IMM
DEPARTMENT OF MATHEMATICAL MODELLING

Technical University of Denmark
DK-2800 Lyngby – Denmark

J. No. 1996-x5-94
August 19, 1996
OS

ENCLOSING SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS

with applications

Ole Stauning

TECHNICAL REPORT

IMM-REP-1996-18

IMM

Abstract

This text describes the method used in the program AWA (German: AnfangsWertAufgaben) which was developed by Rudolf Lohner. The program is capable of solving ordinary differential equations, obtaining verified bounds of the true solution. Picard iterations are used to prove existence and to find a rough enclosure of the solution, this rough enclosure are then sharpened by Taylor expansion. Using a mean value method and Taylor expansion of the variational equation, global errors are propagated. A program package, DTU AWA, has been developed in C++ using the interval packages PROFIL/BIAS [7, 8]. The program uses basically the same method as in the original AWA, but in DTU AWA a precompiler is used to generate the recursive code which is used in the integration to calculate the Taylor coefficients.

I was very lucky to be given the opportunity to work on this report at the Institut Für Angewandte Mathematik, Karlsruhe. I have visited the institute for half a year, and it has been a very good experience for me. I would like to thank my colleagues at the institute for their hospitality. I would especially like to thank Rudolf Lohner, who is the author of AWA, for helping me understanding his method. And for our many discussions about interval analysis.

Contents

1 Introduction

Differential equations seem to be an important tool in theoretical physics. Unfortunately, when these differential equations become a little complex, usually no method for obtaining the precise solutions exist. Instead we turn to approximative solution methods. The computer is a very important tool in this field, and a lot of effort goes into inventing even faster and more precise algorithms to approximate the solution. A common thing about most of these approximation algorithms is however, that they do NOT guarantee that the solution exists in some specified error bounds. This problem becomes even worse when the speed of computers continue to increase. It is possible to approximate solutions to even more complex systems, but it has also become more difficult or even impossible to verify if these approximations had anything to do with the true model, and not just some artifact from the method which is used to do the approximation [11].

A major step in making the computer a more reliable tool was made in the sixties by Moore when he published the book “Interval Analysis” [12]. In interval analysis, intervals are used instead of real numbers. The arithmetic on these intervals is designed so that the result of an operation on intervals encloses all results from the corresponding real operation on all combinations of real numbers in these intervals. When interval arithmetics are applied on a computer, the rounding of every elementary arithmetic operation is directed outwards, so that the intervals always encloses the true solution, and of course the rounding error. This way, when an algorithm is running on a computer, an enclosure of the true result plus the propagation of all intermediate round-off errors is enclosed.

We have basically two reasons to use interval arithmetic in solving ordinary differential equations.

- The first, and most obvious reason to use interval arithmetic, is when variables with uncertain values in some known interval are used in calculations. This is the case in computer arithmetic, where the real numbers are discretized to fit a certain amount of bits. Here the rounding is directed to obtain a maximum and a minimum value in each elementary operation. Another case is when we have some uncertainty in a physics variable, due to some measurement error, and this value is used as input parameters in a computer program.

- Also the ability to perform calculations with sets of real numbers is of great importance. Many interval algorithms uses set theoretic fixed point theorems [4, 10], to prove the correctness of the results they produce. These algorithms are sometimes called self validating algorithms.

2 Interval arithmetic

2.1 The basic interval operations

Let $a, b \in \mathbb{R}$ be real numbers so that $a \leq b$. The closed interval from a to b is denoted by

$$[a,b] = \{x \mid a \leq x \leq b\}. \tag{1}$$

The set of intervals in $\mathbb{R}$ is denoted by $\mathbb{IR}$. Let $[x] \in \mathbb{IR}$ be an interval[1]. Then the lower endpoint of $[x]$, also called the infimum will be written $\underline{x}$ and the upper endpoint, the supremum, will be written $\overline{x}$. I.e. we have $[x] = [\underline{x}, \overline{x}]$.

Let $[x], [y] \in \mathbb{IR}$ be intervals. The elementary operations on $\mathbb{IR}$ is defined by [13]

$$\begin{aligned}
[x] + [y] &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}], && \text{(2a)}\\
[x] - [y] &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}], && \text{(2b)}\\
1/[x] &= [1/\overline{x}, 1/\underline{x}] \quad \text{if } 0 \notin [x], && \text{(2c)}\\
[x] \cdot [y] &= [\min(\underline{x}\,\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\,\overline{y}), && \text{(2d)}\\
&\qquad \max(\underline{x}\,\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\,\overline{y})].
\end{aligned}$$

The basic idea of these arithmetic rules is that for all combinations of $x \in [x]$ and $y \in [y]$ we have $x \circ y \in [x] \bigcirc [y]$, where $\bigcirc$ is one of the above interval operations and $\circ$ is the corresponding real operation. Also the usual functions, such as $\cos, \sin$ etc. can be extended to interval operations. We will in many cases omit specifying if a function is a real or an interval valued function and let it be determined by its argument(s), if some of the arguments are intervals then the function must be an interval valued function.

The intersection of two intervals $[x]$ and $[y]$ are given by

$$[x] \cap [y] = \begin{cases} \emptyset, & \text{if } \overline{x} < \underline{y} \text{ or } \overline{y} < \underline{x}, \\ [\max\{\underline{x}, \underline{y}\}, \min\{\overline{x}, \overline{y}\}], & \text{otherwise.} \end{cases} \tag{3}$$

Since the empty set is not an interval, special care has to be taken when $[x] \cap [y] = \emptyset$. The transitive, partial ordering relation $\subseteq$ in $\mathbb{IR}$ is defined by

$$[x] \subseteq [y] \quad \text{if and only if} \quad \underline{y} \leq \underline{x} \text{ and } \overline{y} \geq \overline{x}. \tag{4}$$

[1] We will use the $[\cdot]$ notation to denote intervals.

Interval vectors and interval matrices are defined from intervals in a similar way that real vectors and matrices are defined from real numbers. The n dimensional interval vector space is denoted by $\mathbb{IR}^n$. If $D \subseteq \mathbb{R}^n$ is a subset of the n dimensional real space then by $\mathbb{I}D$ we will denote the set of all interval vectors in D.

A very well known problem in interval analysis is the problem of overestimation. Overestimation occurs when an interval are used two or more times in a calculation. In all interval operations, the interval operands are considered to be independent of each other, hence the interval operations encloses the real operations with all combinations of real operands in their corresponding interval operands without considering dependencies of the two arguments. This overestimation often is referred to as the *wrapping effect* [9, 13] when a sequence of points in $\mathbb{IR}^n$, $n \geq 2$ is generated from some interval iteration $[x_{i+1}] = f([x_i])$. All iterates $[x_i]$ are bound to be interval vectors, so components of $[x_i]$ are considered independent of each other.

2.2 The mean value form

One way to reduce overestimation in the evaluation of a function $f \in C^1(\mathbb{R}^m, \mathbb{R}^n)$ in a set of real numbers $x \in [x] \in \mathbb{R}^m$, is by using the mean value form.

Theorem 1 *Let $f \in C^1(D, \mathbb{R}^n)$ where $D \subseteq \mathbb{R}^m$ is an open set. For $[x] \in \mathbb{I}D$ we will have $f(x) - f(y) \in f'([x])(x-y)$ for all $x, y \in [x]$.*

Proof

Let $x, y \in [x] \in \mathbb{I}D$. Consider the ith coordinate function $f_i \in C^1(D, \mathbb{R})$. A point ξ_i on the straight line between x and y is known from the mean value theorem to exist, so that

$$f_i(x) - f_i(y) = \sum_{j=1}^{m} \frac{\partial f_i}{\partial x_j}(\xi_i)(x_j - y_j). \tag{5}$$

As $[x]$ is a convex set, the straight line between x and y must lie in $[x]$ and

$$f_i(x) - f_i(y) \in \nabla f_i([x])(x-y), \tag{6}$$

which is true for every f_i, $i = 1, \ldots, m$.

$\square$

We now have that $f(x) \in f(y) + f'([x])([x] - y)$, where $x, y \in [x]$. For small widths of $[x]$ this function often provides a more tight enclosure than the usual interval extension of f. Normally y is chosen to be the midpoint of $[x]$.

3 Enclosing solutions of initial value problems

The method for solving initial value problems, which is described in this report, is due to R. Lohner [9, 10]. For a survey on interval methods for solving ordinary differential equations see [3, 9].

