RIGOROUS NUMERICAL ANALYSIS WITH HIGH-ORDER TAYLOR MODELS

By

Jens Hoefkens

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Mathematics
and
Department of Physics and Astronomy

2001

ABSTRACT

RIGOROUS NUMERICAL ANALYSIS WITH HIGH-ORDER TAYLOR MODELS

By

Jens Hoefkens

Interval techniques have been utilized for rigorous numerical analysis since the 1960s. However, their use has been limited by the dimensionality curse and the dependency problem. The recently developed Taylor model approach alleviates these problems and allows the use of validated numerics in a wide range of applications. To broaden the applicability of the Taylor model method, we introduce new algorithms for the inversion of functional relations and the integration of differential algebraic equations.

First we present a new method for computing verified enclosures of the inverses of given functions over large domains. The approach utilizes Taylor models and the sharpness of the enclosures scales with a high order of the domain. An integral part of the new method is the rigorous determination of invertibility of high dimensional functions over large domains, which is reduced to a verified linear algebra problem involving only first derivatives of the function of interest. Several examples highlighting various aspects of the methods are discussed.

Differential algebraic equations (DAEs) describe important problems in mechanical and chemical engineering. Existing algorithms for the integration of DAE initial value problems have traditionally been restricted to low-index systems and until recently, no practical scheme for the verified integration of DAEs existed. Recognizing the antiderivation as a natural operation on Taylor models yields a method that treats DAEs within a fully differential algebraic context as implicit equations made of conventional functions and the antiderivation. The resulting integration scheme can be applied to high-index problems and allows the computation of guaranteed enclosures of final coordinates from large initial regions.

To demonstrate the general applicability of the Taylor model approach, we present results from verified asteroid orbit integrations and the theory of Hamiltonian systems. We show that the newly developed methods are practical and can indeed outperform conventional interval methods in a wide class of problems. Finally, we discuss some details of implementing interval libraries on general purpose computers and present the concept of language independent software development, which has been used for the design and implementation of the C++ and Fortran 90 interfaces to COSY Infinity.

To Soo, for her love.

ACKNOWLEDGMENTS

Most of all, I would like to thank my parents and my sister for their unconditional love. I am especially thankful to them for supporting me in my decision to leave Germany for the United States. I want them to know that I understand and appreciate their efforts and sacrifices. I also would like to thank Soo Chang for her never ending love, faith, and support over the last four years. Whenever life seems dull, she opens my eyes to the beauty of the world.

The field of interval analysis is small but has a close and friendly community of great scientists. I would like to thank the following colleagues for helpful advice and fruitful discussions over the years: Christian Bischof, George Corliss, Paul Hovland, Ken Jackson, Vladik Kreinovich, Rudolf Lohner, Ned Nedialkov, John Pryce, and Bill Walster. Special thanks goes to Ramon Moore for his ground breaking work on intervals. Working with him on the asteroid problem was one of the best experiences I had during my time in graduate school. However, my deepest thanks and gratitude go to my thesis advisor Martin Berz. He is a constant source of guidance and wisdom. He opened my eyes to the rigorous analysis of Taylor models and deserves my thanks for giving me the opportunity to study at MSU.

I also would like to thank those professors who have kindly served on my guidance committee: John Masterson, Jerry Nolen, Bradley Sherrill, Daniel Stump, and Clifford Weil. I have also enjoyed many discussions with friends and fellow students over the course of the last five years. In particular I would like to mention Jens von Bergmann, Sen Cheng, Jennifer Church, Lars Diening, Bela Erdélyi, Holger Harreis, Michael Lindemann, Kyoko Makino, Declan Mulhall, Cristian Opazo, Khodr Shamseddine, and Ralf Tönjes.

For financial support of my research I owe thanks to the Studienstiftung des deutschen Volkes, the US Department of Energy, the Alfred P. Sloan Foundation, and the National Science Foundation. I am grateful for their continuous support of young researchers and basic science.

Finally, special thanks goes to my father for making me stay in school when I was ready to graduate with a Masters degree. Without him, this dissertation would have never been written.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Over the last four decades, the use of computer programs has gained widespread acceptance in almost all scientific and engineering disciplines. An implicit trust in the computed results is based on the assumptions that computer programs are implemented correctly and perform as intended. However, the validity of these assumptions is questionable, to say the least.

Considering the fallibility of humans, it is presumptuous to believe that software could be produced free of errors. Moreover, since verifying the correctness of implementations is equivalent to the halting problem; it is generally impossible to prove that a given computer program does not contain bugs [117]. While elaborate testing procedures can reduce the number of errors, even the most important and extensively tested software systems can fail. As an example, consider the explosion of an Ariane 5 rocket in 1996 which has been traced to a very simple programming error [77].

However, even assuming that software could be written free of bugs, other sources of errors still exist. Most of these errors are caused by the transition from the perfect world of numerical analysis to the limited world of finite state machines. While numerical analysis often assumes the existence of infinitely many real numbers and infinitely precise computations, computers have only finite memory and floating point numbers are stored with only finitely many digits, resulting in computations that are only approximations to the mathematically correct results.

Most modern computers implement double precision floating point numbers with approximately 16 significant digits. While this is sufficiently accurate in most applications, the resulting rounding and truncation errors can accumulate and lead to arbitrary large deviations from the mathematically correct results. Moreover, while humans prefer decimal numbers, computers usually store information in binary representations [48]. The conversion from decimal numbers to binary numbers, combined with the limited accuracy of the number representations, can be a significant source of truncation errors. To illustrate this, consider the infinite binary representation of 0.1:

$$0.1_{10} = 0.001\,\overline{1001}_2. \tag{1.1}$$

On a computer with 24 bit floating point numbers, the error caused by the truncation amounts to an absolute error of approximately $9.5 \cdot 10^{-8}$. While this seems insignificant, the high failure rates of the Patriot missile system during the *Gulf War* in 1991 have been traced to exactly this truncation error [141].

It is important to note that the problem of roundoff and truncation errors is a fundamental characteristic of finite state computers. While intimate knowledge about the underlying architecture and significant efforts in the implementation of computer programs could reduce the effects of these errors, such errors can never be completely eliminated. Moreover, the extra effort necessary would require a software development model contradicting the idea of using portable high-level programming languages that help the programmer focus on the correctness of the implementation.

Thus, on the one hand we desire the portable programming of general purpose, limited precision binary architectures, on the other hand we know from theory and practice that this combination can lead to significant computational errors. Nevertheless, computers are here to stay and their use is likely to increase. Moreover, computational results are increasingly used as the basis of important decisions, ranging from the operation of nuclear power plants to the world financial markets. In other words, potentially erroneous results are used to control systems of global importance.

Somewhat contrary to naive intuition, the main problem with these numerical inaccuracies is that most computed results are actually sufficiently good approximations of the mathematically correct results and only a very small number of cases exhibit significant errors that require further investigation. It is the goal of *validated methods* to solve this dilemma by providing the users with bounds that are guaranteed to contain the mathematically correct result. While validated methods do not magically increase the accuracy of computations, they provide a self-validating approach that computes both an approximation to the mathematically correct result and a rigorous upper bound on the error of the approximation. In most situations these are relatively tight bounds and only in a few cases will they be large, indicating that the computed results may be rather bad approximations of the correct results.

While the fundamental ideas behind validated methods have already been introduced in the dissertation of R. Moore in 1962 [91], early methods have often computed overly large bounds. Only recently, starting with work by R. Lohner in 1987 [78], have validated methods been able to produce sharper and more usable bounds. In this dissertation we present new results from the field of *interval methods* [92], which lie at the core of the validated approach. The results are based on the theory and the application of the recently developed Taylor models [83, 82], which combine high order Taylor polynomials with intervals for validation. Taylor models can in some situations increase the sharpness of the computed bounds by several orders of magnitude over conventional methods, thereby improving the applicability of validated methods.

Chapter 2

Background Information

In this chapter we introduce the notational conventions that will be used throughout this dissertation. We also summarize the mathematical and computational theories that form the backbone of the material developed in this dissertation: the differential algebra ${}_nD_v$, interval analysis, and Taylor models.

Throughout this dissertations $\mathbb{N}$ denotes the set of positive integers, $\mathbb{Q}$ is the set of rational numbers, $\mathbb{R}$ stands for the set of real numbers, and $\mathbb{C}$ is the set of complex numbers. For notational convenience, we will always assume that $n, v, w \in \mathbb{N}$. Following the usual conventions, $\mathbb{R}^v$ denotes the set of v-dimensional vectors with real entries; elements of $\mathbb{R}^v$ are written as

$$x = (x_1, \ldots, x_v) \,, \tag{2.1}$$

with real numbers $x_i \in \mathbb{R}$ for $i = 1, \ldots, v$. If necessary, we denote vectors by $\vec{x}$ to distinguish them from real numbers x. However, in most cases, such a distinction will not be necessary. Finally, unless stated otherwise, all functions are assumed to be at least $(n+1)$-times continuously differentiable over their domain.

2.1 The Differential Algebra ${}_nD_v$

Many of the results presented in the following chapters rest on the availability of accurate descriptions of high-order Taylor polynomials of sufficiently smooth functions on computers. To avoid the need for storing an infinite number of Taylor coefficients, it is convenient to consider the equivalence classes of functions with the same n-th order Taylor polynomials. Since these are finite objects, the resulting structures can be implemented on computers, and they can be used for the efficient and accurate modeling and representation of complicated functions.

Let $U \subset \mathbb{R}^v$ be an open set containing the origin and consider the space $\mathcal{C}^{n+1}(U, \mathbb{R}^w)$ of $(n+1)$-times continuously differentiable functions that map U into $\mathbb{R}^w$. We define the relation of *equality up to order* n as follows.

Definition 2.1. *For $f, g \in \mathcal{C}^{n+1}(U, \mathbb{R}^w)$ we say that f equals g up to order n if $f(0) = g(0)$, and all partial derivatives of orders up to n agree at the origin. If f equals g up to order n, we denote that by $f =_n g$.*

It is easy to see that equality up to order n establishes an equivalence relation on the space $\mathcal{C}^{n+1}(U, \mathbb{R}^w)$ [17]. The resulting equivalence classes are called *DA vectors*, and the class containing the function $f \in \mathcal{C}^{n+1}(U, \mathbb{R}^w)$ is denoted by $[f]_n$. The collection of these equivalence classes is called ${}_nD_v$. More details on this structure are given in [12, 17].

Proposition 2.1. *For $f \in \mathcal{C}^{n+1}(U, \mathbb{R}^w)$, the n-th order Taylor polynomial $T_n(f)$ of f is contained in $[f]_n$.*

This assertion follows easily from the basic definition of the equivalence classes. However, the fact that the n-th order Taylor polynomial of f can be used as a representative for the class $[f]_n$ opens the door for a computer implementation of the structure ${}_nD_v$ by storing and manipulating the coefficients of Taylor polynomials.

2.1.1 Elementary Operations

Elementary operations like "+" and "×" can be lifted from $\mathcal{C}^{n+1}(U, \mathbb{R}^w)$ in the usual way, and extend to the corresponding operations "⊕" and "⊗" on ${}_nD_v$ [12, 13].

Definition 2.2. *Let $f, g \in \mathcal{C}^{n+1}(U, \mathbb{R}^w)$ be two functions. Then the* sum *of the DA vectors $[f]_n$ and $[g]_n$ is given by*

$$[f]_n \oplus [g]_n = [f + g]_n. \tag{2.2}$$

The product *of the two DA vectors is defined by*

$$[f]_n \otimes [g]_n = [f \times g]_n. \tag{2.3}$$

Together with the scalar multiplication $r \cdot [f]_n = [r \cdot f]_n$, this definition of the elementary operations makes ${}_nD_v$ an algebra [17]. Moreover, the diagram in Fig. 2.1 is closed and commuting. In other words, the extension of the elementary operations to ${}_nD_v$ is transparent to the equivalent classes of DA vectors; i.e., knowledge of the values and derivatives of f and g at the origin is sufficient to obtain Taylor polynomials of their sums and products. Moreover, in [12, 13] the available operations on ${}_nD_v$ have been extended to include subtraction and, for a limited class of DA vectors, even the multiplicative inversion. From now on we omit the distinction between the operations on $\mathcal{C}^{n+1}(U, \mathbb{R}^w)$ and ${}_nD_v$ and will always use the same symbols "+" and "×" for operations between numbers, functions, and DA vectors.

It has been shown that the derivative operation (derivation) can be extended from $\mathcal{C}^{n+1}(U, \mathbb{R}^w)$ to the algebra ${}_nD_v$ in such a way that ${}_nD_v$ becomes a *differential algebra* [12, 17]. While we will not use this intrinsic structure of ${}_nD_v$, we will make frequent use of the *antiderivation* of DA vectors to be presented later.

$$\begin{tikzcd}
f, g \arrow[r] \arrow[d, "(+,\cdot)"'] & {[f]_n, [g]_n} \arrow[d, "(\oplus,\odot)"] \\
f(+,\cdot)g \arrow[r] & {[f]_n(\oplus,\odot)[g]_n}
\end{tikzcd}$$

Figure 2.1: Commuting diagram for the elementary operations on ${}_nD_v$.

2.1.2 Contracting Operators and Fixed Points

In the previous section we have shown how elementary operations on the function space $\mathcal{C}^{n+1}(U, \mathbb{R}^w)$ can be extended to the differential algebra ${}_nD_v$. Here we take a look at the more general concept of *operators* on ${}_nD_v$ and summarize an important fixed point theorem. The availability of this powerful fixed point theorem for operators on ${}_nD_v$ allows the use of the differential algebra ${}_nD_v$ in a large class of numerical applications, ranging from the analysis of dynamical systems [15, 23, 27, 18] to global optimization [68, 88].

Definition 2.3. *For $[f]_n \in {}_nD_v$, the* depth $\lambda([f]_n)$ *is defined to be the order of the first non-vanishing derivative of f if $[f]_n \neq 0$, and $n+1$ otherwise.*

By definition of the equivalence classes, this definition is independent of the choice of $f \in [f]$. We note that any $a \in {}_nD_v$ with $\lambda(a) \geq 1$ satisfies the condition $a^{n+1} = 0$ and is therefore called *nilpotent.* Using the straightforward definition of the depth, contracting operators on ${}_nD_v$ are defined as follows.

Definition 2.4. *Let $\mathcal{O}$ be an operator defined on $M \subset {}_nD_v$. $\mathcal{O}$ is* contracting *on M,* *if for any two* $[f]_n, [g]_n \in M$,

$$\lambda(\mathcal{O}([f]_n) - \mathcal{O}([g]_n)) \geq \lambda([f]_n - [g]_n) \tag{2.4}$$

with equality if and only if $f =_n g$.

This definition has a striking similarity to the corresponding definitions on standard function spaces. Even more so, a theorem that resembles the Banach Fixed Point Theorem can be established on ${}_nD_v$. However, unlike in the case of the Banach Fixed Point Theorem, in ${}_nD_v$ the sequence of iterates is guaranteed to converge in at most $n+1$ steps [17].

Theorem 2.1 (DA Fixed Point Theorem). *Let* $\mathcal{O}$ *be a contracting operator and self-map defined on* $M \subset {}_nD_v$. *Then* $\mathcal{O}$ *has a unique fixed point* $a \in M$. *Moreover, for any* $a_0 \in M$ *it is* $\mathcal{O}^{(n+1)}(a_0) = a$.

A proof and further discussion of the DA Fixed Point Theorem can be found in [17]. Here we just mention that, since $\lambda(a+b) \geq \min(\lambda(a), \lambda(b))$, it follows easily that the sum and composition of two contracting operators $\mathcal{O}_1$ and $\mathcal{O}_2$ defined on M is also a contracting operator.

2.1.3 Functions and Antiderivation

To fully utilize the differential algebra ${}_nD_v$, especially in numerical analysis and computer environments, it is necessary to not only define the elementary operations on ${}_nD_v$ but also the standard mathematical functions commonly available on computers: square root, exponential, logarithm, trigonometric and hyperbolic functions.

At a fundamental level, the basic functions on ${}_vD_n$ are simply defined in terms of the corresponding operations on the function space [17].

Definition 2.5. *For* $f, g \in \mathcal{C}^{n+1}(U, \mathbb{R}^w)$ *we define* $g([f]_n) = [g(f)]_n$.

While this definition is straightforward and of great theoretical value, the actual computation of functions on ${}_nD_v$ is based on addition theorems and the Taylor series of these functions. Although the full details of this procedure are beyond the scope of this summary, the following example illustrates the general approach. Let $[f]_n \in {}_nD_v$ be a DA vector and write $[f]_n = a_0 + b$ with $a_0 = f(0)$. Then it is

$$\exp\left([f]_n\right) = \exp(a_0 + b) = \exp(a_0)\exp(b). \tag{2.5}$$

Using the definition of the exponential function as a power series and the fact that b is nilpotent, in fact

$$\exp\left([f]_n\right) = \exp(a_0) \cdot \sum_{k=0}^{\lfloor n/2 \rfloor} \frac{b^k}{k!}. \tag{2.6}$$

Further details on the implementation of functions of DA vectors can be found in [12, 13, 17].

