	68246	544.86

Example 4: Computation of eigenvalues of

$$\begin{aligned} -u'' + x^2 u &= \lambda u \\ u(-1) = u(1) &= 0. \end{aligned} \tag{20}$$

A real number λ is called an eigenvalue of the boundary value problem (20) if there is a nontrivial solution $u(x)$ of (20). As is well known, the eigenvalues

of (20) form an infinite sequence of real numbers which is bounded from below and tends to infinity (cf. [8], Chap. V).

Following the well-known Sturm–Liouville theory, we compute eigenvalues of (20) with a shooting method with shooting parameter λ , using the initial value problem

$$\begin{aligned} y'' &= (x^2 - \lambda)y \\ y(-1) &= 0, \quad y'(-1) = 1. \end{aligned} \tag{21}$$

For $\lambda \in \mathbb{R}$ fixed, let $y(\cdot; \lambda)$ denote the solution of (21). If $y(\cdot; \lambda)$ has $\varrho - 1$ zeros within $(-1, 1)$ then λ is a lower bound of the ϱ -th eigenvalue of (20) and an upper bound of its $\varrho - 1$ st eigenvalue. λ is the ϱ -th eigenvalue of (20), if additionally $y(1; \lambda) = 0$ holds.

TABLE VII
Four alternative runs of Algorithm 1 with different λ

λ	$N(\lambda)$	$[y](1)$	L	κ	Time
39.799 393 003 660 17	3	-1.669 607 841 203 6 $_{20}^{19}$ E-16	3	94	0.22
39.799 393 003 660 18	4	1.449 994 256 255 94 $_{5}^7$ E-17	3	96	0.16
247.071 500 228 031 8	9	-1.112 981 851 908 2 $_{61}^{59}$ E-16	3	144	0.33
247.071 500 228 031 9	10	3.500 771 860 671 2 $_{63}^{66}$ E-16	3	144	0.33

Lower and upper eigenvalue bounds for specific eigenvalues are computed by solving (21) for different values of λ and $x \in [-1, 1]$, and by determining the number $N(\lambda)$ of zeros of the respective solutions $y(\cdot; \lambda)$ within $(-1, 1)$ (see [19] for the computation of suitable starting values λ and for the reliable determination of $N(\lambda)$). Eigenvalue bounds can then be refined by bisection with respect to λ (faster algorithms were also given in [19]).

In Table VII, the enclosures of $y(1; \lambda)$ from (21) are listed for four different values of λ . From these enclosures, the eigenvalues λ_4 and λ_{10} of (20) are determined with 16 decimal digits of accuracy:

$$\begin{aligned} \lambda_4 &\in 39.799\ 393\ 003\ 660\ 1\frac{8}{7}, \\ \lambda_{10} &\in 247.071\ 500\ 228\ 031\ \frac{9}{8}. \end{aligned}$$

With AWA, the enclosures are slightly worse. Of course, this is again due to the fact that our program runs with multiple precision.

TABLE VIII
Alternative runs of AWA

λ	$[y](1)$	Degree	No. of steps	Time
247.071 500 228 026 5	$-5.27\text{E-}17$ $-4.33\text{E-}14$	20	44	0.38
247.071 500 228 026 6	$3.80\text{E-}15$ $-4.61\text{E-}14$	20	44	0.38
247.071 500 228 037 2	$4.87\text{E-}14$ $-4.91\text{E-}15$	20	44	0.39
247.071 500 228 037 3	$4.38\text{E-}14$ $6.15\text{E-}16$	20	44	0.38

Since zero is contained in the enclosures of $y(1)$ for $\lambda = 247.0715002280266$ or $\lambda = 247.0715002280372$ it is impossible to decide if these numbers are lower or upper eigenvalue bounds. It can only be guaranteed that

$$\lambda_{10} \in 247.071\ 500\ 228\ 0_{265}^{373}.$$

9 Conclusion

We have presented a new enclosure method for ODEs of a special type. Our numerical examples demonstrate that the method can be successfully implemented on a computer. The comparison with AWA shows, that in some examples, our specialized method succeeds when the comprehensive algorithm AWA fails.

As a final remark, we mention that our approach is not restricted to the linear differential equations with polynomial coefficients. With some modifications, it can also be applied to linear differential equations with analytical coefficients and even to some systems of nonlinear differential equations. These extensions will be the subject of future research.

Acknowledgment: The author thanks the referees for helpful comments.

Appendix: Proof of the Lemmas in Section 2

Proof of Lemma 1: From the definition of $P(k, i)$, for $0 \leq i \leq n - 1$ and

$0 \leq j \leq m$,

$$\begin{aligned}
\frac{P(k+1-j, i)}{P(k+1, n)} &= \frac{P(k-j, i)}{P(k, n)} \cdot \frac{(k+1-j+i)(k+1)}{(k+1-j)(k+1+n)} \\
&= \frac{P(k-j, i)}{P(k, n)} \cdot \frac{(k+1-j+i)(k+1)}{(k+1-j+i)(k+1) + (n-i)(k+1) - nj} \\
&\leq \frac{P(k-j, i)}{P(k, n)} \cdot \frac{(k+1-j+i)(k+1)}{(k+1-j+i)(k+1) + k+1 - nm} \\
&< \frac{P(k-j, i)}{P(k, n)} \quad \text{for } k \geq nm.
\end{aligned}$$

The positive numbers $|b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}$ are independent of k , so that each summand in (8) is a strictly decreasing function of k . \square

Proof of Lemma 2: $P(k-j, i)$ is a polynomial of degree i in k , $P(k, n)$ is a polynomial of degree n in k . As in Lemma 1, the numbers $|b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}$ are independent of k so that

$$\lim_{k \rightarrow \infty} \left(\sum_{i=0}^{n-1} \sum_{j=0}^m \frac{P(k-j, i) |b_{ij}| \left(\frac{h}{\omega}\right)^{n-i+j}}{P(k, n)} \right) = 0.$$