3.1 Error propagation using the mean value form

As a non-autonomous ode can be transformed into an autonomous ode, by introducing a time variable in the phase space, we will w.l.o.g. only consider the autonomous initial value problem

$$u' = f(u), \quad u(0) = u_0, \tag{7}$$

where $u(t) \in \mathbb{R}^n$, and $f \in C^p(D,\mathbb{R}^n)$, $D \subset \mathbb{R}^n$ is an open set. The usual way to solve initial value problems is by discretizing it to a discrete map $\tilde{u}_{j+1} = \kappa(\tilde{u}_j, h_j)$, $j \geq 0$, and $\tilde{u}_0 = u_0$, here h_j is the discretization of the independent integration variable t, usually called the *integration time*. $\tilde{u}_j$ will then be an approximation to the real solution $u(t_j)$, where $t_j = \sum_{i=0}^{j-1} h_i$. The problem using this method is, that the introduced errors when discretizing the problem usually are hard to estimate. Even if the error in each integration step can be bounded, it usually is very hard to estimate the effect when this local error is propagated through the iterative integration process. Assume that the local discretization error z_{j+1} is known, so that

$$u_{j+1} = \kappa(u_j, h_j) + z_{j+1}, \quad j \geq 0 \tag{8}$$

is the exact solution, i.e. $u_{j+1} = u(t)$, to the initial value problem (7), at $t = t_{j+1}$, with the initial value $u(0) = u_0 + z_0$. The value of z_0 can be interpreted as an error in the initial value. Assume that $z_0, z_1, \ldots z_{j+1}$ are independent. u_{j+1} can now be considered as a function of the local intermediate errors

$$u_{j+1} = u_{j+1}(z_0, z_1, \ldots, z_{j+1}). \tag{9}$$

If $\kappa(u,h)$ is differentiable with respect to the state variables u, we have from Eq. (8), that

$$\frac{\partial u_{j+1}}{\partial z_k} = \begin{cases} \kappa'(u_j, h_j)\dfrac{\partial u_j}{\partial z_k} & ,k \leq j, \\ I & ,k = j+1. \end{cases} \tag{10}$$

The discretization error z_{j+1} is however not known exactly. Assume instead that an enclosure $[z_{j+1}] \ni z_{j+1}$ of the discretization error can be calculated. The problem is to find a narrow enclosure $[u_{j+1}]$ of the true solution u_{j+1} using an iteration of the form

$$u_{j+1} \in \kappa([u_j],h_j) + [z_{j+1}], \quad j \geq 0 \tag{11}$$

The interval extension of Eq. (10) is

$$\left[\frac{\partial u_{j+1}}{\partial z_k}\right] = \begin{cases} \kappa'([u_j],h_j)\left[\frac{\partial u_j}{\partial z_k}\right] & ,k \leq j, \\ I & ,k = j+1. \end{cases} \tag{12}$$

Let $[Z_{j+1}] = ([z_0],[z_1],...,[z_{j+1}])$. Define $[u_{j+1}]$ to be the mean value form of the right hand side of (11), i.e.

$$[u_{j+1}] = \hat{u}_{j+1} + \sum_{k=0}^{j+1}\left[\frac{\partial u_{j+1}}{\partial z_k}\right]([Z_{j+1}])\,([z_k]-s_k)$$

where $\hat{u}_{j+1} = \hat{u}_{j+1}(s_0,s_1,...,s_{j+1})$ for $j \geq 0$, $\hat{u}_0 = u_0 + s_0$ and $s_k \in [z_k]$. We now have

$$\hat{u}_{j+1} = \kappa(\hat{u}_j,h_j) + s_{j+1} \in \kappa(\hat{u}_j,h_j) + [z_{j+1}]. \tag{13}$$

Define $[\hat{u}_{j+1}] = \kappa(\hat{u}_j,h_j) + [z_{j+1}]$ for $j \geq 0$ and $[\hat{u}_0] = u_0 + [z_0]$. Since $\kappa(\hat{u}_j,h_j)$ is a real vector we have that

$$[\hat{u}_k] - \hat{u}_k = [z_k] - s_k. \tag{14}$$

The restriction $s_k \in [z_k]$ from (13) is fulfilled when $\hat{u}_k \in [\hat{u}_k]$. Define

$$[A_k] = \kappa'([u_k],h_k) \quad \text{and} \quad [A_{j+1,k}] = \left[\frac{\partial u_{j+1}}{\partial z_k}\right]([Z_{j+1}]), \quad k \leq j+1. \tag{15}$$

From the relations (12) we have

$$[A_{j+1,k}] = [A_j][A_{j-1}]\cdots[A_k], \quad k \leq j+1. \tag{16}$$

where the empty product is the identity matrix. Using this relationship, the mean value form can be written

$$[u_{j+1}] = \hat{u}_{j+1} + \sum_{k=0}^{j+1}[A_{j+1,k}]\,([\hat{u}_k]-\hat{u}_k). \tag{17}$$

As $\hat{u}_{j+1}$ in theory is a real vector, the quality of the enclosure of (17) depends completely on the method chosen to calculate the sum. Define $[r_{j+1}]$ to be the sum in (17)

$$[r_{j+1}] = \sum_{k=0}^{j+1} [A_{j+1,k}] \left([\hat{u}_k] - \hat{u}_k \right). \tag{18}$$

3.2 Three enclosure methods

It is possible to calculate $[r_{j+1}]$ as is, using Eq. (18) with (16) directly. This would however be very inefficient. Instead some alternatives will be considered.

Enclosure method 1

Method one combines the sum into one interval vector.

$$[r_0] = [\hat{u}_0] - \hat{u}_0, \tag{19a}$$

$$\left[r_{j+1}\right] = [A_j][r_j] + [\hat{u}_{j+1}] - \hat{u}_{j+1}, \quad \text{for } j \geq 0. \tag{19b}$$

This enclosure is a very rough enclosure, it is in fact only a very simple rewriting of Eq. (18). Very often this kind of enclosure is of no practical use. Only in special cases, where we do not have to fight the wrapping effect, this method is useful. E.g. in the one dimensional case this enclosure is sufficient.

Enclosure method 2

Method two combines the sum into one interval vector $[\hat{r}_{j+1}]$ and one regular real matrix B_{j+1}.

$$[r_0] = B_0[\hat{r}_0], \quad \text{with } B_0 = I \text{ and } [\hat{r}_0] = [\hat{u}_0] - \hat{u}_0. \tag{20a}$$

$$\left[r_{j+1}\right] = B_{j+1}[\hat{r}_{j+1}], \quad \text{with} \tag{20b}$$

$$\left[\hat{r}_{j+1}\right] = (B_{j+1}^{-1}[A_j]B_j)[\hat{r}_j] + B_{j+1}^{-1}([\hat{u}_{j+1}] - \hat{u}_{j+1}), \quad j \geq 0. \tag{20c}$$

This method enables us to bound the solution in a moving and not necessary rectangular coordinate system. B_j has to be chosen so that $[A_j]B_j[\hat{r}_j] + [\hat{u}_{j+1}] - \hat{u}_{j+1}$ fits the solution best in the set theoretic sense and of course B_j also have to be regular. In the general case this enclosure geometrically forms a "parallelepiped" [9]. We will however here only consider the case where the matrix B_j is orthogonal, hence our enclosure is restricted to a moving orthogonal coordinate system.

Consider the following steps:

- Choose a $\tilde{A}_{j+1} \in [A_j]B_j$, for example the midpoint.
- Calculate the vector l so that the ith component l_i is the length of the ith column of $\tilde{A}_{j+1}$ multiplied with the diameter of the ith element in $[\hat{r}_j]$. We see that l consists of the lengths of the edges of the parallelepiped which is the result of the set theoretic operation $\tilde{A}_{j+1}[\hat{r}_j]$. Intuitively the direction in which this set has the largest span is also the most important direction when the parallelepiped is to be enclosed.
- Choose a permutation matrix P_{j+1} so that $\hat{l} = l^T P_{j+1}$ contains elements in decreasing order.
- Calculate $\hat{A}_{j+1} = \tilde{A}_{j+1}P_{j+1}$. Now $\hat{A}_{j+1}$ contains the same columns as $\tilde{A}_{j+1}$, but ordered by "importance".
- Let $B_{j+1} = Q$, where $\tilde{A}_{j+1} = QR$ is a $QR-$factorization. This assures that the coordinate system induced by B_{j+1} "fits best" the longest edges of the set theoretic result of $([A_j]B_j)[\hat{r}_j] + [\hat{u}_{j+1}] - \hat{u}_{j+1}$, [9].

Enclosure method 3

Method three is a combination of the previous methods. Consider

$$
\begin{aligned}
[\hat{r}_0^{(1)}] &= [\hat{r}_0^{(2)}] = [\hat{u}_0] - \hat{u}_0, \;\; B_0 = I && \text{(21a)}\\
[\hat{r}_{j+1}^{(1)}] &= [A_j][\hat{r}_j^{(1)}] + [\hat{u}_{j+1}] - \hat{u}_{j+1}, \quad \text{for } j \geq 0, && \text{(21b)}\\
[\hat{r}_{j+1}^{(2)}] &= (B_{j+1}^{-1}[A_j]B_j)[\hat{r}_j^{(2)}] + B_{j+1}^{-1}([\hat{u}_{j+1}] - \hat{u}_{j+1}), \quad \text{for } j \geq 0. && \text{(21c)}
\end{aligned}
$$

We would then have an enclosure

$$
[r_{j+1}] = B_{j+1}([\hat{r}_{j+1}^{(2)}] \cap B_{j+1}^{-1}[\hat{r}_{j+1}^{(1)}]) \cap ([\hat{r}_{j+1}^{(1)}] \cap B_{j+1}[\hat{r}_{j+1}^{(2)}]). \tag{21d}
$$

4 Taylor expansion

In (11) it was assumed that a discrete interval valued map $\kappa(\hat{u}_j, h_j) + [z_{j+1}]$ was known. It was also assumed that we could enclose the matrix $\kappa'([u_j], h_j)$. In this section a method to obtain these values by means of Taylor expansion will be described. Taylor expansion is a method to obtain higher order derivatives of expressions without doing any symbolic manipulations of expressions.