The approach outlined above allows us for any function f, for which we have a code list or algorithm consisting of finitely many intrinsic functions and elementary operations, to obtain the n-th order Taylor polynomial of f around the origin. By starting the evaluation with the identity DA vectors $[id]_n$, we obtain $T_n(f)$ by evaluating the code list of f with the argument $[id]_n$. This gives a convenient and powerful method of computing derivatives of functions described by computer programs. Unlike conventional automatic differentiation [111, 52], which is often limited to first and second order, the Taylor series approach does not pose any arbitrary limits on the maximum order of derivatives that can be computed.

We conclude this section on functions on ${}_nD_v$ with an example of an operator that is unusual but, considering the structure of the differential algebra ${}_nD_v$, actually quite natural. The antiderivation; i.e., the integration with respect to any of the v variables, turns out to be a contracting operation on ${}_nD_v$.

Proposition 2.2 (Antiderivation is Contracting). *For $k \in \{1, \ldots, v\}$, the* antiderivation $\partial_k^{-1} : {}_nD_v \to {}_nD_v$ *is a contracting operator on* ${}_nD_v$.

The proof of this important result is based on the fact that if $a, b \in {}_nD_v$ agree up to order l, the first non-vanishing derivative of $\partial_k^{-1}(a - b)$ is of order $l + 1$ [17, 59, 60]. It is important to realize that in the DA framework of ${}_nD_v$ there is no fundamental difference between any of the standard mathematical functions like the sine and exponential functions on ${}_nD_v$ and antiderivation. In fact, fully embracing antiderivation as a normal operation on DA vectors will enable us to develop a new and powerful method for the verified integration of ordinary differential equations (ODEs) and differential algebraic equations (DAEs) in Chap. 4.

2.1.4 Summary

For functions that are given by finitely many intrinsic functions and elementary operations, the DA approach is equivalent to evaluating the code list with an n-th order automatic differentiation tool. However, the ease with which the DA approach computes Taylor polynomials, and therefore derivatives, of any given algorithm up to machine precision and virtually arbitrary order makes the differential algebra ${}_nD_v$ an important computational tool for numerical analysis. Together with the powerful DA Fixed Point Theorem that guarantees convergence in at most $n+1$ steps, the method lies at the core of the *map approach* that has been used successfully in the analysis and the design of particle accelerators [10, 12, 14, 15, 17]. The differential algebra ${}_nD_v$ has been implemented in the arbitrary order code COSY Infinity [13, 23, 84]. And by storing only non-vanishing coefficients, the implementation can handle even high-dimensional problems to very high orders.

By combining the symbolic nature of operations on truncated polynomials with the numerical operations on floating point coefficients, differential algebra based methods offer a unique combination of exact symbolic operations with the efficiency of floating point computations. Moreover, the DA approach avoids the pitfalls of conventional computer algebra, since it does not suffer from the dramatic increase in storage that often plagues symbolic computer algebra tools in high-order applications. At the same time, propagating high order descriptions of computer code avoids many of the problems associated with traditional numerical methods and opens the door for rigorous high-order sensitivity analysis of dynamical systems [104].

2.2 Interval Arithmetic

The use of interval methods in computational sciences was started by R. Moore [91], who observed that,

> "Computers carry only a limited number of significant digits. Repeating a computer calculation with more significant digits does not necessarily increase the number of significant digits in the result."

To illustrate this problem, consider the sequence of numbers defined recursively by

$$x_0 = 1 - 10^{-21} \quad \text{and} \quad x_{n+1} = {x_n}^2. \tag{2.7}$$

If we compute this sequence on a computer with 10, 16, or even 20 digit accuracy, it is

$$x_0 = x_1 = \ldots = x_{75} = 1, \tag{2.8}$$

while in fact $x_{75} < 10^{-10}$. Since the question in numerical computations is often how the result compares to some fixed number, this is actually an important problem and computational errors like the above are relatively frequent. These computational errors have many sources: floating point errors, rounding errors, truncation, and initial errors. And since they are hard to identify at the time of actual computation, they are often undetected and can have dire consequences.

The previous example illustrates the main problem of naively using computers to implement results of numerical analysis. Due to the fact the computers can only represent a small subset of the rational numbers $\mathbb{Q}$ accurately, the results obtained by computer operations are generally only approximations of the mathematically correct results. While traditional methods of numerical analysis are very powerful and provide rigorous results in an ideal world, their transformations to computer systems suffer from the intrinsic limitations of finite state machines.

However, modern interval techniques use knowledge of these limitations to compute results that are both accurate and reliable. In this section we summarize the fundamentals of conventional interval methods. The development of powerful new interval techniques is the main focus of this dissertation.

Interval Notations

Before presenting details of interval analysis, we introduce notation and conventions that will be used in connection with intervals throughout the text. For two numbers $a, b \in \mathbb{R}$, the interval $[a, b]$ is defined by

$$[a, b] = \{x \in \mathbb{R} \,|\, a \leq x \leq b\}\,. \tag{2.9}$$

Unless explicitly stated otherwise, intervals will always be assumed to be closed and bounded and therefore compact. Whenever we talk about intervals in general, we use boldface to distinguish them from regular numbers: X is a point, while $\boldsymbol{X}$ denotes a compact interval.

To express containment of points and intervals within intervals, we use the standard notation

$$x \in [a,b] \Leftrightarrow a \leq x \leq b \tag{2.10}$$

$$[a,b] \subset [c,d] \Leftrightarrow c \leq a \leq b \leq d \tag{2.11}$$

The union and intersection of two intervals are defined by

$$[a,b] \cup [c,d] \quad = \quad \{x \in \mathbb{R} \,|\, x \in [a,b] \text{ or } x \in [c,d]\} \tag{2.12}$$

$$[a,b] \cap [c,d] \quad = \quad \{x \in \mathbb{R} \,|\, x \in [a,b] \text{ and } x \in [c,d]\} \tag{2.13}$$

The union of two intervals will generally not be an interval. The intersection of two intervals on the other hand is either an interval or empty. Lastly, we denote the midpoint and the width of an interval by $m\left([a,b]\right)$ and $w\left([a,b]\right)$, respectively.

In the case of $\mathbb{R}^v$ with $v > 1$, we define interval boxes to be vectors of intervals and all previously mentioned operations and relations are defined componentwise. The width w of an interval box is called the magnitude and is defined as the maximum of the widths of its components.

2.2.1 Elementary Operations and Functions

We define elementary operations on intervals set-wise such that they satisfy the fundamental inclusion requirement: for $* \in \{+, -, \times, /\}$ we demand that

$$[a,b] * [c,d] = \{x * y \,|\, a \leq x \leq b, c \leq y \leq d\} \tag{2.14}$$

In other words, the sum, difference, product, or quotient of two intervals is the set of sums, differences, products, or quotients of pairs of real numbers, one from each of the two intervals. All elementary operations can be defined in such a way that the

$$\begin{tikzcd}
x, y \arrow[r, hook] \arrow[d, "{(+, \times, \ldots)}"'] & \boldsymbol{I_x}, \boldsymbol{I_y} \arrow[d, "{(\oplus, \otimes, \ldots)}"] \\
x(+, \times, \ldots)y \arrow[r, hook] & \boldsymbol{I_x}(\oplus, \otimes, \ldots)\boldsymbol{I_y}
\end{tikzcd}$$

Figure 2.2: Commuting diagram for the elementary operations on intervals.

results of operations between intervals are again intervals:

$$\begin{aligned}
\boldsymbol{I_x} \oplus \boldsymbol{I_y} &= [x_1 + y_1, x_2 + y_2] && (2.15)\\
\boldsymbol{I_x} \ominus \boldsymbol{I_y} &= [x_1 - y_2, x_2 - y_1] && (2.16)\\
\boldsymbol{I_x} \otimes \boldsymbol{I_y} &= [\min\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}, \max\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}] && (2.17)
\end{aligned}$$

If the interval $\boldsymbol{I_y}$ does not contain 0, the multiplicative inverse is given by

$$\boldsymbol{I_y}^{-1} = [1/y_2, 1/y_1] \tag{2.18}$$

and the quotient of two intervals is defined in terms of the multiplicative inverse and multiplication:

$$\boldsymbol{I_x} \oslash \boldsymbol{I_y} = \boldsymbol{I_x} \otimes \boldsymbol{I_y}^{-1}. \tag{2.19}$$

With these definitions, the diagram in Fig. 2.2 closes and commutes under the inclusion relationship. From now on, we will denote the elementary operations on intervals and real numbers by the same symbols.

One problem of interval arithmetic is the *dependency problem.* It is best illustrated with the interval $\boldsymbol{I} = [-1, 1]$ and the computation

$$\boldsymbol{I} - \boldsymbol{I} = [-2, 2]. \tag{2.20}$$

Thus, the width of $\boldsymbol{I} - \boldsymbol{I}$ is twice as large as the width of the original interval $\boldsymbol{I}$. This problem, known as the *cancellation effect*, is a manifestation of the more general problem that conventional interval arithmetic has no provision to distinguish between independent and dependent variables. As far as interval analysis is concerned, the two intervals on the left hand side of Eqn. (2.20) are two different entities with no relation between them, and the fact that they have the same endpoints is seen as a mere coincidence. The dependency problem especially affects large computations, since not all occurrences of the dependency problem are as easy to spot as in this example. We note that work is under way to include techniques for resolving the dependency problem within Fortran and C++ compilers [136, 137], where the optimizers already perform a dependency analysis that is capable of also reducing the dependency problem. We mention that even conventional floating point arithmetic is beset by cancellation effects: on a computer with less than twenty digits of accuracy

$$\left(10^{20} + 1\right) - 10^{20} = 0. \tag{2.21}$$

This exemplifies a general problem of floating point arithmetic: computing the difference of two similar numbers often results in large relative errors. And although interval computations always enclose the correct results, the cancellation effect can lead to significant overestimations.

Similar to the way the elementary operations are defined on intervals to satisfy Eqn. (2.14), the standard mathematical functions for intervals are defined such that the basic requirement

$$f([a, b]) = \{f(x) \mid a \leq x \leq b\} \tag{2.22}$$

is always satisfied. In many cases the computation of interval extensions of mathematical functions is straightforward and details will be discussed in Sec. 6.1.

2.2.2 Interval Arithmetic and Set Theory

Since interval arithmetic has its roots in the desire to compute rigorous results with limited precision floating point hardware, it has originally been used to model small intervals. However, by realizing that the interval $[a, b]$ is actually the *set of real numbers* between a and b, where a and b can be rational numbers with exact floating point representations, computer programs become able to accurately handle real numbers. Thus, by using intervals to represent sets of real numbers, intervals bring set theory and the ability to represent *any* real number to computers.

Utilizing the elementary operations and functions defined in the previous section, we can obtain the Fundamental Theorem of Interval Analysis [94].

Theorem 2.2 (Fundamental Theorem of Interval Analysis). *Let $\boldsymbol{I} = [a, b]$ be a compact real interval. If $f : \boldsymbol{I} \to \mathbb{R}$ is a continuous function and $\boldsymbol{f}$ is its inclusion monotone interval extension, then*

$$x \in [a, b] \;\Rightarrow\; f(x) \in \boldsymbol{f}\left(\boldsymbol{I}\right). \tag{2.23}$$

The Fundamental Theorem of Interval Analysis allows the use of computers, despite their limitation, to rigorously answer the question:

> Given a function f described by a computer program, consisting of finitely many elementary operations and intrinsic functions, and an interval $\boldsymbol{I}$, find a set of real numbers that is guaranteed to contain the image of $\boldsymbol{I}$ under the function f.

By allowing the computation of intervals that are guaranteed to contain the range $\boldsymbol{f}\left(\boldsymbol{I}\right)$ of f over $\boldsymbol{I}$, the use of interval analysis allows the use of computers with finite precision to obtain rigorous and trustworthy results.

Some of the most powerful aspects of interval computations are tied to the Brouwer Fixed Point Theorem, which is the finite-dimensional version of the Schauder Fixed Point Theorem:

Theorem 2.3 (Brouwer Fixed Point Theorem). *Let $\boldsymbol{D}$ be a non-empty compact convex subset of $\mathbb{R}^v$ and suppose f is a continuous mapping such that the range $\boldsymbol{f}(\boldsymbol{D}) \subset \boldsymbol{D}$. Then f has a fixed point in $\boldsymbol{D}$, i.e. there is $x \in \boldsymbol{D}$ such that $f(x) = x$.*

Since interval boxes satisfy the requirements on $\boldsymbol{D}$, the Brouwer Fixed Point Theorem can be combined with the Fundamental Theorem of Interval Analysis to allow computer based proofs of the existence of solutions to linear and non-linear systems. One particular important application of this, the basic interval Newton method, will be discussed in Sec. 2.2.3.

The Wrapping Effect

Much like the dependency problem, the *wrapping effect* can lead to substantial overestimations in interval computations. It is caused by the need to *wrap* the results of interval computations in interval boxes that are parallel to the axes. The effect is demonstrated in Fig. 2.3: a rotation by 45 degrees leads to an overestimation of the magnitude of the result by a factor of $\sqrt{2}$.

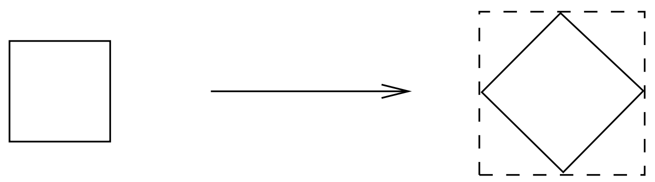


Figure 2.3: Illustration of the wrapping effect caused by a rotation of 45 degrees in the plane.

While efforts have been made to fight the wrapping effect by allowing rotated and even skewed interval boxes [78, 79], the fundamental problem remains and Fig. 2.4

illustrates how the fact that the edges of interval boxes are always given by straight lines can lead to arbitrary large overestimations of the image sets.

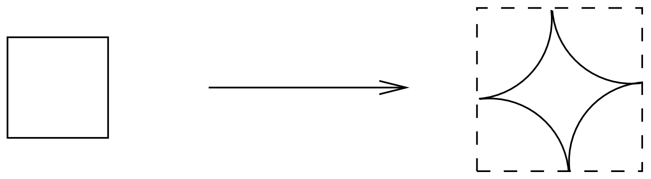


Figure 2.4: Illustration of how the wrapping effect can lead to arbitrary large overestimations.

2.2.3 Applications of Interval Arithmetic

By utilizing numerical analysis tools that have been developed specifically for interval analysis, interval methods have been used successfully in a variety of applications over the last decades. In this section we take a closer look at some of the most important of these algorithms and applications: interval Newton methods, optimization, and integration of ODEs.

Interval Newton Method

Interval Newton methods are among the most fundamental and important interval algorithms. They are, in one form or another, part of almost any other interval algorithm. Since they utilize most of the concepts of other interval methods, e.g. fixed point formulations, iteration, and intersections, they are outstanding examples of the more general interval analysis tools.

Let $f : [a, b] \to \mathbb{R}$ be a $\mathcal{C}^1$ function such that either $f' > 0$ or $f' < 0$ on $[a, b]$ and assume that $\exists\, x \in [a, b]$ such that $f(x) = 0$. The goal of an interval Newton method is to compute a small interval enclosure of x. According to the Mean Value Theorem, for any $y \in [a, b]$ there is some $\xi \in [a, b]$ such that

$$0 = f(x) = f(y) + f'(\xi)(x - y). \tag{2.24}$$

After choosing $y = m([a,b])$ and solving for x, it is

$$x = m([a,b]) - \frac{f(m([a,b]))}{f'(\xi)}. \tag{2.25}$$

with some unknown $\xi \in [a,b]$. However, if we use intervals to evaluate the interval extension of the derivative f', we can guarantee that $f'(\xi) \in \boldsymbol{f'}([\boldsymbol{a},\boldsymbol{b}])$. Assuming that $0 \notin \boldsymbol{f'}([\boldsymbol{a},\boldsymbol{b}])$, it follows that

$$x \in m([a,b]) - \frac{f(m([a,b]))}{\boldsymbol{f'}([\boldsymbol{a},\boldsymbol{b}])}. \tag{2.26}$$

While $f(m([a,b]))$ and $m([a,b])$ in the last equation are points, $\boldsymbol{f'}([\boldsymbol{a},\boldsymbol{b}])$ is an interval and the right hand side is therefore itself an interval. This last equation leads to the following definition of the *Newton operator* N for an interval $\boldsymbol{I} \subset [a,b]$:

$$N(\boldsymbol{I}) = m(I) - \frac{f(m(\boldsymbol{I}))}{\boldsymbol{f'}(\boldsymbol{I})} \tag{2.27}$$

With the Newton operator, the foundation of the standard interval Newton method is given by Thm. 2.4; more details of interval Newton methods can be found in [92, 94, 3, 57].

Theorem 2.4. *For $\boldsymbol{X}_0 = \boldsymbol{I}$, define a sequence of intervals by*

$$\boldsymbol{X}_{n+1} = N(\boldsymbol{X}_n) \cap \boldsymbol{X}_n. \tag{2.28}$$

If $\boldsymbol{I}$ contains a zero of f, $\boldsymbol{X}_n$ will contain the zero for any n. Moreover, if for some n the intersection $N(\boldsymbol{X}_n) \cap \boldsymbol{X}_n$ is empty, $\boldsymbol{I}$ did not contain a zero of f.

It is important to note that intersecting the result of the Newton operator with the previous enclosure ensures that the elements of the sequence never increase in size. Thus, unlike in the regular Newton method where the sequence of iterates might diverge away from a zero, the sequence of magnitudes $w(\boldsymbol{X}_n)$ is monotonically decreasing.