\square

References

- [1] G. Alefeld and J. Herzberger, "Introduction to Interval Computations", Academic Press, New York, 1983.
- [2] Abramowitz, M.; Stegun, I. A.: "Handbook of Mathematical Functions", Dover Publ. Inc., New York, 1972.
- [3] G. Birkhoff and G.-C. Rota, "Ordinary Differential Equations", Ginn and Company, Boston, 1962.
- [4] L. Collatz, "The Numerical Treatment of Differential Equations", Springer, Berlin, 1959.
- [5] G. F. Corliss, Survey of Interval Algorithms for Ordinary Differential Equations, *Appl. Math. Comput.* **31** (1989), 112 – 120.

- [6] G. F. Corliss, Guaranteed Error Bounds for Ordinary Differential Equations, *in* “Theory and Numerics of Ordinary and Partial Differential Equations” (M. Ainsworth, J. Levesley, W. A. Light, and M. Marletta, Eds.), pp. 1–75, Clarendon Press, Oxford, 1995.
- [7] G. F. Corliss and R. Rihm, Validating an A Priori Enclosure Using High-Order Taylor Series, *in* “Scientific Computing and Validated Numerics” (G. Alefeld, A. Frommer, and B. Lang, Eds.), pp. 228 – 238, Akademie-Verlag, Berlin, 1996.
- [8] R. Courant and D. Hilbert, “Methods of Mathematical Physics, Vol. 1”, Interscience Publishers, New York, 1953.
- [9] R. Klatte, U. Kulisch, M. Neaga, D. Ratz, and Ch. Ullrich, “PASCAL–XSC — Language Reference with Examples”, Springer, Berlin, 1992.
- [10] R. Klatte, U. Kulisch, Ch. Lawo, M. Rauch, and A. Wiethoff, “C–XSC, A C++ Class Library for Extended Scientific Computing”, Springer, Berlin, 1993.
- [11] U. Kulisch and W. L. Miranker, “Computer Arithmetic in Theory and Practice”, Academic Press, New York, 1981.
- [12] R. Lohner, Enclosing the Solutions of Ordinary Initial- and Boundary-Value Problems, *in* “Computer Arithmetic: Scientific Computation and Programming Languages” (E. Kaucher, U. Kulisch, and Ch. Ullrich, Eds.), pp. 255–286, Teubner, Stuttgart, 1987.
- [13] R. Lohner, “Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen”, Dissertation, Universität Karlsruhe, 1988.
- [14] R. Lohner, Computation of Guaranteed Solutions of Ordinary Initial and Boundary Value Problems, *in* “Computational Ordinary Differential Equations” (J. R. Cash and I. Gladwell, Eds.), pp. 425–435, Clarendon Press, Oxford, 1992.
- [15] R. Lohner, Interval Arithmetic in Staggered Correction Format, *in* “Scientific Computing with Automatic Result Verification” (E. Adams and U. Kulisch, Eds.), pp. 301–321, Academic Press, San Diego, 1993.
- [16] R. E. Moore, The Automatic Analysis and Control of Error in Digital Computation Based on the Use of Interval Numbers, *in* “Error in Digital Computation, Vol. I” (L. B. Rall, Ed.), pp. 61–130, John Wiley and Sons, New York, 1965.
- [17] R. E. Moore, Automatic Local Coordinate Transformations to Reduce the Growth of Error Bounds in Interval Computation of Solutions of Ordinary Differential Equations, *in* “Error in Digital Computation, Vol. II” (L. B. Rall, Ed.), pp. 103–140, John Wiley and Sons, New York, 1965.

- [18] R. E. Moore, “Interval Analysis”, Prentice Hall, Englewood Cliffs, N.J., 1966.
- [19] M. Neher, Inclusion of Eigenvalues and Eigenfunctions of the Sturm-Liouville Problem, *in* “Computer Arithmetic and Enclosure Methods” (L. Atanassova, J. Herzberger, Eds.), pp. 401–408, Elsevier, Amsterdam, 1992.
- [20] R. Rihm, Interval Methods for Initial Value Problems in ODES, *in* “Topics in Validated Computations” (J. Herzberger, Ed.), pp. 173–207, Elsevier, Amsterdam, 1994.
- [21] J. Schröder, Funktionalanalytische Herleitung von Fehlerabschätzungen und ihre praktische Durchführung auf Rechenanlagen, *ZAMM* **40** (1960), T27 – T37.
- [22] J. Schröder, Fehlerabschätzungen mit Rechenanlagen bei gewöhnlichen Differentialgleichungen erster Ordnung, *Numer. Math.* **3** (1961), 39 – 61.
- [23] H. J. Stetter, Sequential Defect Correction for High-Accuracy Floating-Point Arithmetic, *in* “Numerical Analysis”, pp. 186–202, Lecture Notes in Mathematics, Vol. 1066, Springer, Berlin, 1984.
- [24] H. J. Stetter, Validated Solution of Initial Value Problems for ODE, *in* “Computer Arithmetic and Self-Validating Numerical Methods” (Ch. Ullrich, Ed.), pp. 171–186, Academic Press, San Diego, 1990.
- [25] W. Walter, “Differential and Integral Inequalities”, Springer, Heidelberg, 1970.
- [26] W. V. Walter, FORTRAN–XSC: A Portable Fortran 90 Module Library for Accurate and Reliable Scientific Computing, *in* “Validation Numerics – Theory and Applications” (R. Albrecht, G. Alefeld, and H. J. Stetter, Eds.), pp. 265–285, Computing Supplementum, Vol. 9, Springer, Wien, 1993.