4.1 The recursive relations

Let $u(t)$ be an analytical function in a neighborhood of $t_j \in \mathbb{R}$. The Taylor expansion of u in this neighborhood is

$$u(t) = \sum_{k=0}^{\infty} (u)_k (t - t_j)^k, \tag{22}$$

where $(u)_k$ are the kth Taylor coefficient of u in t_j, given by

$$\begin{aligned} (u)_0 &= u(t_j), \\ (u)_k &= \tfrac{1}{k!} \frac{d^k u}{dt^k}(t_j), \quad k = 1, 2, \ldots . \end{aligned} \tag{23}$$

Let P_{p-1} be the approximating polynomial of degree $p-1$. A number $\xi \in]t_j, t[$ exists, so that

$$u(t) = P_{p-1}(t) + r_p(\xi) = \left(\sum_{i=0}^{p-1} (u)_i (t - t_j)^i \right) + \frac{1}{p!} \frac{d^p u}{dt^p}(\xi). \tag{24}$$

If $R_p([t_j, t])$ is an interval expansion of r_p evaluated in $[t_j, t]$ then

$$u(t) \in P_{p-1}(t) + R_p([t_j, t]). \tag{25}$$

From the definition of the Taylor coefficients (23) we have

$$(u)_k = \frac{1}{k} \frac{1}{(k-1)!} \frac{d^{(k-1)}}{dt^{(k-1)}} \frac{du}{dt}(t_j) = \frac{1}{k} (\frac{du}{dt})_{k-1}, \quad k = 1, 2, \ldots, \tag{26}$$

This form leads directly to the recursive formula for expanding the solution of the ordinary differential equation (7) in a point $t = t_j$.

$$\begin{aligned} (u)_0 &= u(t_j) \\ (u)_k &= \tfrac{1}{k} (f)_{k-1}, \quad k = 1, 2, \ldots . \end{aligned} \tag{27}$$

We see that to obtain the kth coefficient of u we have to know the $(k-1)$th coefficient of f. Fortunately this coefficient of f can usually be calculated using the 0th to the $(k-1)$th coefficients of u. The arithmetic to do so is called Taylor arithmetic. Let u, v be analytical functions in some area around $t = t_j$. In this area the following basic operations apply

$$
\begin{aligned}
(-u)_k &= -(u)_k, && (28a)\\
(u-v)_k &= (u)_k - (v)_k, && (28b)\\
(u+v)_k &= (u)_k + (v)_k, && (28c)\\
(u \cdot v)_k &= \sum_{i=0}^{k} (u)_i (v)_{k-i}, && (28d)\\
(u/v)_k &= ((u)_k - \sum_{i=1}^{k} (v)_i (u/v)_{k-i})/(v)_0. && (28e)
\end{aligned}
$$

These formulas are trivial to prove. Other, more exotic, functions can also be obtained, for example the cos and the sin function. Let u be analytic for t in some area around t_j. From the chain rule we have $\cos(u)' = -\sin(u) \cdot u'$ and $\sin(u)' = \cos(u) \cdot u'$. Using (26), (28a) and (28d) we get

$$
\begin{aligned}
(\cos u)_k &= -\frac{1}{k} \sum_{i=0}^{k-1} (i+1)(\sin(u))_{k-1-i}(u)_{i+1}, && (29a)\\
(\sin u)_k &= \frac{1}{k} \sum_{i=0}^{k-1} (i+1)(\cos(u))_{k-1-i}(u)_{i+1}. && (29b)
\end{aligned}
$$

The chain rule can in a similar way be used to obtain recursive formulas for other functions as well. See App. B for an extensive listing.

Using these tools, it is theoretically possible to recursively calculate all Taylor coefficients for the solution of (7) in the point t_j, provided, of course, with the value $u(t_j)$ as the zero order coefficient $(u)_0$. This is however not feasible on a computer, we cannot calculate an infinite number of coefficients. Instead we have to truncate the Taylor series to a degree $p-1$, and calculate the remainder term $R_p([t_j, t])$ to enclose the real solution using (25). To calculate the remainder term an enclosure of the pth derivative of u in the interval $[t_j, t]$ is needed. Obviously the same recursive relations, that the first p coefficients was calculated from, can be used, but unfortunately an enclosure for u in $[t_j, t]$ as the zero order coefficient has to be provided. At first glance it looks like we have an unsolvable problem. But fortunately, we do not have to provide a sharp bound for this enclosure.

4.2 The Picard operator

The Picard operator is used to find rough enclosures for the solutions of ordinary differential equations [1, 9, 10, 13, 16]. This first hand enclosure can then be improved by Taylor expansion.

The Picard operator, for the initial value problem (7) at $t = t_j$ is defined by

$$p(u)(t) = u_j + \int_{t_j}^{t} f(u(t))dt \quad \text{for } t \in t_j + [0, h_j]. \tag{30}$$

Now, u is a solution to (7) for $t \in t_j + [0, h_j]$ if and only if u is a fix-point in (30)

$$u(t) = p(u)(t) \Leftrightarrow \frac{du}{dt}(t) = f(u(t)),\ \ u(t_j) = u_j. \tag{31}$$

Furthermore, using the continuity of f, it can be proven that an interval I around t_j exists, where (7) has exactly one solution and that the iteration scheme $u^{(i+1)} = p(u^{(i)})$, where $u^{(0)} = u(t_j)$ will produce a sequence converging to this solution in I [16]. Let U be an interval and assume that $u(t) \in U$ for $t \in t_j + [0, h_j]$. Then $p(u)(t) \in P(U)$ for $t \in t_j + [0, h_j]$, where P is the interval Picard operator given by

$$P(U) = [u_j] + F(U) \cdot [0, h_j], \tag{32}$$

and F is an interval extension of f. The most important thing about this interval Picard operator, is that if $P(U) \subseteq U$ then the solution $u(t)$ to (7) is proven, by Banachs fix-point theorem, to exist in the interval $P(U)$ for $t \in t_j + [0, h_j]$. This feature is used to find a first hand enclosure of the solution to (7). The algorithm goes like this: Choose an initial guess $U^0 \supseteq [u_j]$ of an enclosure for $u(t)$, $t \in t_j + [0, h_j]$. Calculate $U^1 = P(U^0)$ using the interval Picard operator (32). If $U^1 \subseteq U^0$ we know that $u(t) \in U^1$ for $t \in t_j + [0, h_j]$, and using this enclosure the remainder term in a Taylor expansion can be calculated. However if $U^1 \not\subseteq U^0$, the Picard iteration has to be repeated, using another U^0. A good way to choose a new guess after a failed Picard iteration is to use ε-inflation

$$U^0 = (1 + \varepsilon)U^1 - \varepsilon U^1 \tag{33}$$

for some value $\varepsilon > 0$ (e.g. $\varepsilon = 0.1$). If it turns out, say after a fixed number of Picard iterations, that we are unable to prove the existence, then we probably have to reduce the discretization variable h_j and try again.

4.3 Discretizing the ordinary differential equation

With the presented tools we are capable of calculating everything needed to use the method described in Sec. 3. Assume that an enclosure $[u_j]$ of the solution $u(t)$ at $t = t_j$ and a point $\hat{u}_j \in [u_j]$is known. In order to use the method from Sec. 3 we have to do the following calculations:

- Choose an integration order p.
- Find a rough enclosure U of the solution in the interval $t \in t_j + [0, h_j]$, using the Picard iterations (32).
- Calculate $[z_{j+1}]$ by expanding a Taylor polynomial of degree p using the recursive relations derived from (7) and $(u)_0 = U$ as the zero order Taylor coefficient. Evaluate the remainder term $R_p([t_j, t_{j+1}])$ to obtain $[z_{j+1}]$.
- Calculate $\kappa(\hat{u}_j, h_j)$ by expanding a Taylor polynomial of degree $p-1$ using the recursive relations derived from (7) and $(u)_0 = \hat{u}_j$ as the zero order Taylor coefficient.
- Calculate $\kappa'([u_j], h_j)$ by expanding the Taylor polynomial of degree $p-1$ using the recursive relations derived from the variational equation of (7)

$$U'(t) = Df(u(t)) \cdot U, \quad U(t_j) = I, \quad t \in t_j + [0, h_j] \tag{34}$$

  and $(u)_0 = [u_j]$, $(U)_0 = I$ as the zero order Taylor coefficients.