Optimization

Interval algorithms for constrained and unconstrained optimization are based on an exhaustive search of the search initial region. They usually follow the general scheme for Lipschitz optimization [106], but use interval evaluations of the objective function instead of estimations based on Lipschitz constants.

The general problem of constrained and unconstrained optimization is given by

$$\text{minimize: } \phi(x) \tag{2.29a}$$

$$\text{such that: } c(x) = 0 \tag{2.29b}$$

$$g(x) \leq 0. \tag{2.29c}$$

$\phi : \boldsymbol{D} \subset \mathbb{R}^v \to \mathbb{R}$ is the *objective* function; $c : \boldsymbol{D} \to \mathbb{R}^k$ and $g : \boldsymbol{D} \to \mathbb{R}^l$ are constraint conditions. If $l \geq 2$, condition (2.29c) is understood to apply to each of the components of g. If $k = l = 0$, the problem is considered to be unconstrained.

The basic algorithm for the exhaustive search starts by dividing the initial search region $\boldsymbol{D}$ into sub-domains and maintaining a list of boxes where the minimum *might* be. The boxes in the list are repeatedly tested and split further if necessary. While this by itself leads to an exponential growth of the number of boxes, the main idea of the algorithm is to speed up the search by rejecting boxes from the list; i.e., to rigorously determine that the minimum *cannot* be inside a particular box. Interval methods play an important part in this algorithm. They are used in storing information on possible regions in the list and, more importantly, they are used to compute verified range enclosures of the objective function to possibly reject boxes from the list of candidate boxes or to verify that the box contains the global minimum. More details of such algorithms can be found in [114, 68, 56].

Sophisticated extensions of the basic branch and bound algorithms exist that use interval methods to improve the decision process of rejecting boxes [115, 34]. However, while global optimization with intervals has found successful applications [37], it often struggles due to the exponential growth in the number of candidate boxes. This is especially true for complicated multidimensional objective functions [21]. More details on constrained and unconstrained optimization can be found in [140, 29, 41, 55, 63, 95, 38]. We would like to highlight the GlobSol package [67, 68, 46] as one particular sophisticated implementation of global optimization algorithms with intervals.

ODE Initial Value Problems

Consider the general explicit first order ODE initial value problem

$$x' = f(t,x),\ \ x(t_0) = x_0 \tag{2.30}$$

with a sufficiently smooth function f defined on a suitable subset of $\mathbb{R} \times \mathbb{R}^v$. Interval methods for the integration of ODE initial value problems provide enclosures for the final positions, such that at every time step t_i the actual solution to the initial value problem is contained in the solution interval. To achieve that goal, the integration methods take extended initial conditions, mathematical truncation, and roundoff errors into account.

However, mostly due to the wrapping effect discussed in Sec. 2.2.2, methods for the integration of ODE initial value problems are among the most demanding problems for designers of interval algorithms [98]. While the first attempts at interval based integration tools [92] have struggled with the propagation of large initial regions even over small time intervals, newer methods like AWA use changes of variables, ellipsoid interval boxes and other techniques [79, 101] to fight the wrapping effect with varying degrees of success. But only recently have interval-based techniques been developed that alleviate the problem of the wrapping effect in the integration of ODE initial value problems [22, 82]. These methods will be summarized in Sec. 2.3.3 and their application to solar system dynamics will be presented in Sec. 5.1. By successfully avoiding the wrapping effect to high orders, the new approach allows the long term propagation of large initial condition sets without substantial overestimations.

2.2.4 Summary

Interval arithmetic is arithmetic defined on sets of numbers instead of single numbers and its results have been connected to modern computing machinery and numerical analysis in 1962 by R. Moore [91]. The use of interval arithmetic allows rigorous computations of verified results on computers by accounting for the effects of finite precision and roundoff errors, mathematical truncation, and extended sets of starting values. As such, interval computations overcome the general limitations of finite state machines and bring the strict mathematical rigor of numerical analysis and set theory to the computational sciences. Good introductions and overviews to interval analysis can be found in [92, 94, 3].

Interval methods have been used successfully in a wide range of applications: chemical and electrical engineering [102, 5], computer graphics [96], dynamical systems and chaos [50, 124], computer assisted proofs [74, 75], and expert systems [70] to name only a few. However, the wrapping effect, the dependency problem, and the dimensionality curse prevent interval methods from widespread acceptance in the computational sciences.

2.3 Taylor Models

Intervals are well suited to model enclosures of real numbers. However, since they only propagate information about function values and neglect any further knowledge about derivatives, they are not well suited for the rigorous enclosure of functions. In fact, modeling functions with intervals usually requires splitting the domain $\boldsymbol{D}$ into a collection $\{\boldsymbol{D}_i\}$ of sufficiently small subdomains and evaluating the functions on each of the $\boldsymbol{D}_i$s. Then the "evaluation" of f consists of finding the appropriate interval enclosure $\boldsymbol{f}(\boldsymbol{D_i})$ of the range of f over the subdomain $\boldsymbol{D}_i$.

While splitting a one-dimensional interval is usually acceptable, splitting higher dimensional interval boxes results in an exponential increase of the number of subdomains. This phenomenon is known as the *dimensionality curse.* It frequently plagues conventional interval techniques and limits their applicability to large, both in terms of dimensionality and domain size, problems.

Taylor models offer a remedy to this problem by combining high-order Taylor polynomials with real number coefficients and intervals for verification. Taylor models allow the rigorous modeling of complicated multidimensional functions over domains that are usually several orders of magnitude larger than the ones over which conventional interval methods can work with the same sharpness.

Definition 2.6 (Taylor Model). *Let $\boldsymbol{D} \subset \mathbb{R}^v$ be a box with $x_0 \in \boldsymbol{D}$. Let $P : \boldsymbol{D} \to \mathbb{R}^w$ be a polynomial of order n and $R \subset \mathbb{R}^w$ be an non-empty convex compact set. Then $(P, x_0, \boldsymbol{D}, R)$ is called a Taylor model of order n with expansion point x_0 over $\boldsymbol{D}$.*

Following these notations, P is called the *reference polynomial*, x_0 is the *expansion or reference point*, and R is called the *remainder bound* of the Taylor model. For the rigorous modeling of functions on computers, we usually view Taylor models as subsets of function spaces by virtue of the following definition.

Definition 2.7 (Taylor Models as Sets of Functions). *Let $T = (P, x_0, \boldsymbol{D}, R)$ be an order n Taylor model. Then, identify T with the set of functions $f \in \mathcal{C}^{n+1}(\boldsymbol{D}, \mathbb{R}^w)$*

such that $f(x) - P(x) \in R$ for all $x \in \boldsymbol{D}$, and the n-th order Taylor series of f around x_0 equals P. Furthermore, if a $\mathcal{C}^{n+1}$ function f is contained in a Taylor model T, we call T a Taylor model for f.

To illustrate the concept of Taylor models, consider two Taylor models T_1 and T_5 of orders one and five given by

$$T_1 = (x, 0, [-1.5, 1.5], (-1.122182, 1.122182)) \tag{2.31}$$

$$T_5 = \left(x - x^3/3! + x^5/5!, 0, [-1.5, 1.5], (-0.015781, 0.015781)\right) \tag{2.32}$$

The two Taylor models T_1 and T_5 have been computed as Taylor models for the sine function with COSY Infinity [23, 26] and are shown together with their remainder bounds in Fig. 2.5. While the first order Taylor model has remainder bounds that are similar in size to the domain interval; i.e., the first order reference polynomial is a rather bad approximation of the sine function over the domain, the fifth order Taylor model encloses the sine function with a sharpness that is of the order of one percent of the domain size. We mention that the requirements on the Taylor expansion of the functions contained in a Taylor model are sometimes omitted, and Taylor models are then simply viewed as sets of $\mathcal{C}^{n+1}$ functions that are R-close to the reference polynomial. However, these situations are rare and will always be mentioned explicitly.

Similar to the way intervals allow the rigorous implementation of set theory and numerical analysis on computers, Taylor models allow the verified representation of arbitrary smooth functions within computer programs. As such, Taylor models combine the mathematically rigorous results of functional analysis with the approximative floating point representations on computers to obtain methods to rigorously compute analytical results.

For purposes of illustration, Figure 2.6 compares the enclosure of a function by

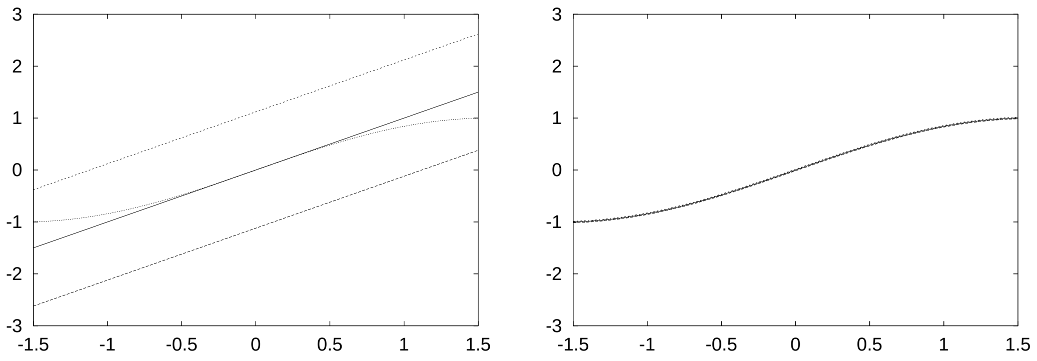


Figure 2.5: Taylor models of order one and five for the sine function over the domain interval $[-1.5, 1.5]$.

Taylor models and regular intervals. In this case, the virtue of the method lies in the fact that a single Taylor model over a relatively large domain box guarantees a sharpness that interval bounding cannot achieve, even with multiple smaller domains. This helps significantly in fighting the dimensionality curse inherent in interval bounding where it is of the utmost importance to avoid subdivisions of the domain boxes, due to the otherwise exponential growth in the number of subdomains.

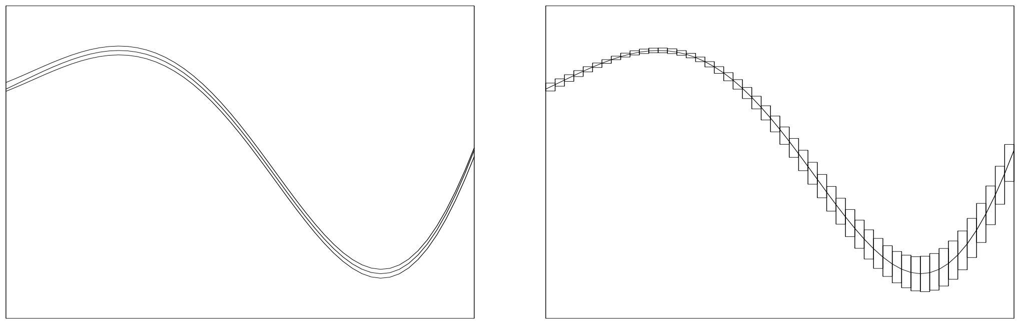


Figure 2.6: Left: Enclosing a function by a Taylor model of order eight; right: Interval bounding of the same function with 50 sub-intervals.

We note that the actual implementation of Taylor models and related operations in COSY Infinity is mostly due to K. Makino [82]. Since the definition of Taylor models is so intimately related to Taylor polynomials, the implementation of Taylor models rests on the foundations laid by the differential algebra package and the interval implementation to be discussed in Sec. 6.1.

2.3.1 Arithmetic and Operations

Methods have been developed for arithmetic operations on Taylor models that preserve the defining inclusion relationships. Hence these methods allow the computation of Taylor models for any sufficiently smooth computer function. In the following, let P and $\boldsymbol{D}$ be as above and denote by $B(P, \boldsymbol{D})$ a guaranteed enclosure of the range of the polynomial function P over the box $\boldsymbol{D}$. Moreover, for $k \in \mathbb{N}$, $P_{(k)}$ and $P_{(k+)}$ are the parts of P of orders up to k and greater than k, respectively. Using this notation, the next theorem summarizes results presented in [82] and shows how elementary operations can be defined on Taylor models.

Theorem 2.5. *Let $T_1 = (P_1, x_0, \boldsymbol{D}, R_1)$ and $T_2 = (P_2, x_0, \boldsymbol{D}, R_2)$ be two Taylor models of order n and define*

$$R_P = R_1 \cdot R_2 + R_1 \cdot B(P_2, \boldsymbol{D}) + B(P_1, \boldsymbol{D}) \cdot R_2 + B((P_1 \cdot P_2)_{(n+)}, \boldsymbol{D}). \tag{2.33}$$

Obtain new Taylor models T_S and T_P by

$$T_S = (P_1 + P_2, x_0, \boldsymbol{D}, R_1 + R_2), \tag{2.34}$$

$$T_P = \left((P_1 \cdot P_2)_{(n)}, x_0, \boldsymbol{D}, R_P\right). \tag{2.35}$$

Then, T_S and T_P are Taylor models for the sum T_S and product T_P of T_1 and T_2. In particular, for two functions $f_1 \in T_1$ and $f_2 \in T_2$, it is

$$(f_1 + f_2) \in T_S \text{ and } (f_1 \cdot f_2) \in T_P. \tag{2.36}$$

Proof. If we define $\mathcal{C}^{n+1}$ functions $\delta_1 = f_1 - P_1$ and $\delta_2 = f_2 - P_2$, then $\delta_1(x) \in R_1$ and $\delta_2(x) \in R_2$ for any $x \in \boldsymbol{D}$. Thus, for a given $x \in \boldsymbol{D}$

$$((f_1 + f_2) - (P_1 + P_2))(x) = \delta_1(x) + \delta_2(x) \in R_1 + R_2 = R_S. \tag{2.37}$$

Since the n-th order Taylor expansion of the sum $f_1 + f_2$ equals the sum of the Taylor series, T_S is indeed a Taylor model for the sum $T_1 + T_2$. Also, over the domain $\boldsymbol{D}$

$$\begin{aligned} \big((f_1 \cdot f_2) - (P_1 \cdot P_2)_{(n)}\big) &= ((P_1 + \delta_1)(P_2 + \delta_2)) - \big((P_1 \cdot P_2) - (P_1 \cdot P_2)_{(n+)}\big) \\ &= P_1 \cdot \delta_2 + \delta_1 \cdot P_2 + \delta_1 \cdot \delta_2 + (P_1 \cdot P_2)_{(n+)}. \end{aligned} \tag{2.38}$$

Moreover, since the n-th order Taylor polynomial of $f_1 \cdot f_2$ equals the polynomial product $(P_1 \cdot P_2)_{(n)}$, T_P is a Taylor model for the product $T_1 \cdot T_2$. $\square$

The general theory of arithmetic on Taylor models has been developed in [83, 22, 82] and all elementary operations have been defined such that the basic inclusion properties are maintained throughout the arithmetic. In particular, all operations are inclusion monotone and lead to closed commuting diagrams as shown in Fig. 2.7.

$$\begin{tikzcd} f, g \arrow[r, hook] \arrow[d, "{[+, \times]}"'] & T_f, T_g \arrow[d, "{[\oplus, \otimes]}"] \\ f[+, \cdot]g \arrow[r, hook] & T_f[\oplus, \otimes]T_g \end{tikzcd}$$

Figure 2.7: Commuting diagram for elementary operations on Taylor models.

As with the elementary operations, it is possible to extend the standard mathematical functions to Taylor models such that the fundamental inclusions are maintained; the effect of this is illustrated by the closed commuting diagram for the sine function shown in Fig. 2.8. The actual implementation of the intrinsic functions uses an approach resembling the way functions are defined on ${}_nD_v$ to obtain Taylor models for the standard mathematical functions available on computers. More details on the definition and implementation of intrinsic functions on Taylor models can be found in [83, 22, 82].

$$\begin{tikzcd}
f \arrow[r, hook] \arrow[d, "{[\sin(\,\cdot\,)]}"'] & T_f \arrow[d, "{[\sin(\,\cdot\,)]}"] \\
\sin(f) \arrow[r, hook] & \sin(T_f)
\end{tikzcd}$$

Figure 2.8: Commuting diagram for sine function on Taylor models.

Last but not least, we define the antiderivation of Taylor models, which is a particularly powerful operation with important applications in verified Taylor model computations.

Definition 2.8. *For an n-th order Taylor model $T = (P, x_0, \boldsymbol{D}, R)$ and $k = 1, \ldots, v$, let*

$$Q_k = \int_0^{x_k} P_{(n-1)}\left(x_1, \ldots, x_{k-1}, \xi_k, x_{k+1}, \ldots, x_v\right) d\xi_k. \tag{2.39}$$

The antiderivative ∂_k^{-1} of T is defined by

$$\partial_k^{-1}(T) = \left(Q_k, x_0, \boldsymbol{D}, \left(\boldsymbol{B}(P_{(n)} - P_{(n-1)}, \boldsymbol{D}) + R\right) \cdot \boldsymbol{B}(x_k, \boldsymbol{D})\right). \tag{2.40}$$

Since Q_k is of order n, the definition assures that for a n-th order Taylor model T, the antiderivative $\partial_k^{-1}(T)$ is again a n-th order Taylor model. Moreover, since all terms of P of exact order n are bound into the new remainder, this definition of the antiderivation is inclusion monotone and lets the diagram shown in Fig. 2.9 commute.

As in the case of the differential algebra ${}_nD_v$, it is noteworthy that the antiderivation does not fundamentally differ from other intrinsic functions on Taylor mod-

$$\begin{tikzcd}
f \arrow[r, hook] \arrow[d, "\int_0^{x_k}"'] & T_f \arrow[d, "\partial_k^{-1}"] \\
\int_0^{x_k} f(\xi) d\xi_k \arrow[r, hook] & \partial_k^{-1}(T_f)
\end{tikzcd}$$

Figure 2.9: Commuting diagram for antiderivation of Taylor models.

els. Moreover, since the corresponding DA operation is contracting and smoothness-preserving, it has desirable properties for computational applications. Finally, it should also be noted that the antiderivation of Taylor models is compatible with the corresponding operation on the differential algebra ${}_nD_v$.

2.3.2 Accuracy of Taylor Models

As indicated earlier, one of the main advantages of Taylor models over conventional interval methods lies in the fact the propagating high-order information on the functions allows the accuracy of the enclosures to increase. In fact, the accuracy of the enclosures obtained by Taylor models scales with a high order of the domain size. This makes Taylor models particularly well suited for high-dimensional problems over large domains, since it reduces the number of necessary domain splittings. The following theorem states that in most cases the width of the remainder bounds scales in fact with the $(n+1)$-st order of the domain size.

Theorem 2.6. *Let f be a function that is represented by a finite number of elementary operations and intrinsic functions, and assume that B is an inclusion monotone polynomial bounder that scales at least linear with the magnitude of the domain. If*

$T = (P, x_0, \boldsymbol{D}, R)$ is a Taylor model of order n obtained by evaluating the code list representing f, then the remainder bound R scales with the $(n+1)$-st order of the magnitude of $\boldsymbol{D}$.

Proof. The proof follows by induction over the elementary operations and intrinsic functions that make up the code list for f. Firstly, the assertion is correct for the constant Taylor models and for the identity function, since the respective reference polynomials can be bound with an arbitrary precision.

Since for two Taylor models A and B, $R_{A+B} = R_A + R_B$, the assertion also holds for the sum of Taylor models. As shown earlier, the remainder bound of the product $A \cdot B$ is given by

$$R_{A \cdot B} = R_A \cdot R_B + R_A \cdot B\left(P_B\right) + B\left(P_A\right) R_B + B\left(\left(P_A \cdot P_B\right)_{(n+)}\right). \tag{2.41}$$

Since each of the terms scales with at least order $(n+1)$, so does their sum. Thus, the assertion holds for all elementary operations. (Note that division is defined and implemented in terms of the multiplicative inverse, which is conceptually an intrinsic function.)

Finally, the computation of remainder bounds of intrinsic functions is exemplified by the exponential function. According to [82],

$$R_{\exp(A)} = \left(B\left(P_A - P_A(0)\right) + R_A\right)^{n+1} \cdot \exp\left([0,1] \cdot \left(B\left(\left(P_A - P_A(0)\right)\right) + R_A\right)\right). \tag{2.42}$$

By inclusion monotonicity, the second term never increases with a decreasing domain size and thus, the product scales with the $(n+1)$-st order of the domain size.

Using the complete definitions [83, 82], similar arguments can be made for all the intrinsic functions of Taylor models, including the computation of the multiplicative inverse. Thus, the remainder bound of any finite Taylor model computation does indeed scale with the $(n+1)$-st order of the domain. $\square$

Figure 2.10 illustrates how the fact that the sharpness of Taylor models scales with the $(n+1)$-st order of the domain allows us to obtain sharp bounds quickly, even in the multidimensional case. While the Taylor model of order seven has remainder bounds that are of the same order as the domain, the size of the remainder bounds quickly decreases with increasing order of the Taylor models. Finally, a Taylor model of order ten encloses the function with a remainder bound that is only a fraction of the underlying domain.

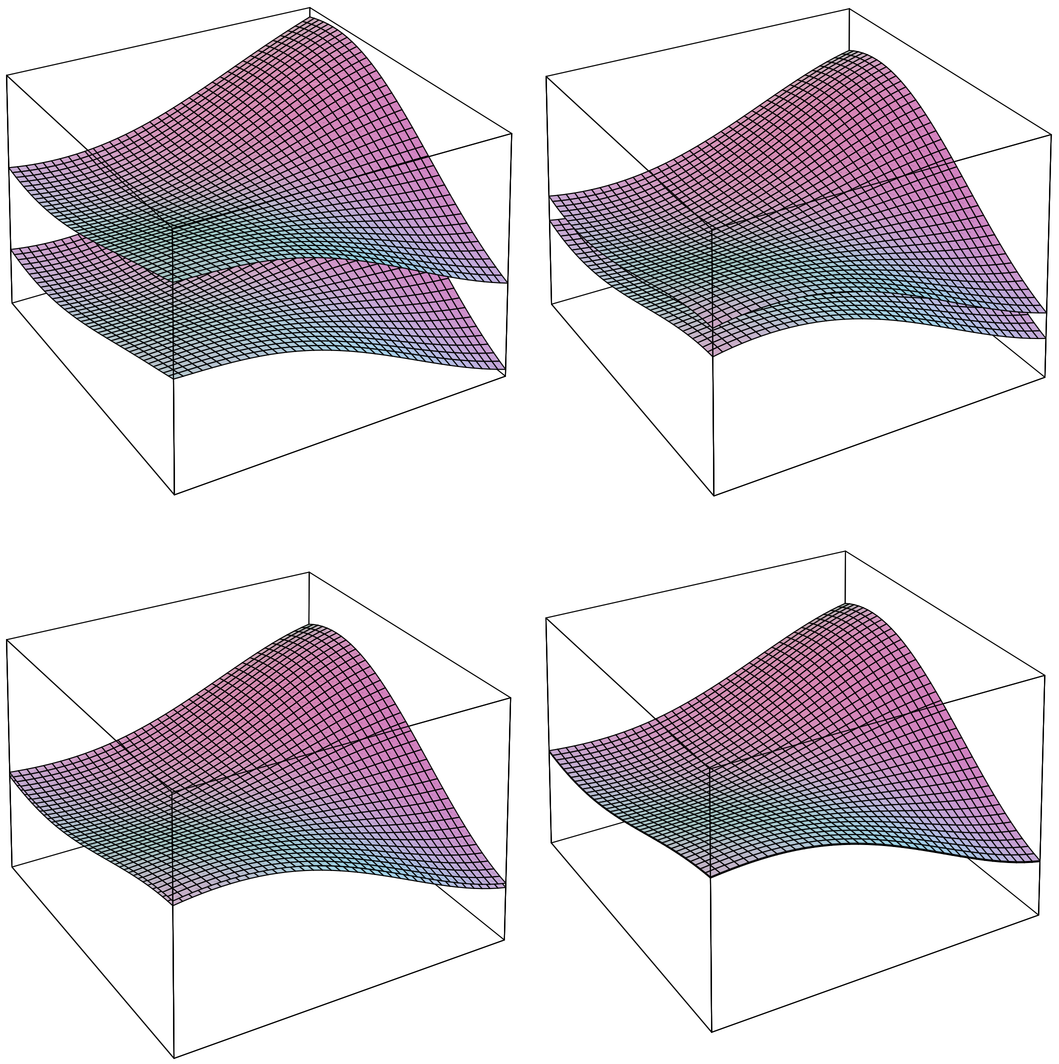


Figure 2.10: Interval bounding of a two-dimensional function by Taylor models of orders seven, eight, nine, and ten.

2.3.3 Applications of Taylor Models

Since their first development in 1996 [83], Taylor model methods have been used for a variety of applications. While recent results will be discussed in later chapters, we summarize some of the fundamental Taylor model algorithms in this section to illustrate the general applicability of the approach.

Bounding Ranges of Functions

One of the most challenging problems in global optimization is the problem of determining bounds on the ranges of functions over given domain boxes. While interval arithmetic can answer that question rigorously [113], the cancellation and dependency problems often lead to range bounds that are too pessimistic to be of practical value.

Taylor models reduce the problem of bounding the range of arbitrary functions to the problem of bounding the range of polynomials. While this is still a hard problem, it is already substantially easier to solve than the more general problem [129]. These Taylor model based techniques have been used in several situations, including verified bounding of highly complex functions [21, 16] and global optimization [88].

Rigorous Quadrature

Quadrature, the numerical computation of integrals, is an important area of numerical analysis. Extensive theories provide formulas for computing approximative values of the integral and rigorous upper bounds for the integration errors. While these methods often converge with high-orders, they usually also require the computation of bounds of higher order derivatives. Although conventional interval methods and automatic differentiation can in principle be used to compute these bounds, the computational overhead for accurate results is often tremendous [92, 131] and the computations suffer from the well known problems of interval arithmetic and automatic differentiation: exponential inflation of code and dependency problems.

Using the antiderivation of Taylor models, the computation of integrals becomes a mere application of the antiderivation. Simply modeling the function by a Taylor model over the region of interest and using the antiderivation yields, according to Def. 2.8, a rigorous enclosure of the primitive. Hence, subsequently computing a rigorous bound of the range of the Taylor model for the primitive gives the desired result. Details on using this approach for high-dimensional verified quadrature are given in [25].

Integration of Ordinary Differential Equations

While conventional interval methods have long been used for the integration of ODEs [92, 66, 35], the wrapping effect and the dependency problem have prevented these methods from successfully integrating complicated systems over large time intervals. On the other hand, one of the most important applications of Taylor model methods lies in the computation of solution of ordinary differential equations (ODEs) under avoidance of the wrapping effect for practical purposes [24]. To illustrate the approach, we consider the initial value problem

$$x' = f(t,x), \quad x(t_0) = x_0. \tag{2.43}$$

It is a well established fact that the solution to Eqn. (2.43) can be obtained as the fixed point of the Picard operator $\mathcal{P}$ given by

$$\mathcal{P}\left(x\right) = x_0 + \int_{t_0}^{t} f\left(\tau, x\right) d\tau. \tag{2.44}$$

Using the antiderivation for Taylor models, the operator $\mathcal{P}$ can be extended to Taylor models and yields a method that allows the computation of verified *enclosures of flows* of Eqn. (2.43). This approach has been discussed in [24, 82] and has recently been used successfully in a variety of applications ranging from solar system dynamics (c.f. Sec. 5.1 and [27]) to beam physics (c.f. Sec 5.2 and [81, 82]). Unlike methods that use regular interval computations to enclose the final conditions, the Taylor model approach avoids the wrapping effect to very high order and is therefore capable of propagating extended initial regions over large integration intervals.

2.3.4 Summary

At the time of writing, Taylor models are a relatively new concept combining high order Taylor polynomials with floating point coefficients with intervals for verification. They allow verified computations while avoiding some of the difficulties inherent in normal interval arithmetic. The Taylor model approach guarantees inclusion of functional dependencies with an accuracy that scales with the $(n+1)$-st order of the domain over which the functions are evaluated. In particular, as shown in [85], this method can often substantially alleviate the following problems inherent in conventional interval arithmetic:

- Sharpness of the Result
- Dependency Problem
- Dimensionality Curse

The remainder of this dissertation focuses on the development of new algorithms and computational methods utilizing the Taylor model approach to obtain guaranteed results.

Chapter 3

Verified High-Order Inversion

In this chapter we derive Taylor model methods to prove the existence of inverse functions and to compute Taylor models for them if they exist. The fundamental problem can be paraphrased as follows.

> Given a function $f : \boldsymbol{D} \subset \mathbb{R}^v \to \mathbb{R}^v$ (known only up to some accuracy) defined over a box $\boldsymbol{D}$, is the function invertible over its range $\boldsymbol{f}(\boldsymbol{D})$? And if so, find a representation of the inverse as accurately as possible.

In real analysis, as opposed to interval analysis, it is usually sufficient to determine invertibility in the neighborhood of a point, and the Inverse Function Theorem establishes a sufficient condition for this:

Theorem 3.1 (Inverse Function Theorem). *Let $U \subset \mathbb{R}^v$ be open and assume that $f : U \to \mathbb{R}^v$ is of class $\mathcal{C}^{n+1}$ with $n \geq 0$. If*

$$D(f)(x_0) : \mathbb{R}^v \to \mathbb{R}^v \tag{3.1}$$

is a linear isomorphism, then there is a neighborhood $V \subset U$ of x_0 such that $f : V \to \mathbb{R}^v$ has a $\mathcal{C}^{n+1}$ inverse $g : f(V) \to V$ such that

$$g \circ f = id_V \text{ and } f \circ g = id_{f(V)}. \tag{3.2}$$

While the Inverse Function Theorem is one of the most powerful theorems in conventional point analysis, its main limitations in practical situations are that it does not give any means of computing the neighborhood V over which the function f has an inverse, nor does it provide a method to actually determine that inverse function. This is usually not a problem in point analysis where the Inverse Function Theorem is used with great success. However, in numerical analysis and interval analysis we are often interested in both computing the inverse function and deciding the question of invertibility over extended regions.

In this chapter we take a fresh look at the problem of computing rigorous enclosures of inverse functions. We first derive a new Taylor model based criterion for invertibility that requires only knowledge about first derivatives. While the method has important applications in studying dynamical systems [133], we will use it as a first step in computing Taylor models for inverse functions. In a second step, we show in Sec 3.2 how Taylor models for inverse functions can be computed from the original Taylor models. By studying several examples, we demonstrate that the method outperforms conventional interval approaches both in terms of accuracy and usable domain size.

3.1 Rigorous Invertibility Analysis

The first step in the process of computing inverse functions is the determination of invertibility over the domain of interest. While the Inverse Function Theorem guarantees invertibility over neighborhoods of points, it gives no estimates on the sizes of these regions of invertibility. In this section we derive Taylor model based methods to rigorously prove invertibility of functions over given domains and demonstrate the applicability of the approach by comparing its performance with several conventional interval methods.

3.1.1 Invertibility from First Derivatives

There are a variety of ways to decide invertibility for a given function f, some of which rely on second and higher order derivatives. These methods are often computationally expensive since they require bounding complicated functions like the operator norm of the second derivative map over the domain of interest [2]. In this section we present a method that requires only bounds on first partial derivatives to decide the question of invertibility. Furthermore, and perhaps more importantly, the method exhibits some important structure regarding the points at which the derivatives actually have to be evaluated. We will later capitalize on this structure to significantly reduce cancellation problems in the necessary verified linear algebra. It should be noted that this method is not local in nature, but can indeed guarantee global invertibility everywhere in the given domain.

The following theorem enables us to decide whether a given function is invertible over a domain by just evaluating its first derivatives. Hence it greatly reduces the computational overhead necessary. It seems to originate in works by K. Kovalevsky from the beginning of the twentieth century but has been "rediscovered" by many others (e.g. [103, E 5.3-4]).

Theorem 3.2 (Invertibility from First Derivatives). *Let $\boldsymbol{D} \subset \mathbb{R}^v$ be a box and $f : \boldsymbol{D} \to \mathbb{R}^v$ a $\mathcal{C}^1$ function. Assume that the matrix*

$$M = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\chi_1) & \cdots & \frac{\partial f_1}{\partial x_v}(\chi_1) \\ \vdots & & \vdots \\ \frac{\partial f_v}{\partial x_1}(\chi_v) & \cdots & \frac{\partial f_v}{\partial x_v}(\chi_v) \end{pmatrix} \tag{3.3}$$

is invertible for every choice of $\chi_1, \ldots, \chi_v \in \boldsymbol{D}$. Then f has a $\mathcal{C}^1$-inverse defined on $\boldsymbol{f}(\boldsymbol{D})$, where $\boldsymbol{f}(\boldsymbol{D})$ denotes the range of f over $\boldsymbol{D}$.