We see that in order to use the method from Sec. 3, $2n + n^2$ Taylor polynomials has to be expanded in every integration step. For this reason, it is very important to optimize the code which is used to generate these Taylor polynomials.

Example

Consider the following ordinary differential equation [11]

$$\begin{aligned} u' &= u - v - u^3, \\ v' &= u - u^2 v. \end{aligned} \tag{35}$$

In order to find the recursive formulas, used to expand the Taylor polynomials for the solution, the equation has to be split up into a list of simple operations (a code-list), in which each of the simple operations can be expanded using the recursive relations from Sec. 4.1 and App. B. One possible code-list and the corresponding recursive relations[2] is

$$\begin{array}{rclrcl} T_1 &=& u\cdot u, & (T_1)_k &=& \sum_{i=0}^{k}(u)_i(u)_{k-i}, \;\; k\geq 1\\ T_2 &=& u\cdot T_1, & (T_2)_k &=& \sum_{i=0}^{k}(u)_i(T_1)_{k-i}, \;\; k\geq 1\\ T_3 &=& v-u, & (T_3)_k &=& (v)_k-(u)_k,\\ u' &=& T_3-T_2, & (u)_{k+1} &=& ((T_3)_k-(T_2)_k)/(k+1),\\ T_4 &=& T_1\cdot v, & (T_4)_k &=& \sum_{i=0}^{k}(T_1)_i(v)_{k-i}, \;\; k\geq 1\\ v' &=& u-T_4, & (v)_{k+1} &=& ((u)_k-(T_4)_k)/(k+1). \end{array} \tag{36}$$

Note that the value of T_1 is used twice, first to calculate T_2 and then to calculate T_4. It is very important to optimize the code-list, in this way, so that common subexpressions are identified in the generation of the code-list, and no overhead in the calculations of the Taylor coefficients occur.

The variational equation of Eq. (35) is

$$\begin{pmatrix} u'_u & u'_v\\ v'_u & v'_v \end{pmatrix} = \begin{pmatrix} 1-3u^2 & -1\\ 1-2uv & -u^2 \end{pmatrix}\cdot\begin{pmatrix} u_u & u_v\\ v_u & v_v \end{pmatrix} \tag{37}$$

The following additional code can be added to (36) to obtain the expansion of the solution of the variational equation together with the expansion of the solution of (35) itself

$$\begin{array}{rclrcl} T_5 &=& 1-3\cdot T_1, & (T_5)_k &=& -3(T_1)_k, \;\; k\geq 1\\ T_6 &=& T_5\cdot u_u, & (T_6)_k &=& \sum_{i=0}^{k}(T_5)_i(u_u)_{k-i}, \;\; k\geq 1\\ u'_u &=& T_6-v_u, & (u_u)_{k+1} &=& ((T_6)_k-(v_u)_k)/(k+1),\\ T_7 &=& T_5\cdot u_v, & (T_7)_k &=& \sum_{i=0}^{k}(T_5)_i(u_v)_{k-i}, \;\; k\geq 1\\ u'_v &=& T_7-v_v, & (u_v)_{k+1} &=& ((T_7)_k-(v_v)_k)/(k+1),\\ T_8 &=& 1-2uv, & (T_8)_k &=& -2\sum_{i=0}^{k}(u)_i(v)_{k-i}, \;\; k\geq 1\\ T_9 &=& T_8\cdot u_u, & (T_9)_k &=& \sum_{i=0}^{k}(T_8)_i(u_u)_{k-i}, \;\; k\geq 1\\ T_{10} &=& T_1\cdot v_u, & (T_{10})_k &=& \sum_{i=0}^{k}(T_1)_i(v_u)_{k-i}, \;\; k\geq 1\\ v'_u &=& T_9-T_{10}, & (v_u)_{k+1} &=& ((T_9)_k-(T_{10})_k)/(k+1),\\ T_{11} &=& T_8\cdot u_v, & (T_{11})_k &=& \sum_{i=0}^{k}(T_8)_i(u_v)_{k-i}, \;\; k\geq 1\\ T_{12} &=& T_1\cdot v_v, & (T_{12})_k &=& \sum_{i=0}^{k}(T_1)_i(v_v)_{k-i}, \;\; k\geq 1\\ v'_v &=& T_{11}-T_{12}, & (v_v)_{k+1} &=& ((T_{11})_k-(T_{12})_k)/(k+1). \end{array} \tag{38}$$

From this example we see that even for simple ordinary differential equations, the generation of the code-lists by hand is a rather time consuming process. Also the possibility of making errors in the process is large.

[2]Note that the calculation of T_1 can be optimized by using the formula for u^2 from App. B.

5 DTU AWA

DTU AWA is a program package which solves initial value problems with error bounds. The package is written in C++ and uses the interval package PROFIL/BIAS [7, 8] which can be obtained via anonymous ftp. The writing of the DTU AWA package is not considered to be finished, as of the writing of this report. We will, for this reason, not go into too much detail concerning the package.

The package consists of two parts:

- The precompiler ode2cc (computer slang for 'ode to cc') which is used to generate the codelist. The user first writes the ordinary differential equation using a very simple syntax. ode2cc is then called to convert the equation into a codelist using a C++ syntax. The codelist does only have to be generated once for each ordinary differential equation. Also constants can be specified, they can be changed later, without generating another codelist.
- The integration library DTU_AWA consists of all the routines used to integrate the initial value problem. This library should be included from the users program together with the automatically generated codelist and of course the interval libraries.

It is possible to use any of the two parts without the other. It is for example possible to use a user generated codelist in DTU_AWA if the user is not satisfied with the automatically generated codelist from ode2cc.

5.1 Automatic generation of the codelist using `ode2cc`

The precompiler ode2cc is very simple to use. Nevertheless it is actually a rather complicated program. Maybe even more complicated than the integration library DTU_AWA itself. The reason for this complexity is to provide a tool which is easy to use, but also a tool which generates efficient code-lists, without too much overhead[3]. The codelist only have to be generated once, so the speed of the preprocessor is not too important.

The precompiler ode2cc consists (roughly) of the following elements

- A dynamical string class.

[3] Hand made code-lists are usually more efficient, but also time-consuming to produce.

- A lexical analyzer, which reads a text file and identifies tokens.
- A library of parser builders and basic parsers. These tools are used to build the compound parsers from.
- A compound parser to parse expressions and equations.
- A symbolic algebra rewriting tool, used to perform symbolic differentiation and simplifications of expressions.
- A compound parser to parse the ordinary differential equations.
- A program which identifies common subexpressions in a list of expressions.
- The main program which first uses the above programs to process the ordinary differential equation, and then writes out the codelist in C++ syntax.

We will not go into the detail on the syntax of the input file for the ode2cc precompiler, as this syntax is likely to change in future versions of ode2cc, but instead give some examples.

Example

The ordinary differential equation (35) is specified in File 5.1. A call to the pre-

File 5.1 circle.ode

```
ode [u,v]{
equation

 u't = u*(1-sqr(u))-v,
 v't = u*(1-u*v);

}
```

processor `ode2cc circle.ode > circle.cge` will then generate the codelist in File 5.2 defining two functions. These two C++ functions are used by the DTU_AWA integrator to expand the Taylor polynomial in the following way: First of all, the values `T[0][0]` and `T[1][0]` should be initialized by the user

File 5.2 circle.cge

```
void init(INTERVAL** T){
/* Remember to initialize:
 T[0][0] = u;
 T[1][0] = v;
---------------------*/
 T[2][0] = 1;
 T[3][0] = Sqr(T[0][0]);
 T[4][0] = T[2][0]-T[3][0];
 T[5][0] = T[0][0]*T[4][0];
 T[6][0] = T[5][0]-T[1][0];
 T[7][0] = T[0][0]*T[1][0];
 T[8][0] = T[2][0]-T[7][0];
 T[9][0] = T[0][0]*T[8][0];

 T[0][1] = T[6][0];
 T[1][1] = T[9][0];
}
```

```
void stepper(int c,INTERVAL** T){
 T[3][c] = SQR(T[0],c);
 T[4][c] = T[2][c]-T[3][c];
 T[5][c] = MUL(T[0],T[4],c);
 T[6][c] = T[5][c]-T[1][c];
 T[7][c] = MUL(T[0],T[1],c);
 T[8][c] = T[2][c]-T[7][c];
 T[9][c] = MUL(T[0],T[8],c);

 T[0][c+1] =T[6][c]/(c+1);
 T[1][c+1] =T[9][c]/(c+1);
}
```

with the zero order Taylor coefficients of u and v. Then the function `init` is called to initialize all the zero order coefficients of the intermediate values in the codelist and to calculate the 1. order Taylor coefficient of u and v. After the initialization the function `stepper` is called $p-1$ successive times to expand the pth order Taylor polynomials for the solutions u and v.