Proof. It suffices to show that f is injective over $\boldsymbol{D}$. So assume that $\exists\, y \in \boldsymbol{f}(\boldsymbol{D})$ and $\exists\, x_1, x_2 \in \boldsymbol{D}$ such that

$$f(x_1) = f(x_2) = y. \tag{3.4}$$

For $t \in [0,1]$ and $i = 1, \ldots, v$, define auxiliary functions

$$h_i(t) = f_i\left(x_1 \cdot (1-t) + x_2 \cdot t\right). \tag{3.5}$$

Since $h_i(0) = h_i(1)$ for all $i = 1, \ldots, v$, the Mean Value Theorem asserts that for each i there exists a $t_i \in [0,1]$ such that

$$\frac{dh_i}{dt}(t_i) = (\nabla f)\left(x_1 \cdot (1-t_i) + x_2 \cdot t_i\right) \cdot (x_2 - x_1) = 0 \tag{3.6}$$

Then for $i = 1, \ldots, v$, define

$$\chi_i = x_1 \cdot (1-t_i) + x_2 \cdot t_i. \tag{3.7}$$

By the convexity of $\boldsymbol{D}$, $\chi_i \in \boldsymbol{D}$ for all $i = 1, \ldots, v$ and moreover,

$$(\nabla f_i) \cdot (x_2 - x_1) = 0 \text{ for all } i = 1, \ldots, v. \tag{3.8}$$

However, the last equation is equivalent to $M \cdot (x_2 - x_1) = 0$. But by assumption M is regular and the kernel of the linear map associated with M is therefore just $\{0\}$. We conclude that $x_1 = x_2$ and therefore that f is one-to-one over the domain $\boldsymbol{D}$. As such it has an inverse, which, by the Inverse Function Theorem, is also of class $\mathcal{C}^1$. $\square$

Regularity of the Jacobian at each point in the domain is a necessary but insufficient condition for invertibility. Thus, while Thm. 3.2 resembles the Inverse Function Theorem, it is in fact stronger in the requirements on f. To see that it is impossible to deduce invertibility from the fact that the Jacobian is non-singular at each point within the domain, consider the function

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x^5 - 10x^3y^2 + 5xy^4 \\ y^5 - 10x^2y^3 + 5x^4y \end{pmatrix}, \tag{3.9}$$

which is related to the complex function $\tilde{f} : \mathbb{C} \to \mathbb{C}$, $\tilde{f}(z) = z^5$ via the identification $z = x + iy$. The derivative of f at any point $(x, y) \in \mathbb{R}^2$ is given by

$$M(x, y) = \begin{pmatrix} 5x^4 - 30x^2y^2 + 5y^4 & 20x^3y - 20xy^3 \\ 20xy^3 - 20x^3y & 5x^4 - 30x^2y^2 + 5y^4 \end{pmatrix}. \tag{3.10}$$

The determinant $\det(M) : \mathbb{R}^2 \to \mathbb{R}$ of M is a polynomial defined on the whole space $\mathbb{R}^2$ and

$$\det(M(x, y)) = \left(5x^4 - 30x^2y^2 + 5y^4\right)^2. \tag{3.11}$$

Hence, over the domain $\boldsymbol{D} = [0.1, 1] \times [0.1, 1]$, the derivative of f is regular at every point. However, for $\varphi_1 = \pi/15$ and $\varphi_2 = 7\pi/15$

$$f\begin{pmatrix} \cos\varphi_1 \\ \sin\varphi_1 \end{pmatrix} = f\begin{pmatrix} \cos\varphi_2 \\ \sin\varphi_2 \end{pmatrix} \tag{3.12}$$

And since $\cos\varphi_1 \approx 0.978$, $\sin\varphi_1 \approx 0.208$, $\cos\varphi_2 \approx 0.105$, and $\sin\varphi_2 \approx 0.995$, f satisfies the requirements of the Inverse Function Theorem at each point in the domain $\boldsymbol{D}$ but fails to be an isomorphism. This shows that a naive application of the Inverse Function Theorem is not sufficient to establish invertibility over extended domains. However, Thm. 3.2 shows that if we view the Jacobian of f as a function of v^2 variables, instead of just v variables, its regularity at each point in the extended domain $\boldsymbol{D}^v$ gives a sufficient criterion to guarantee invertibility of f.

An immediate corollary of Thm. 3.2 is the following interval formulation of sufficient conditions for the existence of the inverse function, which in this version is very well known [100, 119].

Corollary 3.1. *Let f and $\boldsymbol{D}$ be as in Thm. 3.2. For $i, j = 1, \ldots, v$ let $\boldsymbol{p}_{i,j} \subset \mathbb{R}$ be compact intervals such that*

$$\frac{\partial f_i}{\partial x_j}(x) \in \boldsymbol{p}_{i,j} \ \ \forall x \in \boldsymbol{D}. \tag{3.13}$$

If the interval matrix $\boldsymbol{P} = \left(\boldsymbol{p}_{i,j}\right)$ is regular, then f has a $\mathcal{C}^1$-inverse defined on $\boldsymbol{f}(\boldsymbol{D})$.

A practical algorithm based on Thm. 3.2 and Cor. 3.1 requires efficient and accurate methods to model partial derivatives and depends on the availability of efficient methods for the determination of regularity of an interval matrix with potentially large entries. Both these problems can become challenging for naive interval methods which have difficulties modeling complicated functions: they often give significant overestimations in bounding the derivatives of the functions, and suffer from the well known problems that come with an increasing number of variables and an increase in domain sizes.

Moreover, interval methods are not particularly well suited for practical applications of Thm. 3.2 via Cor. 3.1 because of difficulties in handling the fact that the individual rows of the Jacobian can each be evaluated at one point. A new Taylor model based method that circumvents this problem will be presented in the next section. But nevertheless, once the interval matrix $\boldsymbol{P}$ of Cor. 3.1 has been determined as accurately as possible, the next question is how to establish regularity of it. While in lower dimensions it may often be sufficient to compute the interval determinant of the matrix, more sophisticated methods are needed for higher dimensional problems.

It has been shown that regularity of a matrix is equivalent to the componentwise distance to the next singular matrix being greater than one [122]. In the following paragraphs we present two advanced interval based methods to prove the regularity of the interval Jacobian $\boldsymbol{P}$. In Sec. 3.1.3 we will compare these interval methods for the determination of invertibility with a new approach based on Taylor models.

The first method of verifying the regularity of an interval matrix $\boldsymbol{A} = \hat{A} + [-\Delta, \Delta]$ is based on the singular value decompositions of both the midpoint matrix $\hat{A}$ and the Δ-matrix. If $\sigma(M, i)$ denotes the i-th singular value of a matrix M (sorted in non-increasing order), then the following assertion holds [120, 121]:

Theorem 3.3. *Given $\boldsymbol{A} = \hat{A} + [-\Delta, \Delta]$. Then $\sigma(\Delta, 1) < \sigma(\hat{A}, n)$ implies that $\boldsymbol{A}$ is non-singular.*

The success of this method depends mostly on the sharpness of the models of the partial derivatives, and as such this method has difficulties scaling with dimensionality and complexity of the functions of interest.

The next theorem has been presented in [100], and it is one of the most powerful approaches to proving the regularity of a given interval matrix $\boldsymbol{A}$.

Theorem 3.4. *Given $\boldsymbol{A} = \hat{A} + [-\Delta, \Delta]$. If $\hat{A}$ is regular, let S be an approximate inverse of $\hat{A}$. If the spectral radius $\rho(I - S \cdot \boldsymbol{A})$ is less than 1, then any matrix in $\boldsymbol{A}$ is non-singular.*

This method uses a preconditioning of the matrix $\boldsymbol{A}$, which allows the method to work very well for medium sized problems, but as we will see below, in higher dimensions this method still suffers from the fact that the entries of the product matrix $S \cdot \boldsymbol{A}$ are computed from addition and subtraction of intervals and as such are subject to significant overestimations due to cancellation and other dependency problems.

We note that the most practical method to establish the contraction of an interval matrix $\boldsymbol{A}$ is to start out with an arbitrary non-empty interval vector $\boldsymbol{x}_0$ containing the origin and iterate $\boldsymbol{x}_{k+1} = \boldsymbol{B} \cdot \boldsymbol{x}_k$ with $\boldsymbol{B} = I - S \cdot \boldsymbol{A}$. If for any $k \in \mathbb{N}$ we can show that $\boldsymbol{x}_{k+1} \subset \boldsymbol{x}_k$, we have proved that $\boldsymbol{B}$ is indeed contracting and therefore $\rho(\boldsymbol{B}) < 1$ as required by Thm. 3.4.

3.1.2 Verifying Invertibility with Taylor Models

We now come back to the question whether a given function f, defined over some interval box $\boldsymbol{D}$, is invertible over its image. We will combine the results of Thm. 3.2

with the regularity criterion Thm. 3.4 and Taylor model based techniques to derive a new algorithm to rigorously answer this question. Compared to the corresponding interval version of Cor. 3.1, we will be able to avoid an overly pessimistic behavior.

The proof of Thm. 3.2 is based on proving regularity of the matrix

$$M = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\chi_1) & \cdots & \frac{\partial f_1}{\partial x_v}(\chi_1) \\ \vdots & & \vdots \\ \frac{\partial f_v}{\partial x_1}(\chi_v) & \cdots & \frac{\partial f_v}{\partial x_v}(\chi_v) \end{pmatrix} \tag{3.14}$$

with points $\chi_1, \ldots, \chi_v \in \boldsymbol{D}$. Thus, any combination of entries from the same row of M can be evaluated using the same set of domain variables. While interval arithmetic cannot capitalize on this intrinsic structure, as we shall see in the following, Taylor models are particularly well suited for this task. Instead of modeling each partial derivative of the functions f_i by intervals, we now model each of the gradients by one single Taylor model, leaving us with the task of proving regularity of a matrix-valued Taylor model, which we will be doing using a method derived from Thm. 3.4.

A naive application of Thm. 3.2 leads to matrix valued Taylor models depending on the v^2 variables in $\boldsymbol{D}^v$. However, since the size of n-th order Taylor models in v variables is approximately

$$\begin{pmatrix} n+v \\ v \end{pmatrix} = \frac{(n+v)!}{v!\,n!}, \tag{3.15}$$

a dependence on v^2 variables would restrict the method to low orders even for relatively small problems.

In the remainder of this section we present an algorithm that uses Thms. 3.2 and 3.4 to prove invertibility of functions modeled by Taylor models over extended domains and requires only Taylor models depending on v variables. Thus, the method avoids an inflation of dimensionality beyond the v variables that are required anyway to model the functions of interest.

Algorithm

(1) For $i, j = 1, \ldots, v$ let $T_{i,j}$ be Taylor models for the partial derivatives of f; i.e.,

$$\frac{\partial f_i}{\partial x_j} \in T_{i,j}. \tag{3.16}$$

(2) Let $\boldsymbol{T}$ be an interval enclosure of the range of the Taylor models $T_{i,j}$; i.e., $g \in T_{i,j} \Rightarrow g(x) \in \boldsymbol{T}_{i,j}$ for all $x \in \boldsymbol{D}$.

(3) Let $\hat{T}$ be the midpoint matrix of $\boldsymbol{T}$ and similarly for the transpose $\hat{T}^T$ of $\boldsymbol{T}^T$.

(4) Let S be an approximate floating point inverse of $\hat{T}^T$.

(5) Compute a new Taylor model N by

$$N_{i,j} = \delta_{i,j} - \sum_{k=1}^{v} S_{i,k} T^T_{k,j} = \delta_{i,j} - \sum_{k=1}^{v} S_{i,k} T_{j,k}, \tag{3.17}$$

where $\delta_{i,j}$ denotes the Kronecker Delta.

(6) For $i = 1, \ldots, v$ compute Taylor models r_i with the Taylor models e_{v+j} that are defined over new domains, independent of those of the $N_{i,j}$'s:

$$r_i = \sum_{j=1}^{v} N^T_{i,j} \cdot e_{v+j} = \sum_{j=1}^{v} N_{j,i} \cdot e_{v+j}. \tag{3.18}$$

(7) Find interval enclosures $\boldsymbol{I}_i$ for the Taylor models r_i.

(8) If $\boldsymbol{I}_i \subset [-1, 1]$ for all $i = 1, \ldots, v$, the original function f is invertible.

Remarks

In practice, the Taylor models $T_{i,j}$ needed in step (1) can be obtained by evaluating the first order automatic differentiation [111, 112] code belonging to f in Taylor model arithmetic. Additionally, extensions of the Taylor model approach are being developed that allow the determination of the $T_{i,j}$ directly from an extended Taylor model of f. Finally, if f is computed as the flow of a differential equation; i.e., the function that maps initial conditions to final conditions, Taylor models can be obtained as solutions of an augmented differential equation (c.f. Sec. 5.2).

As outlined earlier, all Taylor models with the same index i; i.e., belonging to the same row of the matrix, can be evaluated using the same set of domain variables. The next step now is to compute a suitable preconditioning matrix for the transpose of T. However, instead of showing that the matrix T is regular, we will show the equivalent statement that the transpose of T is regular. By working with the transpose T^T in steps (3–5), the resulting entries of the matrix N are all computed using linear combinations of Taylor models that can be evaluated over the same set of domain variables. Hence each entry of N is again a Taylor model in v variables, and the columns of N are now modeled over the same set of domain variables. As a consequence, the sum $\sum_{k=1}^{v} S_{i,k}T_{j,k}$ can be evaluated in Taylor model arithmetic, and in particular any manifestations of the dependency problem are suppressed, as in other Taylor model computations.

The final stage of the method in steps (6–8) is to show that the spectral radius of the resulting matrix-valued Taylor model N is less than unity. To establish this, the image of the unit ball (in the maximum norm) under the map N is considered. If we succeed in showing that the image is properly contained in the unit ball itself, invertibility is verified.

Similar to the above, it proves advantageous not to work with the matrix N, but rather with its transpose N^T. In this way, mixing of Taylor models from different columns of N is again avoided, which allows suppression of blow up in the necessary linear algebra. This is justified since the transposed matrix has the same spectral radius as the original one. As a final application of Taylor models, we model the unit ball by Taylor models as follows. For any small $\varepsilon > 0$, the identity function over the interval $[-1, 1]$ is contained in $(x, 0, [-1, 1], (-\varepsilon, \varepsilon))$. Thus, if we denote the i-th such Taylor model by e_i, a bound on the range of e_i contains the closed unit interval in x_i. It should be noted that due to the special nature of the Taylor models r_i, the bounding is actually quite simple: each of the v terms of the sum that constitutes the Taylor models r_i has a different set of variables. Thus, there is no cancellation in the sum, and hence the sharpest possible bound of the sum equals the sum of the bounds of the individual terms. Moreover, the latter are just Taylor models multiplied by the interval $[-1, 1]$, and hence their ranges are just the ranges of the Taylor model multiplied by that interval.

Summary

The presented method utilizes Taylor models at two crucial points. Firstly Taylor models are used to model the partial derivatives of the function f. As will be seen later, this helps tremendously in fighting the issues of complexity and cancellation in modeling the derivatives. Moreover, the Taylor models are used to compute enclosures for the derivative matrices under consideration, and they are used for all arithmetic operations on these matrices which minimizes the overestimation due to cancellation, since the majority of the functional dependence is propagated in the reference polynomial and therefore not subject to the dependency problem [85].

Finally, we have been able to modify the computation in such a way that all arithmetic operations on Taylor models can be performed using only v variables, which allows for a favorable computational complexity and allows the method to scale well to high dimensional problems.

3.1.3 Comparison of Invertibility Criteria

As outlined earlier, we are mainly interested in proving invertibility for a given function over domains as large as possible. But not only the sheer sizes of the domains of invertibility matter, but an equally important question is how good we are able to fill out the maximum region of invertibility from within; i.e., how close to a critical point can we still prove invertibility. Moreover, in many applications we are likely to be confronted with high dimensional problems. The following examples will demonstrate how the new Taylor model based method performs in each of these areas as compared to standard interval methods.

In each of the following examples we have tested a number of functions for invertibility over various domains. All tests have been performed with each of the methods listed below, and the results are presented in graphs and discussed. Note that the determinant test has only been used for problems of up to six variables.

(1) Interval tests based on Cor. 3.1 and an interval determinant

(2) Interval tests based on Cor. 3.1 and Thm. 3.3

(3) Interval tests based on Cor. 3.1 and Thm. 3.4

(4) Taylor model based tests as presented in Sec. 3.1.2

For the interpretation of the results discussed in this section it is important to note that in all cases the necessary bounds on range enclosures have been computed using regular interval arithmetic: no dedicated range bounders utilizing domain decomposition have been used either in the interval or Taylor model approaches to keep the computational overhead within reasonable limits.

Invertibility as a Function of Dimensionality

The first example illustrates how the performance of the presented methods behaves as a function of dimensionality. Invertibility of one thousand 15-th order polynomials with uniformly distributed random coefficients in $[-1, 1]$ has been tested with the methods (1) to (4). The domain of the random polynomials was $[-0.005, 0.005]^v$ for $v = 1, \ldots, 10$. Fig. 3.1 shows the percentage of these random polynomials that could be verified to be invertible as a function of dimensionality. To simulate realistic conditions, the Taylor models for these order 15 polynomials have been of order ten and the resulting remainder bounds are in the order of 10^{-12}.

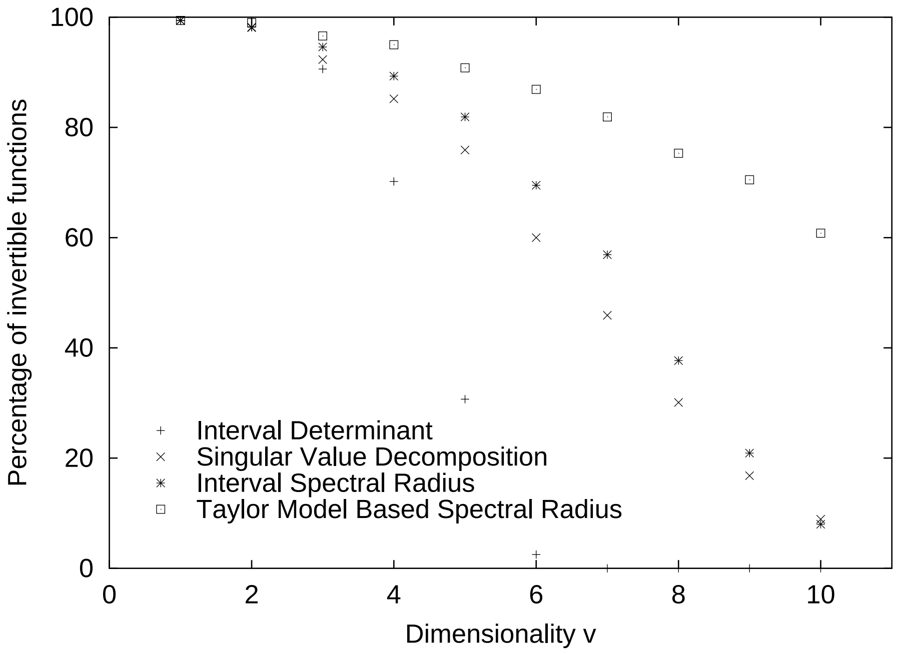


Figure 3.1: Percentage of random functions that can be shown to be invertible as a function of dimensionality.