Also a codelist for the extended system, i.e. the ordinary differential equation with the variational equation has to be specified[4]. Here the symbolic differentiation tool may be of help. Note that a pling symbol “ ’ ” is a binary operation which means “differentiated with respect to”. For example the expression `(u*(1-u*v))'u` will be rewritten to the expression `1-u*v-u*v` . This final expression is not very optimal, maybe future versions of ode2cc will take care of this. Before the search for identical subexpressions and the generation of the codelist, the ordinary differential equation is rewritten to the form in File 5.4. Obviously a lot of improvements can be made and it may be a good idea to do some manually rewritings. Especially when the codelist is going to be used in many calculations. We will however not do manually rewritings in this example. After the search for identical subexpressions the codelist in File 5.5 is produced.

[4]In future versions of ode2cc this is also likely to change as all information about the variational equation is given in the ordinary differential equation.

File 5.3 circle.eode

```
ode [u,v,uu,uv,vu,vv]{
equation

 u't = u*(1-sqr(u))-v,
 v't = u*(1-u*v),

 uu't = (u*(1-sqr(u))-v)'u*uu + (u*(1-sqr(u))-v)'v*vu,
 uv't = (u*(1-sqr(u))-v)'u*uv + (u*(1-sqr(u))-v)'v*vv,
 vu't = (u*(1-u*v))'u*uu + (u*(1-u*v))'v*vu,
 vv't = (u*(1-u*v))'u*uv + (u*(1-u*v))'v*vv;

}
```

File 5.4

```
ode [u,v,uu,uv,vu,vv]{
equation

 u't=u*(1-sqr(u))-v,
 v't=u*(1-u*v),

 uu't=(1-sqr(u)-u*2*u)*uu+(-u*2*u-1)*vu,
 uv't=(1-sqr(u)-u*2*u)*uv+(-u*2*u-1)*vv,
 vu't=(1-u*v-u*v)*uu+(-sqr(u))*vu,
 vv't=(1-u*v-u*v)*uv+(-sqr(u))*vv;
}
```

5.2 The integrator DTU_AWA

Once the recursive relations has been generated, they can be included in a C++ program together with the DTU_AWA library to perform validated integration.

The integrator in DTU_AWA, discretizes the ordinary differential equation using the steps described in Sec. 4.3. After each discretization step, the mean value form is used with enclosure method number 3 from Sec. 3.2. The user of DTU_AWA has to specify some parameters before calling the integrator: The order of the Taylor expansions have to be specified. This order also determines the accuracy of the integration. Also the initial step-size has to be specified. This step-size is automatically adjusted throughout the integration. The user has an option to calculate an enclosure of the solution to the variational equation. Enabling this option only costs a little overhead, as most of the information on the variational equation is available anyway.

Example

The initial value problem given by the ordinary differential equation (35), using the non-optimized codelist from ode2cc, was solved using the initial values $u = 1+[-4,4]\cdot 10^{-3}$ and $v = [-4,4]\cdot 10^{-3}$. The integration interval was $[0,7]$. See Fig. 1 for a plot of the result. The boxes in Fig. 1 encloses the solution at certain values of the independent integration variable. From the figure, it can be seen that the uncertainty grows very rapidly as the system is integrated. This is typically the case in interval calculations, when large intervals are used as initial values the overestimation often becomes too large leading to unacceptable results. Sometimes we will even get overflow, and the calculations will stop. We have to start with small intervals to obtain better results. The vector field given by the right hand side of the differential equation is plotted in Fig. 2 to give an impression of the behavior of the system.

File 5.5 circle.ecge

```
void init(INTERVAL** T){
/* Remember to initialize:
 T[0][0] = u;
 T[1][0] = v;
 T[2][0] = uu;
 T[3][0] = uv;
 T[4][0] = vu;
 T[5][0] = vv;
---------------------*/
 T[6][0] = 1;
 T[7][0] = Sqr(T[0][0]);
 T[8][0] = T[6][0]-T[7][0];
 T[9][0] = T[0][0]*T[8][0];
 T[10][0] = T[9][0]-T[1][0];
 T[11][0] = T[0][0]*T[1][0];
 T[12][0] = T[6][0]-T[11][0];
 T[13][0] = T[0][0]*T[12][0];
 T[14][0] = 2;
 T[15][0] = T[14][0]*T[0][0];
 T[16][0] = T[0][0]*T[15][0];
 T[17][0] = T[8][0]-T[16][0];
 T[18][0] = T[17][0]*T[2][0];
 T[19][0] = -T[16][0];
 T[20][0] = T[19][0]-T[6][0];
 T[21][0] = T[20][0]*T[4][0];
 T[22][0] = T[18][0]+T[21][0];
 T[23][0] = T[17][0]*T[3][0];
 T[24][0] = T[20][0]*T[5][0];
 T[25][0] = T[23][0]+T[24][0];
 T[26][0] = T[12][0]-T[11][0];
 T[27][0] = T[26][0]*T[2][0];
 T[28][0] = -T[7][0];
 T[29][0] = T[28][0]*T[4][0];
 T[30][0] = T[27][0]+T[29][0];
 T[31][0] = T[26][0]*T[3][0];
 T[32][0] = T[28][0]*T[5][0];
 T[33][0] = T[31][0]+T[32][0];

 T[0][1] = T[10][0];
 T[1][1] = T[13][0];
 T[2][1] = T[22][0];
 T[3][1] = T[25][0];
 T[4][1] = T[30][0];
 T[5][1] = T[33][0];
}
```

```
void stepper(int c,INTERVAL** T){
 T[7][c] = SQR(T[0],c);
 T[8][c] = T[6][c]-T[7][c];
 T[9][c] = MUL(T[0],T[8],c);
 T[10][c] = T[9][c]-T[1][c];
 T[11][c] = MUL(T[0],T[1],c);
 T[12][c] = T[6][c]-T[11][c];
 T[13][c] = MUL(T[0],T[12],c);
 T[15][c] = MUL(T[14],T[0],c);
 T[16][c] = MUL(T[0],T[15],c);
 T[17][c] = T[8][c]-T[16][c];
 T[18][c] = MUL(T[17],T[2],c);
 T[19][c] = -T[16][c];
 T[20][c] = T[19][c]-T[6][c];
 T[21][c] = MUL(T[20],T[4],c);
 T[22][c] = T[18][c]+T[21][c];
 T[23][c] = MUL(T[17],T[3],c);
 T[24][c] = MUL(T[20],T[5],c);
 T[25][c] = T[23][c]+T[24][c];
 T[26][c] = T[12][c]-T[11][c];
 T[27][c] = MUL(T[26],T[2],c);
 T[28][c] = -T[7][c];
 T[29][c] = MUL(T[28],T[4],c);
 T[30][c] = T[27][c]+T[29][c];
 T[31][c] = MUL(T[26],T[3],c);
 T[32][c] = MUL(T[28],T[5],c);
 T[33][c] = T[31][c]+T[32][c];

 T[0][c+1] =T[10][c]/(c+1);
 T[1][c+1] =T[13][c]/(c+1);
 T[2][c+1] =T[22][c]/(c+1);
 T[3][c+1] =T[25][c]/(c+1);
 T[4][c+1] =T[30][c]/(c+1);
 T[5][c+1] =T[33][c]/(c+1);
}
```

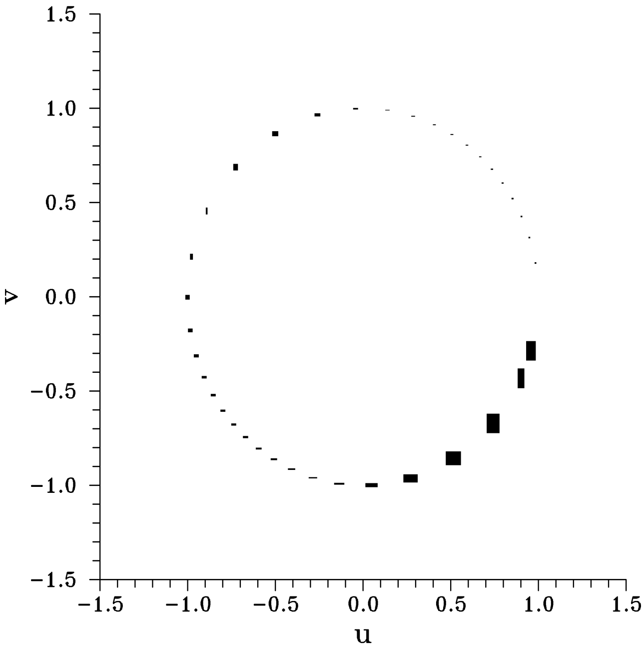

Figure 1: The intermediate steps in the integration of the differential equation (35).