As expected, for increasing dimensionality the number of successfully established invertible polynomials is decreasing. But it is important to note that the Taylor model based method performs much better in establishing invertibility and suffers much less from an increase in the number of variables than the interval based methods do.

Unfortunately there is no good way to assess what fraction of the original functions are truly invertible; but it is to be expected that this fraction decreases with the dimensionality, accounting for the drop of predicted invertibility in all approaches.

Invertibility as a Function of Domain Size

For the next example we have investigated how the presented methods of establishing invertibility behave as functions of the domain size. We have restricted ourselves to the medium sized problem of 10-th order polynomials in six variables. For each polynomial we have applied the methods (1) to (4) over domains of increasing size. All domain boxes have been placed symmetrically around the origin and the abscissa in Fig. 3.2 shows the magnitude of the domains. For each of the different domain sizes we have plotted the percentage of polynomials for which invertibility could be proven using the four different schemes. As in the previous example, the computation was performed with 1000 random polynomials and the Taylor models were extended with artificial remainder bounds in the order of 10^{-12}. The results of this experiment are illustrated in Fig. 3.2.

In Thm. 2.6 we have shown that the accuracy of Taylor models scales with the $(n+1)$-st order of the domain size, while interval arithmetic scales at most quadratically with the domain sizes. That explains why the Taylor model based method can assert invertibility over domains that are much larger than the ones interval based methods can handle. Moreover, it should also be noted that the number of invertible polynomials naturally decreases with an increasing domain size, since the probability of singular points being contained in the domain $\boldsymbol{D}$ increases with the volume of $\boldsymbol{D}$.

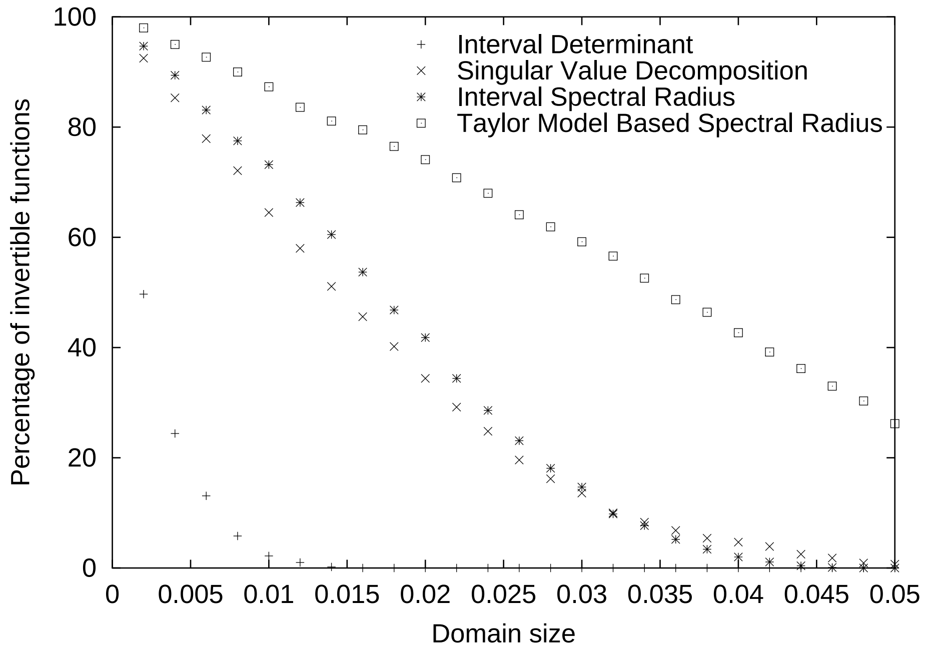


Figure 3.2: Percentage of random functions that can be shown to be invertible as a function of domain size.

Invertibility as a Function of Non-Linearity

This example demonstrates how the performance of the discussed methods changes with the non-linearity of the functions of interest. To that end we have considered functions $f = (f_1, \ldots, f_6) : [-1,1]^6 \subset \mathbb{R}^6 \to \mathbb{R}^6$ given by

$$f_i(x_1, \ldots, x_6) = \frac{\sum_{k=1}^{6} a_{i,k} x_k}{\left(1 - \sum_{k=1}^{6} b_{i,k} x_k\right)^2 + \varepsilon^2} \tag{3.19}$$

with $\varepsilon \in \mathbb{R}$ and 6×6 matrices $A = (a_{i,k})$ and $B = (b_{i,k})$ with coefficients in $[-1,1]$. For small values of ε and appropriate coefficients $a_{i,k}$ and $b_{i,k}$, this may give ill-defined functions because of vanishing denominators. However, in these cases we count the functions as not invertible.

We have generated 500 functions by choosing the coefficients of the matrices A and B randomly in the interval $[-1,1]$, and the presented methods have been used to prove invertibility over $[-1,1]^6$. Fig. 3.3 shows the percentage of functions that can be shown to be invertible depending on the non-linearity ε. All Taylor model computations have been performed in order 6 and the value of ε has been increased linearly from 0 to 250.

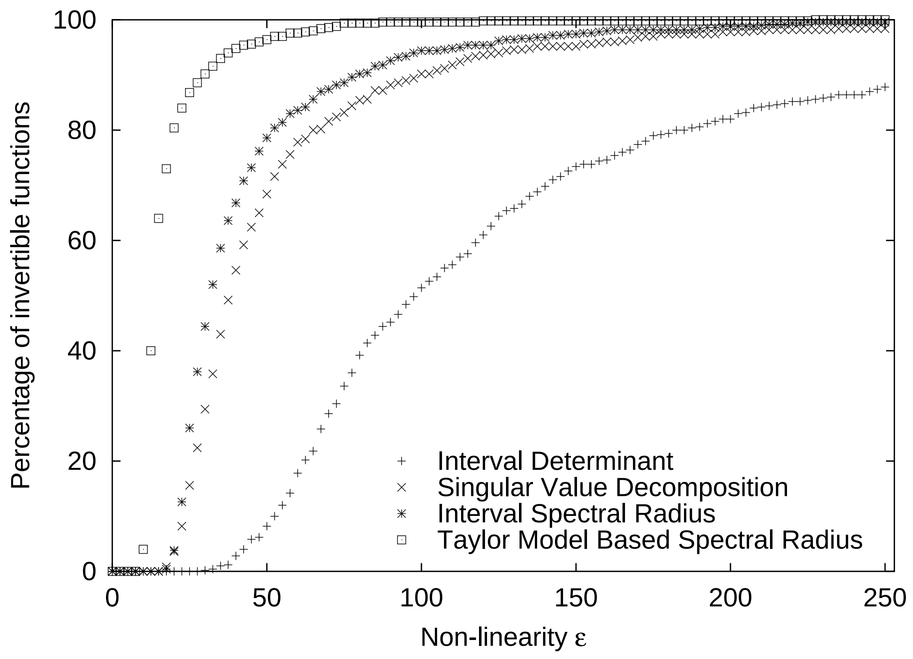


Figure 3.3: Percentage of functions that can be shown to be invertible as a function of the non-linearity ε.

The non-linearity of these functions is mostly determined by the quantity ε, such that for $\varepsilon \gg 1$, the functions are almost linear and their invertibility depends mostly on the invertibility of A. Since almost all computer generated random matrices are invertible, it is to be expected that all methods succeed in proving invertibility for sufficiently large ε. Fig. 3.3 indicates that this is indeed the case.

For $\varepsilon \sim 1$ on the other hand, the resulting functions show a large non-linearity and all methods fail in proving invertibility. It is likely that due to the size of the domain box and the high degree of non-linearity, these functions are truly not invertible.

Between these two extremes, with increasing ε, the different methods become more successful in establishing invertibility. However, the Taylor model based method starts succeeding in proving invertibility very suddenly for $\varepsilon \simeq 20$, while the success of the other methods sets in only for larger ε, and increases at a much slower rate.

Invertibility in the Vicinity of a Critical Point

As another example of how the presented methods scale to larger domain sizes and to illustrate how they behave in the neighborhood of a singular point, consider the function $f = (f_1, \ldots, f_6) : \mathbb{R}^6 \to \mathbb{R}^6$ with components f_i given by

$$\begin{aligned} f_i(x_1, \ldots, x_6) &= (x_i - x_0)^2 + \lambda \cos((x_i - x_0) \cdot (x_{\pi(i)} - x_0)) \\ &\quad -\lambda(x_{\pi(i)} - x_0) \sin(x_i - x_0) \end{aligned} \tag{3.20}$$

where $\pi(i) = i + 1$ for $i = 1, \ldots, 5$ and $\pi(6) = 1$.

At the point $(x_0, \ldots, x_0)$ the Jacobian of f vanishes and hence for symmetric domain boxes centered at the origin, $2x_0$ is an upper bound for the magnitude of the domain of invertibility. Fig. 3.4 shows the percentage of this maximal domain diameter $2x_0$ (for $x_0 = 0.1$) over which the different methods can prove invertibility as a function of the parameter λ, which is gradually increased from 0 to 2. The invertibility tests have been performed for the domains $[-p \cdot 0.001, p \cdot 0.001]^6$ with $p = 1, \ldots, 100$, and the figure shows the maximal value of p for which the various methods can determine invertibility as a function of λ. The results indicate that the Taylor model based test can guarantee invertibility over a much larger region and is less sensitive to the perturbation λ.

It is important to note that all interval based methods perform equally poorly for increasing λ. This indicates that the real problem of establishing invertibility in this example is with the use of conventional intervals for the modeling of the derivatives and not so much with the different methods to establish regularity of the interval matrix of derivatives. This example illustrates how Taylor models can enclose even complicated functions extremely accurate: the remainder bounds of the Taylor models

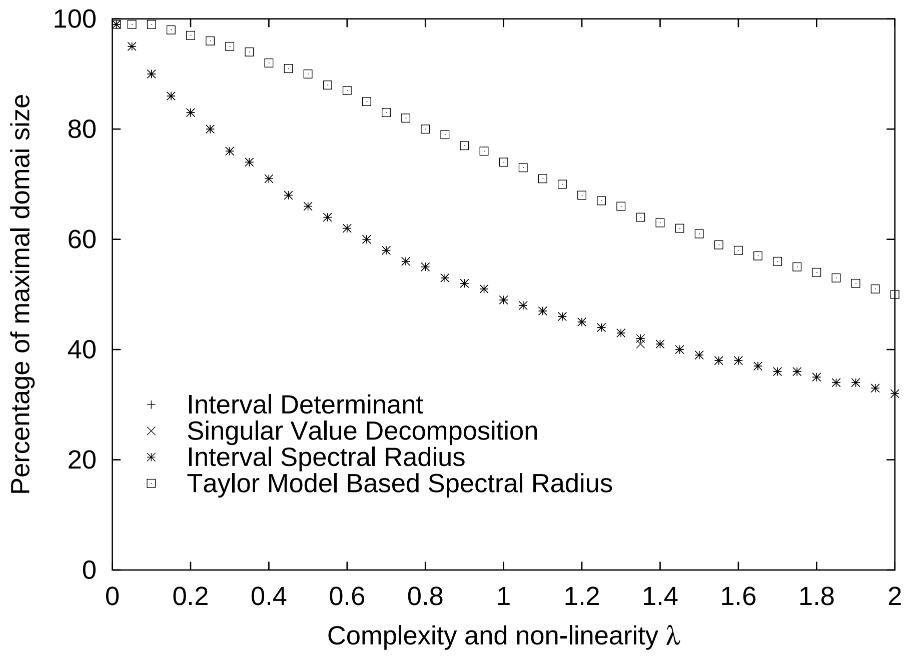


Figure 3.4: Percentage of maximal domain size over which invertibility can be proven as a function of complexity and non-linearity in the partial derivatives.

for the partial derivatives are all in the order of 10^{-12}.

Invertibility as a Function of Functional Complexity

In this last example we study how the presented methods behave as a function of computational complexity of the original function f. To that end, we have emulated functional complexity using the following method.

For the previously introduced random polynomials of order ten in six variables with coefficients in $[-1, 1]$ we model computational complexity c by

$$f(x) = \frac{1}{\sqrt{c}} \sum_{i=1}^{c} f_i(x), \tag{3.21}$$

where each of the f_i is a random polynomial as before and the scaling factor has been introduced to maintain the standard deviation of the coefficients of the resulting polynomials. All tests were performed over the $[-0.005, 0.005]^6$ domain box and the results are shown in Fig. 3.5.

It is important to note that the functions have been evaluated by adding the results of the individual polynomial evaluations. This is a realistic model of practical applications where the supplied functions often come as *black boxes* that do not permit any further simplifications to control cancellations.

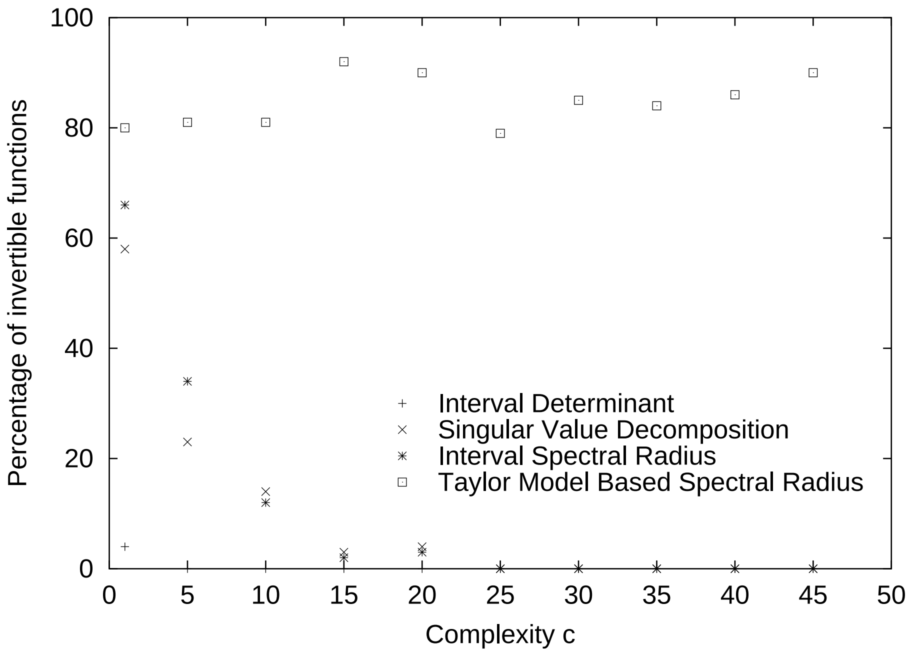


Figure 3.5: Percentage of random functions that can be shown to be invertible as a function of functional complexity.

The first observation worth noting here is that the Taylor model based method does not suffer from an increase in complexity, but rather even seems to improve with it. This is caused by the way we simulate computational complexity: while the coefficients in the original polynomials are uniformly distributed in $[-1, 1]$, the coefficients of the resulting “complex” one are not uniform anymore, and this distributional change leads to a simpler behavior with improved invertibility.

As outlined in Sec. 2.3, Taylor models are particularly well suited to deal with the cancellation problems that plague conventional interval arithmetic, since the bigger part of that cancellation takes place in the polynomial coefficient real number arithmetic and only a small fraction of it contributes directly to the remainder bounds. (This is in contrast to normal interval arithmetic where all the functional dependence is propagated in the remainder bound.) As such, Taylor models are expectedly much better in modeling the derivatives themselves and hence the Taylor model based methods can succeed in proving invertibility for computationally complex functions that cannot be properly modeled by intervals.

3.2 Guaranteed Enclosures of Inverse Functions

Once the existence of an inverse function has been established for a function f contained in a Taylor model, the question of actually computing a Taylor model for the inverse function f^{-1} arises naturally. In the following we develop methods for computing inclusions of inverses of general functions in Taylor models. We show how Taylor model techniques can be combined with the methods for the verification of invertibility presented earlier to derive a new method that allows the computation of Taylor models containing the inverses of given invertible functions enclosed in Taylor models.

3.2.1 Polynomial Inverses

If we start out with a Taylor model T for an invertible function f, the first step in computing a Taylor model S for f^{-1} is the computation of the reference polynomial of S. According to Def. 2.6, this requires the computation of the n-th order Taylor polynomial of f^{-1}. In this section we utilize the DA Fixed Point Theorem to calculate this n-th order Taylor polynomial of the inverse.