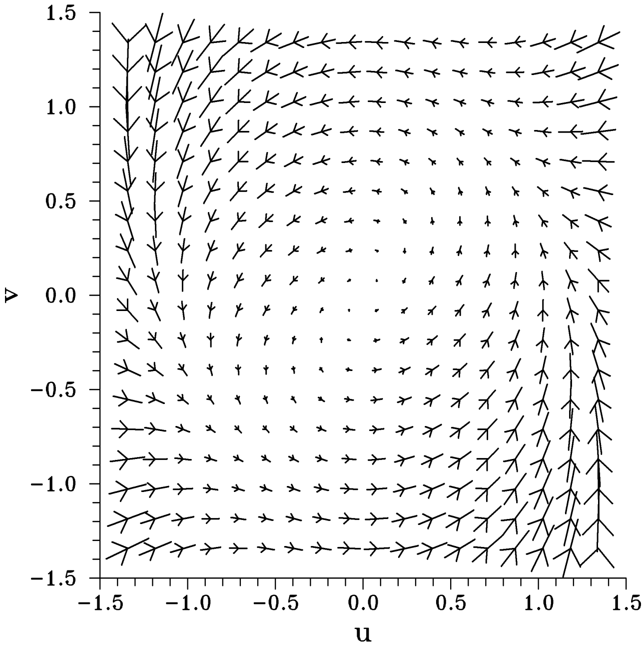

Figure 2: A plot of the vector field given by the right hand side of (35).

6 Proving existence of periodic solutions

The existence of periodic solutions and their stability is a very important field in dynamical systems. If we are investigating a model which has a stable periodic orbit then the solution will drift into this orbit from a large set of initial states and never leave it again. This is the reason why stable periodic solutions are more easy to find than the unstable ones. With help from validated integration it is possible to prove the existence of periodic solutions to ordinary differential equations, no matter if they are stable or unstable.

Consider

$$u' = f(u) \quad \text{and} \quad u(0) = u(T), \tag{39}$$

where $u(t) \in \mathbb{R}^n$ and $f \in C^p(D,\mathbb{R}^n)$ where $D \subset \mathbb{R}^n$ is an open set. Here u is a periodic solution with the period T. A way to find periodic solutions is to define a discrete map Φ

$$\Phi(u_0) = u(T) \quad \text{where} \quad u' = f(u) \quad \text{and} \quad u(0) = u_0. \tag{40}$$

If we find a u_0 so that $u_0 = \Phi(u_0)$ then u_0 is a value on a periodic solution. One way to find such values is to use Newtons method to find zeros of the function $\Phi(u_0) - u_0$.

6.1 The Brusselator

The Brusselator is a well known ordinary differential equation which simulates a chemical process [6].

$$\begin{aligned} u' &= A + u^2 v - Bu - u + \alpha\cos(\omega t), \\ v' &= Bu - u^2 v. \end{aligned} \tag{41}$$

The equation is interesting because, even when $\alpha = 0$ so that the forcing term disappears, it has a periodic solution. Assume that (u, v) is a periodic solution with period p, then (x, y) given by

$$\begin{aligned} x(\tau) &= u(p \cdot \tau), \\ y(\tau) &= v(p \cdot \tau). \end{aligned} \tag{42}$$

is also periodic, but with the period 1. If we rewrite Eq. (41) using Eq. (42) and add both p and τ as phase variables we will get the equation in x and y shown

File 6.1 bruss.ode

```
ode [x,y,p,t]{
 constant A, B, a, o;
 equation

  x't = p*(A+x*(x*y-B-1)+a*cos(o*p*t)),
  y't = p*(x*(B-x*y)),
  p't = 0,
  t't = 1;

}
```

in File 6.1. which is the final form, ready to be compiled by `ode2cc`. Note that `t't` has been added to the equation to make it autonomous, while `p't` is added so that we can obtain first order information on the dependency between the discrete map Φ given by Eq. (40) and the value of p by using the variational equation. The code-lists were produced using `ode2cc` with some interactive optimization of the symbolic differentiation. The length of the code-lists were 24 for the normal differential equation and 116 for the equation including the variational equation.

The unforced case

The unforced case, when $\alpha = 0$, has been investigated with the parameters $A = \frac{2}{5}$ and $B = \frac{6}{5}$. To give an impression of how the system looks like, with this set of parameters, the vector field given by the right hand side of Eq. (41) has been plotted in Fig. 3. It is known that a periodic solution cannot surround a region containing no equilibrium point [5]. Since $(u,v) = (\frac{2}{5},3)$ is the only equilibrium point we know that a periodic solution $(u(t),v(t))$ must contain points where $u(t) = \frac{2}{5}$ and points where $v(t) = 3$. We use this information to find a periodic solution. In this example we have fixed the value of $y_0 = 3$. Using Eq. (40) we obtain

$$\Phi\left(\begin{bmatrix} x_0 \\ 3 \\ p \\ 0 \end{bmatrix}\right) - \begin{bmatrix} x_0 \\ 3 \\ p \\ 1 \end{bmatrix} = 0, \tag{43}$$

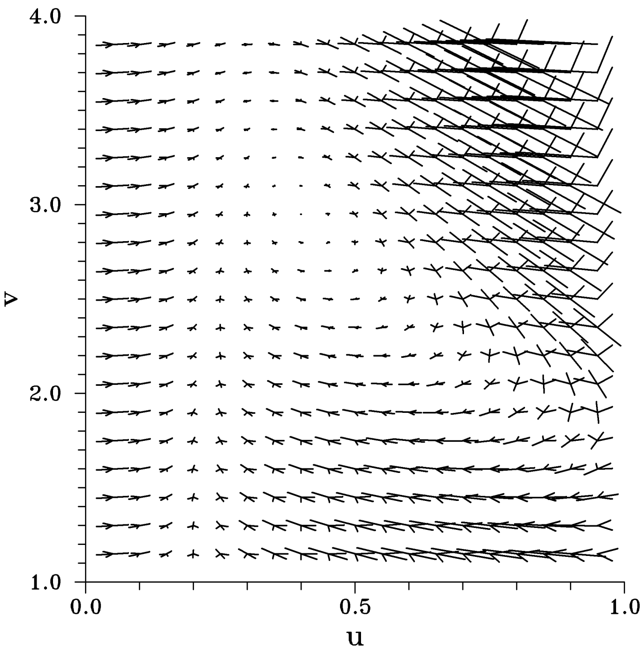

Figure 3: A plot of the vector field given by the right hand side of the Brusselator (41) with the parameters $\alpha = 0$, $A = \frac{2}{5}$ and $B = \frac{6}{5}$.

with $T = 1$ (see Eq. (40)). But we know that the value of p is constant during the integration and we know the value of the independent variable τ is equal to 1 when the integration has stopped, so the real problem is only two dimensional. Let X and Y be the first and the second elements of Φ with the arguments as in Eq. (43). Now the equation

$$\begin{aligned} 0 &= x_0 - X(x_0, p), \\ 0 &= 3 - Y(x_0, p), \end{aligned} \tag{44}$$

will have a solution (x_0, p) if $(u, v) = (x_0, 3)$ is a point on a periodic solution of Eq. (41) with the period p. This equation has been solved, using the interval

Newton method [2, 12, 13] which is a self validating method

$$N\left(\begin{bmatrix}[x_0]\\ [p]\end{bmatrix}\right) = \begin{bmatrix}\hat{x}_0\\ \hat{p}\end{bmatrix} - D\left(\begin{bmatrix}[x_0]-X([x_0],[p])\\ 3-Y([x_0],[p])\end{bmatrix}\right)^{-1}\cdot\begin{bmatrix}\hat{x}_0-X(\hat{x}_0,\hat{p})\\ 3-Y(\hat{x}_0,\hat{p})\end{bmatrix} \quad (45)$$

where $\hat{x}_0 \in [x_0]$ and $\hat{p} \in [p]$, in this report we use the approximative mid points. It was possible to prove the existence and uniqueness[5] of a periodic solution for the values $[x_0] = 0.30^{81}_{74}$ and $[p] = 16.75^{7}_{1}$. In Fig. 4 the intermediate steps in the integration can be seen. The bounds of the solution can very easily be sharpened

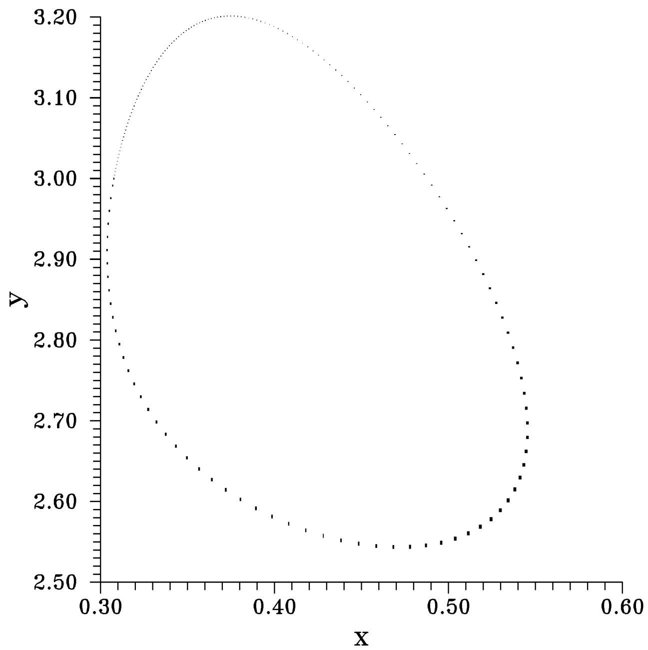

Figure 4: A validated enclosure of the periodic orbit for discrete values of the integration variable. With parameters $\alpha = 0$, $A = \frac{2}{5}$ and $B = \frac{6}{5}$.