Assume that $f \in \mathcal{C}^{n+1}(\mathbb{R}^v, \mathbb{R}^v)$ is an origin-preserving map; i.e., $f(0) = 0$. If the derivative M of f at 0 is a linear isomorphism, then the Inverse Function Theorem guarantees that there is a neighborhood V of 0, such that f^{-1} exists and is also of class $\mathcal{C}^{n+1}$ on $f(V)$. Once the linearization of f has been shown to be invertible, within the DA framework it is possible to compute a representative of the equivalence class of $[f^{-1}]_n$ from a single representative $\mathcal{M}$ of the equivalence class $[f]_n$. Split the map $\mathcal{M}$ as

$$\mathcal{M} = M + \mathcal{N}_{\mathcal{M}} \tag{3.22}$$

where $\mathcal{N}_{\mathcal{M}}$ denotes the purely non-linear part of $\mathcal{M}$. Composing from the right with the locally existing inverse results in

$$\mathcal{I} \;=\; \mathcal{M} \circ \mathcal{M}^{-1} = M \circ \mathcal{M}^{-1} + \mathcal{N}_{\mathcal{M}} \circ \mathcal{M}^{-1} \tag{3.23}$$

$$\Rightarrow\; \mathcal{M}^{-1} = M^{-1} \circ \left(\mathcal{I} - \mathcal{N}_{\mathcal{M}} \circ \mathcal{M}^{-1}\right) = \mathcal{O}\left(\mathcal{M}^{-1}\right) \tag{3.24}$$

where $\mathcal{I}$ is the identity map. The last equation is actually a fixed point relation for the map $\mathcal{M}^{-1}$. Viewing this equation as a relation on equivalence classes, it turns out that $\mathcal{O}$ is contracting because

$$\left(\mathcal{O}\left([\mathcal{A}]_n\right) - \mathcal{O}\left([\mathcal{B}]_n\right)\right) = M^{-1} \circ \left([\mathcal{N}_{\mathcal{M}}]_n \circ [\mathcal{B}]_n - [\mathcal{N}_{\mathcal{M}}]_n \circ [\mathcal{A}]_n\right). \tag{3.25}$$

Thus, with $[\mathcal{A}]_n$ and $[\mathcal{B}]_n$ agreeing up to order k, the nilpotency of $\mathcal{N}$ implies that $[\mathcal{N}_{\mathcal{M}}]_n \circ [\mathcal{B}]_n$ and $[\mathcal{N}_{\mathcal{M}}]_n \circ [\mathcal{A}]_n$ agree up to order $k+1$.

According to the DA Fixed Point Theorem, this assures the existence of a unique fixed point of the operator $\mathcal{O}$, which can be reached in at most $n+1$ iterations. Moreover, this method generates all derivatives of up to order n of f^{-1} at the origin in finitely many steps through mere iteration. Because it only requires the iteration of a rather simple operator, this approach is particularly useful for computational applications. More details on this method can be found in [17].

While the presented method by itself allows only the computation of inverse polynomials for origin-preserving maps, we will show at the end of the next section how it can nevertheless be used for the computation of Taylor models for general inverse functions.

3.2.2 Inverse Taylor Models

The computation of verified inverses starts by modeling the functional dependence of interest by a Taylor model and the a priori determination of invertibility. With the knowledge that the function contained in the Taylor model is actually invertible, the computation of an enclosure of the inverse follows the steps outlined in this section.

First, it should be clarified what exactly constitutes an inverse Taylor model, since compatibility requirements of the underlying domains force us to consider two kinds of inverse Taylor models as given by the next definition. We denote by $B(T)$ an inclusion of the range of all functions in the Taylor model T and by I a Taylor model $(x, 0, \boldsymbol{D}, (-\varepsilon, \varepsilon))$ of the identity function over the domain $\boldsymbol{D}$ (with some arbitrary small $\varepsilon > 0$). Finally, for a polynomial P and a Taylor model T, $P(T)$ denotes the result of the formal evaluation of the polynomial P with the argument T. This is well defined since it requires only additions and multiplications of Taylor models with Taylor models and real numbers.

Definition 3.1 (Inverse Taylor Model). *Let $T = (P_n, x_0, \boldsymbol{D}, R)$ and $S = (G_n, y_0, \boldsymbol{\Delta}, \Omega)$ be two Taylor models. S is called a* left-inverse Taylor model for T *if*

1. $G_n \circ P_n =_n \mathcal{I}$,
2. $P_n(x_0) = y_0$,
3. $f \in T \Rightarrow \boldsymbol{f}(\boldsymbol{D}) \subset \boldsymbol{\Delta}$,
4. $B(G_n(T) - I) \subsetneq \Omega$.

Similarly, S is called a right-inverse Taylor model for T *if T is a left-inverse Taylor model for S.*

It turns out that the rather general definition of left-inverse Taylor models ensures the verified enclosure of left-inverses for all invertible functions contained in T. Moreover, the computation of left-inverse Taylor models is usually sufficient for practical applications, their computation does not require a modification of the original domains, and their domains are easily computed.

Theorem 3.5. *Let $T = (P_n, x_0, \boldsymbol{D}, R)$ and $S = (G_n, y_0, \boldsymbol{\Delta}, \Omega)$ be given Taylor models such that S is a left-inverse Taylor model for T. Assume that $f \in T$ is invertible over $\boldsymbol{D}$. Then there is $g \in S$ such that $g = f^{-1}$ on $\boldsymbol{f}(\boldsymbol{D})$.*

Proof. Define $\delta_f : \boldsymbol{D} \to \mathbb{R}^v$ by $\delta_f(x) := G_n(f(x)) - x$. Note that the Taylor expansion of δ_f around $x_0 \in \boldsymbol{D}$ vanishes up to order n since the expansion of f around x_0 agrees with P_n up to order n and by assumption $G_n(P_n(x)) =_n x$.

Since f is invertible over $\boldsymbol{D}$, $\forall\, y \in \boldsymbol{f}(\boldsymbol{D})$, there is a unique $x_y \in \boldsymbol{D}$ such that $f(x_y) = y$. Then, for $y \in \boldsymbol{f}^{\boldsymbol{u}}(\boldsymbol{D})$ define the function

$$\tilde{g}(y) = G_n(y) - \delta_f(x_y) = G_n(y) - \delta_f(f^{-1}(y)). \tag{3.26}$$

Then, for any $x \in \boldsymbol{D}$ we have

$$\tilde{g}\left(f\left(x\right)\right) = G_n\left(f\left(x\right)\right) - \delta_f\left(f^{-1}\left(f\left(x\right)\right)\right) \tag{3.27a}$$

$$= x + \delta_f\left(x\right) - \delta_f\left(f^{-1}\left(f\left(x\right)\right)\right) = x. \tag{3.27b}$$

Thus $\tilde{g}$ is the unique inverse to f on $\boldsymbol{f}(\boldsymbol{D})$. Since $|\tilde{g}(y) - G_n(y)| = |\delta_f(f^{-1}(y))| \in \Omega$, it follows that $\tilde{g}$ is contained in the Taylor model $S_f := (G_n, y_0, \boldsymbol{f}(\boldsymbol{D}), \Omega)$.

As the continuous image of a compact set, $\boldsymbol{f}(\boldsymbol{D})$ is itself compact and hence the function $\tilde{g}$ can be extended $(n+1)$-times continuously differentiable to a function g

defined on the whole set $\boldsymbol{\Delta}$. Moreover, since Ω is a proper superset of $\tilde{\boldsymbol{g}}\left(\boldsymbol{f}\left(\boldsymbol{D}\right)\right)$, it is possible to satisfy $\left(\boldsymbol{g}-\boldsymbol{G_n}\right)\left(\boldsymbol{\Delta}\right) \subset \Omega$. Since the n-th order Taylor expansion of g around $y_0 = f(x_0) \in \boldsymbol{D}$ equals the n-th order expansion of $\tilde{g}$, which in turn equals G_n, it follows that $g \in S$. Finally, for $x \in \boldsymbol{D}$, $g(f(x)) = \tilde{g}(f(x)) = x$. $\square$

The first step in translating the previous theorem into a practicable algorithm lies in the efficient computation of the inverse polynomial G_n. To this end, let $T = (P_n, x_0, \boldsymbol{D}, R)$ be a given Taylor model and assume that P_n is locally invertible around x_0. Then consider the polynomial

$$\tilde{P}(x) = P(x + x_0) - P(x_0). \tag{3.28}$$

Apparently $\tilde{P}(0) = 0$, and hence, according to Sec. 3.2.1, $[\tilde{P}^{-1}]$ can be calculated in at most $n+1$ steps. With $y_0 = P(x_0)$ we obtain $G_n(y)$ as

$$G_n(y) = x_0 + \tilde{P}^{-1}\left(y - y_0\right). \tag{3.29}$$

Once the reference polynomial of the inverse Taylor model has been computed, the next step is the determination of an appropriate domain $\boldsymbol{\Delta}$ such that either $f \in T \Rightarrow \boldsymbol{f}\left(\boldsymbol{D}\right) \subset \boldsymbol{\Delta}$ (left-inverse Taylor model) or $g \in S \Rightarrow \boldsymbol{g}\left(\boldsymbol{\Delta}\right) \subset \boldsymbol{D}$ (right-inverse Taylor model). Fig. 3.6 illustrates the various domains involved in the computation of inverse Taylor models and their relationships

$$X = \bigcup_{f \in T} \boldsymbol{f}\left(\boldsymbol{D}\right) \text{ and } Y = \bigcap_{f \in T} \boldsymbol{f}\left(\boldsymbol{D}\right), \tag{3.30}$$

and hence X and Y are the smallest and largest sets that could be used as domains for left- and right-inverse Taylor models, respectively. Since a general computation and representation of X and Y is impossible, a practical application strives to find a small overestimation $\boldsymbol{W}$ for X and a large underestimation $\boldsymbol{Z}$ for Y such that

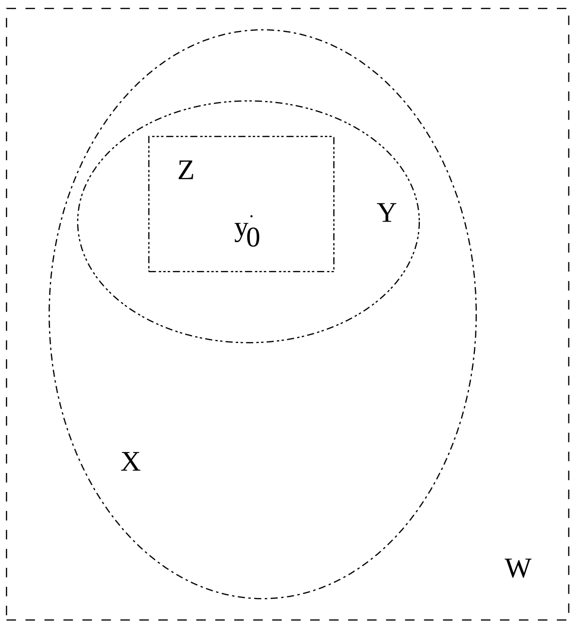


Figure 3.6: Illustration of the domains in the definitions of inverse Taylor models.

$\boldsymbol{Z} \subset Y \subset X \subset \boldsymbol{W}$. Our implementation of this method chooses $\boldsymbol{\Delta} = \boldsymbol{T}(\boldsymbol{D})$ as the domain for the left-inverse Taylor model. Computation of the latter consists of bounding the range of the reference polynomial P_n over $\boldsymbol{D}$ and adding the remainder bound R. While the bound on the polynomial does not necessarily have to be accurate to obtain a left-inverse Taylor model, the practical application of left-inverse Taylor models benefits from overestimations as small as possible.

3.3 Examples of Taylor Model Inversion

In the following we demonstrate the performance of the inversion methods presented in the previous two sections in examples. All computations have been performed using the Taylor model objects in the arbitrary order code COSY Infinity [23, 26, 82].

3.3.1 One-Dimensional Function

As a first example, consider the one-dimensional sine function over the domain $\boldsymbol{D} = [-0.5, 0.5]$. The functional dependence is modeled by a 19-th order Taylor model over that domain, and the result is presented in Table 3.1. The 19-th order Taylor model encloses the sine function with an accuracy that is close to the machine epsilon of approximately 10^{-16}.

```
RDA VARIABLE:   NO=  19, NV=     1
   I  COEFFICIENT            ORDER EXPONENTS
   1   1.000000000000000       1       1
   2  -.1666666666666667       3       3
   3  0.8333333333333333E-02   5       5
   4  -.1984126984126984E-03   7       7
   5  0.2755731922398589E-05   9       9
   6  -.2505210838544172E-07  11      11
   7  0.1605904383682162E-09  13      13
   8  -.7647163731819817E-12  15      15
   9  0.2811457254345521E-14  17      17
   -------------------------------------
VAR    REFERENCE POINT               DOMAIN INTERVAL
  1  0.000000000000000   [-.5000000000000000  ,0.5000000000000000]
              REMAINDER BOUND INTERVAL
  R     [-.1085432243394823E-014,0.1085432243394823E-014]
*********************************************************
```

Table 3.1: Taylor model of one-dimensional sine function computed with COSY Infinity. Shown are Taylor coefficients, reference point, domain information, and remainder bound.

Invertibility of the sine function is easily verified using the method presented in Sec. 3.1.2. The next step is the computation of a polynomial approximation of the inverse of the reference function. Not surprisingly, the methods of Sec. 3.2.2 obtain the 19-th order Taylor expansion of the arcsine function up to machine precision. Proper computation of the remainder bounds and the domain gives the left-inverse Taylor model shown in Tab. 3.2.

Fig. 3.7 shows the difference between the arcsine function and the reference poly-

```
RDA VARIABLE:   NO=  19, NV=     1
   I  COEFFICIENT            ORDER EXPONENTS
   1   1.000000000000000       1       1
   2  0.1666666666666667       3       3
   3  0.7500000000000000E-01   5       5
   4  0.4464285714285714E-01   7       7
   5  0.3038194444444444E-01   9       9
   6  0.2237215909090909E-01  11      11
   7  0.1735276442307693E-01  13      13
   8  0.1396484375000000E-01  15      15
   9  0.1155180089613970E-01  17      17
  10  0.9761609529194068E-02  19      19
  ---------------------------------------
VAR    REFERENCE POINT              DOMAIN INTERVAL
  1  0.000000000000000   [-.5210953054937487  ,0.5210953054937487]
               REMAINDER BOUND INTERVAL
  R     [-.7707363654262549E-008,0.7707363654262549E-008]
*********************************************************
```

Table 3.2: Left-inverse Taylor model for the Taylor model shown in Table 3.1. Shown are Taylor coefficients, reference point, domain information, and remainder bound.

nomial over the interval $\mathbf{sin}\,(\boldsymbol{D}) = [-0.479425\ldots, 0.479425\ldots]$. As expected, the error stays within the remainder bounds (illustrated by the horizontal lines), and the remainder bounds represent an overestimation of the true error by less than a factor of four. It should be noted, however, that the arcsine function as such is not contained in the resulting left-inverse Taylor model. But, as outlined earlier, the part of the arcsine function necessary to act as a left-inverse of the sine function over the domain $\mathbf{sin}\,(\boldsymbol{D})$ is contained and can be extended such that the resulting function is defined over the whole domain of the left-inverse Taylor model.

3.3.2 Six-Dimensional Function

The following example involves a six-dimensional exponential function that has a dense 8-th order Taylor polynomial with 3003 non-vanishing coefficients in each of the six reference polynomials.

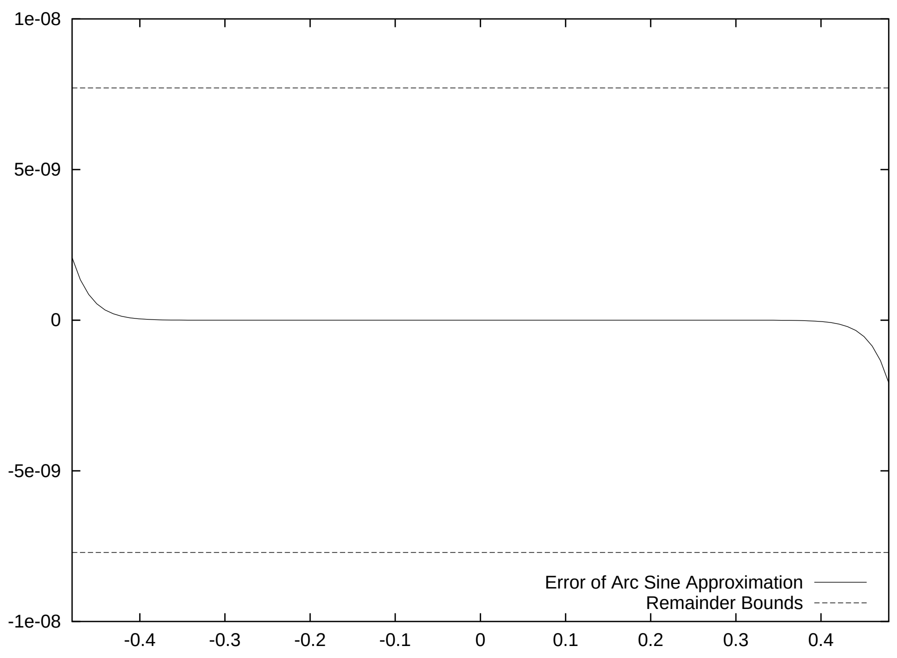


Figure 3.7: Difference between the arcsine function and the reference polynomial of the 19-th order Taylor model shown in Table 3.2.

Let $A = (a_{ij})$ be an invertible 6×6 matrix and consider $f = (f_1, \dots, f_6) : \mathbb{R}^6 \to \mathbb{R}^6$ defined by

$$f_i(x_1, \dots, x_6) = \exp\left(\sum_{j=1}^{6} a_{ij} x_j\right) - 1. \tag{3.31}$$

If $B = (b_{ij})$ is the inverse matrix of A, the inverse function $g = (g_1, \dots, g_6)$ of f is given by

$$g_i(y_1, \dots, y_6) = \sum_{j=1}^{6} b_{ij} \log\left(y_j + 1\right). \tag{3.32}$$

Using the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & -1 \end{pmatrix}, \tag{3.33}$$

the 8-th order Taylor model method for verified inversion has been applied to the function f over the domain box $\boldsymbol{D} = [-0.01, 0.01]^6$.