[5]See App. A for a discussion of the interval representation used in this report.

to an uncertainty under 10^{-11} in both x_0 and p by repeating the Newton iterations. Note that we want to prove uniqueness of a solution for as large a set of initial values as possible, this is the reason why we integrate using large intervals. Also the stability of the orbit has been calculated by computing an enclosure of the eigenvalues for the Jacobi matrix from the discrete map Φ in the (x,y) space. The periodic orbit has been proven to be stable with one real eigenvalue equal to 1, which is always the case in autonomous systems and one real eigenvalue with absolute value less than one (approx. 0.4396). This stability means that a set of initial values for the system will make an approximate solution, using a normal integration scheme, converge towards this periodic solution. If we integrate using intervals, we may be able to integrate the system in infinite time, without loosing precision.

The forced case

In the forced Brusselator, when $\alpha \neq 0$ the period of the orbits can only be of the same periods as the forcing term, and for this reason we only have to search for periodic orbits for $p = \frac{2\pi}{\omega} i,\ \ i = 1,2,\ldots$. As the vector field is non-constant with respect to t no equilibrium points exists, and a coordinate-value on the periodic orbit can not be fixed as we could in the unforced case. Because of this we have to use Newtons method in the (x,y) space. For a given p in Eq. (43) the equation

$$\begin{aligned} 0 &= x_0 - X(x_0,y_0), \\ 0 &= y_0 - Y(x_0,y_0), \end{aligned} \tag{46}$$

where X and Y are the first and the second elements of Φ, will have a solution (x_0,y_0) if (x_0,y_0) is a point on a periodic solution of (41) with the period p. As in the unforced case, interval Newton method is used to prove existence of periodic solutions.

Using the parameters $A = \frac{2}{5}$, $B = \frac{6}{5}$, $a = \frac{3}{100}$ and $\omega = \frac{\pi}{4}$ it was possible to prove the existence and uniqueness of a period 1 solution[6] ($p = \frac{2\pi}{\omega}$) with the values $[x_0] = 0.38^{83}_{73}$ and $[y_0] = 3.05^{2}_{1}$. See Fig. 5. By repeating the Newton iterations the bounds of the result has been sharpened to an uncertainty under 10^{-12} in both $[x_0]$ and $[y_0]$. The periodic solution has been proven to be unstable with two real eigenvalues (approx. -0.6469 and -1.8025). Because of the instability, this periodic orbit will never show up in a normal integration process. Even if we start on the orbit, the integration error will eventually make the approximate

[6] The periodicity of a periodic solution is denoted relatively to the period of the forcing term.

solution drift away from the true solution. If intervals are used in the integration, these intervals will continue to grow, until the algorithm breaks down. Also a

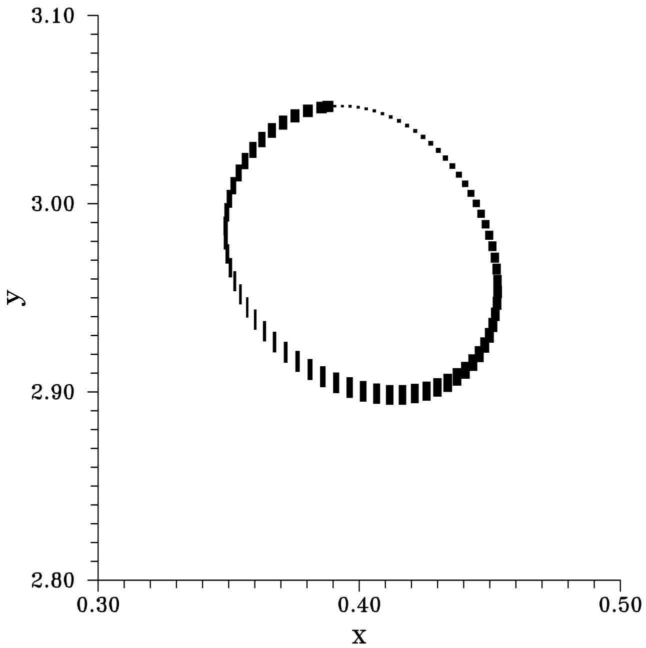

Figure 5: A validated enclosure of the periodic orbit, with period $p = \frac{2\pi}{\omega}$, for discrete values of the integration variable. With parameters $A = \frac{2}{5}$, $B = \frac{6}{5}$, $\alpha = \frac{3}{100}$ and $\omega = \frac{\pi}{4}$.

period 2 solution ($p = \frac{4\pi}{\omega}$) has been proven to exist. See Fig. 6. Both existence and uniqueness has been proven with the values $[x_0] = 0.3850^9_0$ and $[y_0] = 3.25^{14}_{20}$. The bounds of the solution has been sharpened to an uncertainty under 10^{-11} in both $[x_0]$ and $[y_0]$. The periodic solution has been proven to be stable with two real eigenvalues (approx. -0.0560 and -0.9858). Like in the autonomous system this stable periodic orbit can be seen using normal integration. Also here we may be able to integrate the system in infinite time, without loosing precision.

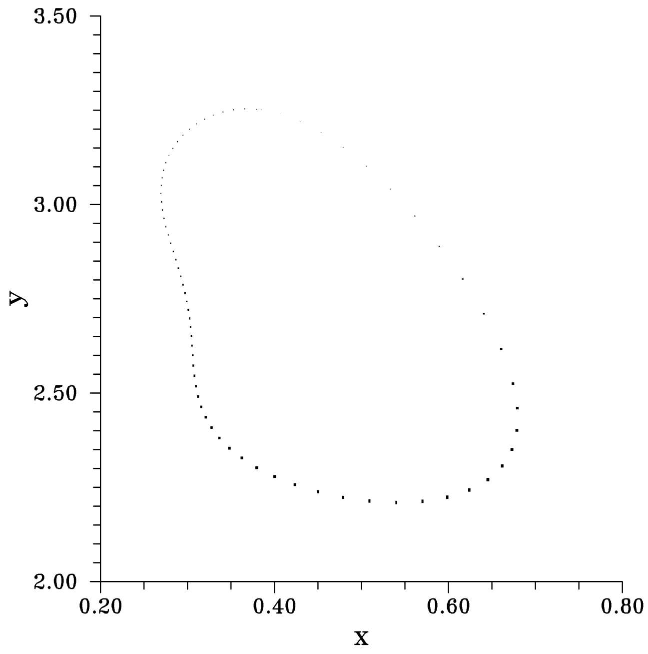

Figure 6: A validated enclosure of the periodic orbit, with period $p = \frac{4\pi}{\omega}$, for discrete values of the integration variable. With parameters $A = \frac{2}{5}$, $B = \frac{6}{5}$, $\alpha = \frac{3}{100}$ and $\omega = \frac{\pi}{4}$.

7 Conclusion

We have seen that numerical integration of ordinary differential equations can be done so that the true solution of the integration is bounded by intervals. This is of great importance in non-linear differential equations, as many problems in this field, to this day, are unsolvable using pure mathematics. In this report we have only considered initial value problems, but the method can also be used to solve boundary value problems [9, 10].

An integration program package DTU AWA, has been developed to do validated integration. This package is still in its developing phase as a lot of improvements can be made to the preprocessor which generates the code list used in the integration, as well as to the integration algorithm itself. The source for the current version of DTU_AWA can be obtained by sending an Email to the author of this report. The PROFIL/BIAS libraries are public domain and can be obtained by anonymous ftp to

`<ftp://ti3sun.ti3.tu-harburg.de>`.

The original AWA program was developed by Rudolf Lohner. His package is based on the XSC languages, which are commercial languages based on FORTRAN, PASCAL and C, but with interval arithmetics provided. Further information on the XSC languages can be obtained from

`<http://www.uni-karlsruhe.de/∼iam/lehrstuhl2.html>`.

The source code and some (stand alone) executables of the AWA program can be obtained by anonymous ftp to

`<ftp://iamk4515.mathematik.uni-karlsruhe.de/pub/awa/>`.

A Representation of intervals

A general problem of using exact numerical methods is that the computer representation of floating point numbers are not easy for a human to read, humans are used to numbers with base 10 while computers use base 2. Normally this is no problem because of conversion programs which converts numbers in between the two bases. But unfortunately these programs often make roundoff errors so that our interpretation of a computer generated proof may be wrong. In most cases it is enough to know that the computer has proven some result for some intervals of which we do not know the exact values. But if we are publishing these intervals we have to know the exact values down to the last digit.

Fortunately the integer numbers are normally exact representable on computers, that is if we stick to small values. But in most cases the numbers used in a calculation are not representable on a computer and intervals are used instead to bound the correct values. To generate correct bounds we use integers and interval arithmetics. E.g. to enclose the number $\frac{1}{3}$ we perform the division in rounded interval arithmetics to obtain a lower and an upper bound of the value. If we want a correct enclosure of the interval 0.30^{81}_{74} we will calculate the outwards rounded result of $030^{81}_{74}/10^4$.

Sometimes outwards rounding are not good enough. E.g. if we want to prove the existence and uniqueness of some solution in an interval which is not representable on a computer. The computer generated proof will then instead be valid for the rounded value of the interval. If outwards rounding is used to enclose the interval then the solution could lie outside the original interval. If inwards rounding is used then another solution could exist between the interval and the inwards rounded interval. To deal with this problem both rounding has to be applied and the calculation has to be done twice, for the outwards as well as for the inwards rounded interval. Actually we only have to prove uniqueness for the outwards rounded interval and existence for the inwards rounded interval to get existence and uniqueness of the original interval.

B List of Taylor-coefficient formulas

$$(u \cdot v)_k = \sum_{j=0}^{k} (u)_j (v)_{k-j}, \tag{47}$$

$$(u/v)_k = \frac{1}{v}\left((u)_k - \sum_{j=1}^{k} (v)_j (u/v)_{k-j}\right), \tag{48}$$

$$(u^a)_k = \frac{1}{ku} \cdot \sum_{j=0}^{k-1} (a(k-j)-j)(u)_{k-j}(u^a)_j, \tag{49}$$

$$(u^2)_k = \begin{cases} 2 \cdot \sum_{j=0}^{(k+1)\mathrm{div}2-1} (u)_j (u)_{k-j}, & k \text{ is odd}, \\ 2 \cdot \sum_{j=0}^{(k+1)\mathrm{div}2-1} (u)_j (u)_{k-j} + (u)^2_{(k+1)\mathrm{div}2}, & k \text{ is even}. \end{cases}, \tag{50}$$

$$(\sqrt{u})_k = \begin{cases} \frac{1}{\sqrt{u}}\Big((u)_k - 2 \cdot \sum_{j=1}^{(k+1)\mathrm{div}2-1} (\sqrt{u})_j (\sqrt{u})_{k-j} \\ \qquad -(\sqrt{u})_k \sqrt{u}\Big), & k \text{ is odd}, \\ \frac{1}{\sqrt{u}}\Big((u)_k - 2 \cdot \sum_{j=1}^{(k+1)\mathrm{div}2-1} (\sqrt{u})_j (\sqrt{u})_{k-j} \\ \qquad -(\sqrt{u})_k \sqrt{u} - (\sqrt{u})^2_{(k+1)\mathrm{div}2}\Big), & k \text{ is even}. \end{cases}, \tag{51}$$

$$(\exp u)_k = \sum_{j=0}^{k-1} (1 - \frac{j}{k})(\exp u)_j (u)_{k-j}, \tag{52}$$

$$(\log u)_k = \frac{1}{u}\left((u)_k - \sum_{j=1}^{k-1} (1 - \frac{j}{k})(u)_j (\log u)_{k-j}\right), \tag{53}$$

$$(\sin u)_k = \frac{1}{k} \cdot \sum_{j=0}^{k-1} (j+1)(\cos u)_{k-1-j}(u)_{j+1}, \tag{54}$$

$$(\cos u)_k = -\frac{1}{k} \cdot \sum_{j=0}^{k-1} (j+1)(\sin u)_{k-1-j}(u)_{j+1}, \tag{55}$$

$$(\arcsin u)_k = \frac{1}{\sqrt{1-u^2}} \left((u)_k - \frac{1}{k} \cdot \sum_{j=1}^{k-1} j(\arcsin u)_j (\sqrt{1-u^2})_{k-j} \right), \tag{56}$$

$$(\arccos u)_k = \frac{-1}{\sqrt{1-u^2}} \left((u)_k + \frac{1}{k} \cdot \sum_{j=1}^{k-1} j(\arccos u)_j (\sqrt{1-u^2})_{k-j} \right), \tag{57}$$

$$(\tan u)_k = \frac{1}{\cos^2 u} \left((u)_k - \frac{1}{k} \cdot \sum_{j=1}^{k-1} j(\tan u)_j (\cos^2 u)_{k-j} \right), \tag{58}$$

$$(\cot u)_k = \frac{-1}{\sin^2 u} \left((u)_k + \frac{1}{k} \cdot \sum_{j=1}^{k-1} j(\cot u)_j (\sin^2 u)_{k-j} \right), \tag{59}$$

$$(\arctan u)_k = \frac{1}{1+u^2} \cdot \left((u)_k - \frac{1}{k} \sum_{j=1}^{k-1} j(\arctan u)_j (1+u^2)_{k-j} \right), \tag{60}$$

$$(\mathrm{arccot} u)_k = -\frac{1}{1+u^2} \cdot \left((u)_k + \frac{1}{k} \sum_{j=1}^{k-1} j(\mathrm{arccot} u)_j (1+u^2)_{k-j} \right), \tag{61}$$

$$(\sinh u)_k = \frac{1}{k} \cdot \sum_{j=0}^{k-1} (j+1)(\cosh u)_{k-1-j}(u)_{j+1}, \tag{62}$$

$$(\cosh u)_k = \frac{1}{k} \cdot \sum_{j=0}^{k-1} (j+1)(\sinh u)_{k-1-j}(u)_{j+1}, \tag{63}$$

$$(\mathrm{arsinh} u)_k = \frac{1}{\sqrt{u^2+1}} \left((u)_k - \frac{1}{k} \cdot \sum_{j=1}^{k-1} j (\mathrm{arsinh} u)_j (\sqrt{u^2+1})_{k-j} \right), \tag{64}$$

$$(\mathrm{arcosh} u)_k = \frac{-1}{\sqrt{u^2-1}} \left((u)_k - \frac{1}{k} \cdot \sum_{j=1}^{k-1} j (\mathrm{arcosh} u)_j (\sqrt{u^2-1})_{k-j} \right), \tag{65}$$

$$(\mathrm{artanh} u)_k = \frac{1}{1-u^2} \cdot \left((u)_k - \frac{1}{k} \sum_{j=1}^{k-1} j (\mathrm{artanh} u)_j (1-u^2)_{k-j} \right), \tag{66}$$

$$(\mathrm{arcoth} u)_k = \frac{1}{1-u^2} \cdot \left((u)_k - \frac{1}{k} \sum_{j=1}^{k-1} j (\mathrm{arcoth} u)_j (1-u^2)_{k-j} \right). \tag{67}$$

References

[1] V. I. Arnol'd, (1992) *Ordinary differential equations.* Springer-Verlag Berlin, Heidelberg, News York.

[2] O. Caprani and K. Madsen, (1992) *Introduktion til Interval Analyse.* Technical report NI-92-03, Danmarks Tekniske Højskole.

[3] G. F. Corliss, G. S. Krenz and P. H. Davis, (1988) *Bibliography on interval methods for the solution of ordinary differential equations.* Technical report **289**, Department of Mathematics, Statistics and Computer Science, Marquette University, Milwaukee, Wisconsin.

[4] J. G. Hocking and G. S. Young, (1961) *Topology.* Addison-Wesley Publishing company, inc.

[5] D. W. Jordan and P. Smith, (1977) *Nonlinear Ordinary Differential Equations.* Oxford applied mathematics and computing science series.

[6] C. Knudsen, J. Sturis and J. S. Thomsen, (1991) *Generic bifurcation structures of Arnol'd tongues in forced oscillators.* Physical review A **44** 6, p.3503 – 3510.

[7] O. Knüppel, (1993) *BIAS - Basic Interval Arithmetic Subroutines.* Technische Universität, Hamburg - Harburg.

[8] O. Knüppel, (1993) *PROFIL - Programmer's Runtime Optimized Fast Interval Library.* Technische Universität, Hamburg - Harburg.

[9] R. Lohner, (1987) *Enclosing the Solutions of Ordinary Initial and Boundary Value Problems.* Computerarithmetic, Scientific Computation and Programming Languages.

[10] R. Lohner, (1988) *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen.* Universität Karlsruhe.

[11] E. N. Lorenz, (1989) *Computational chaos – a prelude to computational instability.* Physica D **35**, p. 299 – 317 .

[12] R. E. Moore, (1966) *Interval analysis.* Prentice-Hall, Englewood Cliffs, NJ.

[13] R. E. Moore, (1978) *Methods and application of interval analysis.* SIAM Stud. App. Math.

[14] L. B. Rall, (1969) *Computational solution of nonlinear operator equation.* Robert E. Krieger publishing company, inc.

[15] D. Shiriaev, (1993) *Fast Automatic Differentiation for Vector Processors and Reduction of the Spatial Complexity in a Source Translation Environment.* Universität Karlsruhe.

[16] O. Stauning, (1994) *Interval Analyse i Dynamisk Systemteori.* Lyngby IMM-EKS-1994-24, Technical University of Denmark.