The resulting left-inverse Taylor model is defined over the domain $[-0.061686, 0.061686]^6$, and it has been verified that it contains the true inverse function g. The remainder bounds of the individual components of the left-inverse Taylor model are listed in Tab. 3.3. Since the range of the exponential function and its inverse are in the order of 10^{-1}, the relative accuracy of the remainder bounds is about 10^{-10}. In Sec. 3.6.1 we will present a comparison of these results with conventional interval techniques.

Component	*Remainder Bound*
1	[-0.4190638646976846E-011, 0.4184087823912867E-011]
2	[-0.2791908825275360E-011, 0.2791908821988238E-011]
3	[-0.2791908824574486E-011, 0.2791908821987869E-011]
4	[-0.1396454411975411E-011, 0.1396454410994258E-011]
5	[-0.1396454411909750E-011, 0.1396454410994186E-011]
6	[-0.1396454411225902E-011, 0.1396454410994267E-011]

Table 3.3: Remainder Bounds of the left-inverse Taylor model for six-dimensional exponential function given by Eqns. (3.31) and (3.33).

3.4 A Superconvergent Newton Method

While the conventional interval Newton method presented in Sec. 2.2.3 is a very powerful and versatile tool for the rigorous enclosing of zeros, its relatively slow quadratic convergence often results in the need for domain subdivisions.

We will now apply the methods derived in the previous section to a higher order version of the interval Newton method. For practical root finding problems the new method offers a larger domain of convergence, and is more robust for complicated functions because of the decrease of the dependency and cancellation problems. Since the method converges with an order higher than two, it is also called *superconvergent*.

While the conventional interval Newton methods are essentially based on first order Taylor approximations and inversions, the availability of high order Taylor models and the corresponding inversion tools enables us to overcome the limitation of slow convergence of the otherwise powerful interval Newton method. Based on the idea of approximating the function f by a high order Taylor model and computing a left inverse Taylor model for it, the Taylor model based computation of zeros is elegant and straightforward, and can easily be implemented using the COSY Infinity [23, 84] language environment. The algorithm takes the following form:

(1) Set $k = 0$ and let $\boldsymbol{D}^{(0)}$ be a box that contains a zero of f and assume that f is invertible over $\boldsymbol{D}^{(0)}$.

(2) Model the functional dependence of f over $\boldsymbol{D}^{(k)}$ by an n-th order Taylor model $T^{(k)}$ with the reference point $m\left(\boldsymbol{D}^{(k)}\right)$.

(3) Compute a left-inverse Taylor model $S^{(k)}$ for $T^{(k)}$.

(4) Compute a new domain via $\boldsymbol{D}^{(k+1)} = \boldsymbol{D}^{(k)} \cap S^{(k)}(0)$. If the enclosure $\boldsymbol{D}^{(k+1)}$ of the zero is not accurate enough, replace k by $k+1$ and repeat from (2).

Obviously, for $n = 1$ this method is equivalent to the traditional interval Newton method with centered forms (c.f. Sec. 2.2 and [92, 94, 3]). However, the high order version of the interval Newton algorithm allows for a fast and accurate determination of guaranteed enclosures of the zeros of a given function. Since the accuracy of n-th order Taylor models scales with the $(n+1)$-st order of the domain size, this method converges very rapidly once the domain size has become sufficiently small. This fact will be illustrated further in Sec. 3.6.2. Using the notation of the previous algorithmic description, the following is an important consequence:

Theorem 3.6 (Interval Newton for Taylor Models). *Given a Taylor model T for f as before and assume that f is invertible over the domain $\boldsymbol{D}$. If f has a zero $x^* \in \boldsymbol{D}$ then, for $\boldsymbol{D}^{(0)} = \boldsymbol{D}$, it is $x^* \in \boldsymbol{D}^{(k)}$ for each $k \in \mathbb{N}_0$. On the other hand, if $\boldsymbol{D}^{(k)} \cap S^{(k)}(0) = \emptyset$ for any $k \in \mathbb{N}_0$, then f has no zeros in $\boldsymbol{D}$.*

Proof. We prove the first part by induction. For $k = 0$ the assertion is true by assumption, since $x^* \in \boldsymbol{D} = \boldsymbol{D}^{(0)}$. Now we assume that the assertion is true for some $k \in \mathbb{N}_0$. Then, since $x^* \in \boldsymbol{D}^{(k)}$ and $f \in T^{(k)}$, 0 is contained in the domain of $S^{(k)}$. Moreover, since $S^{(k)}$ is a left-inverse Taylor model for $T^{(k)}$, $\exists\, g \in S^{(k)}$ such that $g(f(x^*)) = x^*$. However, since $f(x^*) = 0$, that implies that $x^* \in S^{(k)}(0)$. Consequently, the first part of the theorem holds for all $k \in \mathbb{N}_0$.

The second part of the assertion follows from the first part: if there was a zero in $\boldsymbol{D}^{(k)}$ then, according to the first part, it would also be in $\boldsymbol{D}^{(k)} \cap S^{(k)}(0)$. However, if the intersection is empty, there was no zero to begin with. $\square$

It is important to note that, just like for the conventional interval Newton method, it is generally not possible to give rigorous estimates on the speed of convergence of this method. However, once the requirements of Thm. 3.6 have been satisfied, it will usually converge with the $(n+1)$-st order of the size of the inclusion $\boldsymbol{D}$; see also Sec. 3.6.2 for a further discussion. It should also be pointed out that unlike the extended Interval Newton method, the presented algorithm cannot handle local extrema of f within the domain. However, this can easily be circumvented by performing extended Interval Newton preconditioning steps that split the original domain until the Taylor models on the smaller boxes satisfy the requirements of Thm. 3.6.

3.4.1 Examples

The following examples demonstrate the Taylor model based high-order extension of the Interval Newton method. In particular, they show that the method converges extremely fast over relatively large domains.

One-Dimensional Fixed Points

This example compares the presented DA and Taylor model high-order extensions of the Newton method and the floating point and interval versions of that algorithm in a one-dimensional setting. It shows how the convergence rate of the high-order methods can indeed outperform the traditional algorithms by a wide margin.

The floating depth h of a cylindrical trunk of density $\rho = 0.66$ and radius r in water is given by

$$h = r\left(1 - \cos\left(\alpha_0/2\right)\right), \tag{3.34}$$

where $\alpha_0 = 3.655403079564624$ is the unique fixed point of

$$\alpha = \sin(\alpha) + 2\pi\rho. \tag{3.35}$$

This fixed-point relation for α_0 can readily be transformed into a root finding problem and the resulting function is shown in Fig. 3.8.

We used several Newton methods to determine the fixed point by solving the corresponding root finding problem: a traditional point Newton algorithm, an interval Newton method, a Taylor model based Newton method as presented in this section, and a non-verified point-polynomial version of the latter. The interval $[3.3, 4.3]$ has been the initial enclosure of the zero, and the midpoint 3.8 has been used as a starting value for the non-verified methods. The computations have been performed in order 19 with an accuracy goal of 10^{-14}. The final approximations and enclosures of α_0 are

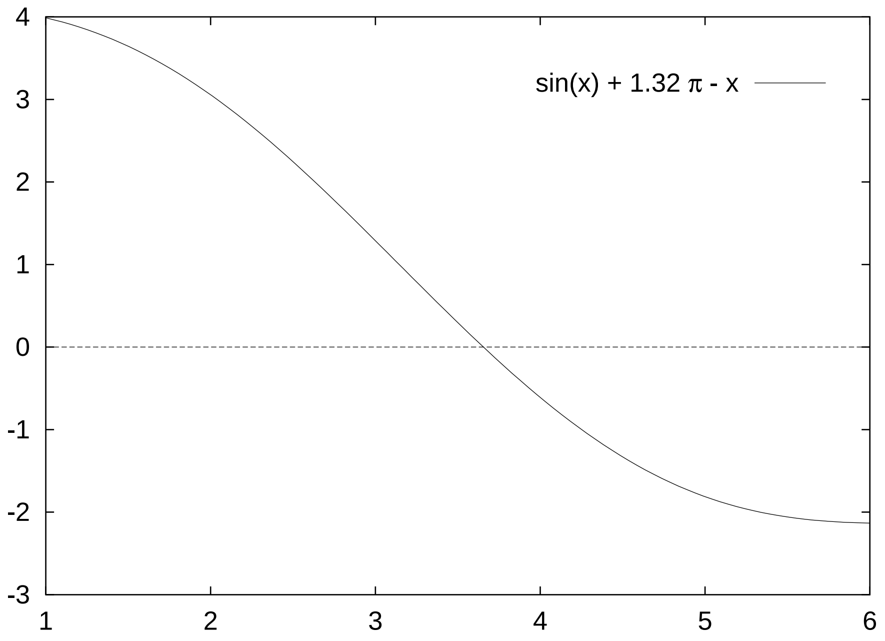


Figure 3.8: Transformed Fixed-Point problem: the fixed point α_0 of Eqn. (3.35) is the intersection point of the two graphs.

shown in Tab. 3.4 together with the number of iterations and CPU time required by the different methods.

Newton Method	*Result*	*Steps*	*Time*
Floating Point	3.655403079564624	4	0.0000166s
Polynomial	3.655403079564624	1	0.0016249s
Interval	[3.655403079564622, 3.655403079564625]	4	0.0000390s
Taylor model	[3.655403079564619, 3.655403079564628]	1	0.0048408s

Table 3.4: Comparison of different Newton methods for the determination of the fixed point α_0 of Eqn. (3.35).

It should be noted that the high-order methods reach the desired accuracy after just a single step. The traditional algorithms on the other hand need four iterations to achieve the desired precision for this rather simple function. However, since the computational complexity of Taylor models exceeds the one of intervals, and since all methods work without domain splitting, the traditional algorithms are still more efficient.

Polynomial Functions

The following one-dimensional example uses the verified Newton methods to enclose the value of π to almost machine epsilon — in fact the desired accuracy goal has been set to 10^{-12}. To that end, the sine function has been approximated by its 25-th order Taylor polynomial P, and the two Newton methods have been used to determine the unique zero of P in the interval $[1.8, 4]$. The order of the approximation has been chosen in such a way that the value of that unique zero and π agree up to machine epsilon.

$$P(x) = \sum_{k=0}^{12} (-1)^k \frac{x^{2k+1}}{(2k+1)!} \tag{3.36}$$

Due to the alternating sign of the terms, the naive implementation and evaluation of P exhibits strong cancellations. While it is clear that more efficient methods for the approximation of the sine function exist, this particular setup is typical for applications where the functional dependence is often given by complicated and redundant expressions that exhibit strong cancellation. While interval methods are known to have problems with this, the Taylor model approach can successfully avoid these problems. Fig. 3.9 shows the sine function and its 25-th order approximation between 0 and 12. In the region around 3 that is most interesting for this problem, the two functions agree up to machine precision.

This example illustrates how Taylor models can successfully control cancellation and how the high-order methods converge much faster than the regular first-order Newton methods: over the same initial domain, it takes the regular interval Newton method eight iterations to enclose the zero with the desired accuracy. On the other hand, Tab. 3.5 shows that the Taylor model method achieves a comparable accuracy in just a single step. Moreover, the interval Newton method had to resort to extended

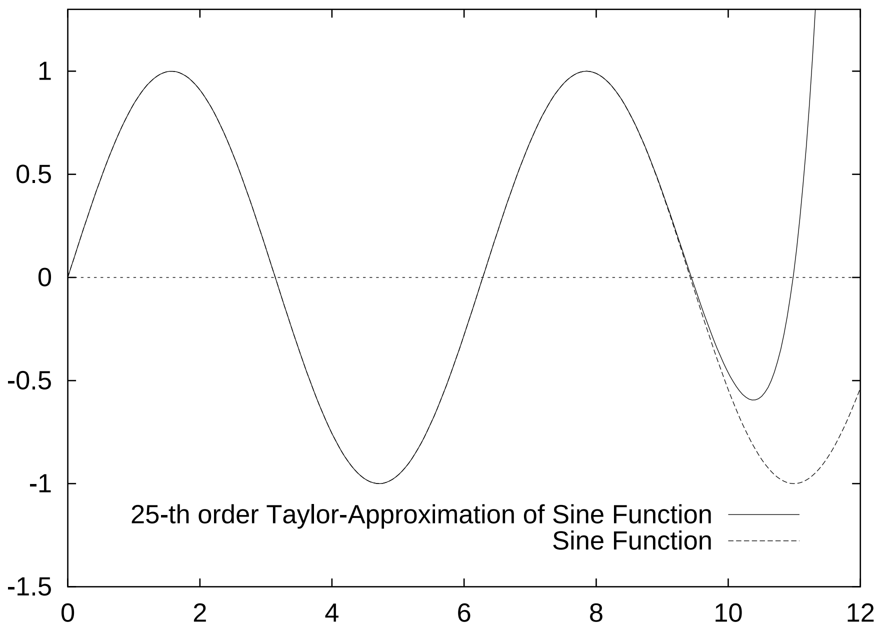


Figure 3.9: Taylor polynomial approximation of order 25 of the sine function over the interval $[0, 12]$.

interval divisions and subsequent domain splitting in the first four of the eight steps.

A Multidimensional Function

This example illustrates the ability of the Taylor model based Newton method to readily extend to higher dimensional problems. The example shows the determination of the zeros of the six-dimensional exponential function defined in Sec. 3.3.2 with initial enclosures of the zeros given by $[-0.25, 0.25]^6$ for Taylor models and $[-0.02, 0.02]^6$ for the conventional interval Newton method. The computation has been performed with 8-th order Taylor models and the desired accuracy has been set to 10^{-12}. The enclosures returned after each step of the iteration are listed in Tab. 3.6.

Unlike the conventional interval method, where the computation and inversion of the interval matrix for the derivative poses a significant challenge, the Taylor model based Newton method generalizes easily from the one-dimensional case to

Method	*Step*	*Enclosures of π*
Interval Newton	0	$[1.800000000000000, 4.000000000000000]$
	1	$[2.919077672711509, 4.000000000000001]$
	2	$[2.919077672711509, 3.427717355896160]$
	3	$[2.919077672711508, 3.165511666111006]$
	4	$[3.085798724266786, 3.165511666111007]$
	5	$[3.136629538119056, 3.154844930481172]$
	6	$[3.141102949424404, 3.141988434766514]$
	7	$[3.141592414707353, 3.141592894936869]$
	8	$[3.141592653589790, 3.141592653589801]$
Taylor models	0	$[1.800000000000000, 4.000000000000000]$
	1	$[3.141592653589765, 3.141592653589826]$

Table 3.5: Enclosures of π, obtained by the interval Newton and Taylor model methods.

higher dimensional problems. Furthermore, the rapid convergence of the method over a relatively large domain demonstrates the power of the high-order approach. It converges in only two steps to an extremely accurate enclosure of the zero. But even more striking, the second step improves the sharpness by about 11 orders of magnitude. Once again, this is a vivid example of how the accuracy of Taylor models scales with the $(n+1)$-st order of the domain size.

The interval Newton method on the other hand, reached the desired accuracy in eight steps; since the computational complexity of the Taylor models is about three orders higher than that of intervals, this improvement does not yet justify the use of Taylor models. However, the interval Newton method required a much smaller initial enclosure of the zero to converge. Combining this with the dimensionality of the problem and assuming that the domain has to be split uniformly in each coordinate direction, the Taylor model method turns out to be two orders of magnitude more efficient than the interval Newton. We will revisit this particular aspect of the method in Sec. 3.6.2.

Method	*Step*	*Magnitude of Enclosures of Zero*
Interval Newton	0	[-0.20000000000000E-001, 0.20000000000000E-001]
	1	[-0.20000000000000E-001, 0.20000000000000E-001]
	2	[-0.20000000000000E-001, 0.20000000000000E-001]
	3	[-0.20000000000000E-001, 0.20000000000000E-001]
	4	[-0.87410175378563E-002, 0.87410175378563E-002]
	5	[-0.96852367808498E-003, 0.96852367808498E-003]
	6	[-0.54933148362165E-005, 0.54933148362165E-005]
	7	[-0.82475281503554E-010, 0.82475281503554E-010]
	8	[-0.94355649697386E-014, 0.94355649697386E-014]
Taylor models	0	[-0.25000000000000E-000, 0.25000000000000E-000]
	1	[-0.47478831445046E-003, 0.47478831445046E-003]
	2	[-0.60171167482408E-014, 0.60171167482408E-014]

Table 3.6: Enclosures of the zero for the six-dimensional exponential function, computed with an interval Newton method and the high order Taylor model approach.

3.5 Implicit Equations

In this section we develop a method to compute Taylor model enclosures of functions that are described by implicit relations. This method has interesting applications in prescribed path control [58] and can even be used to obtain solution enclosures for implicit ordinary differential equations [59]. While the new approach uses the methods for computing inverse Taylor models discussed earlier, the method is based on the Implicit Function Theorem.

Theorem 3.7 (Implicit Function Theorem). *Given two open sets $U \subset \mathbb{R}^v$ and $V \subset \mathbb{R}^w$ and a function $f \in \mathcal{C}^{n+1}(U \times V, \mathbb{R}^w)$. Write $f(x, y)$ with $x \in U$ and $y \in V$ and suppose that $(x_0, y_0) \in U \times V$ and that $f(x_0, y_0) = 0$. If*

$$D\left(f|_{(\{x_0\}\times\mathbb{R}^w)\cap W}\right)(x_0, y_0) : \mathbb{R}^w \to \mathbb{R}^w \tag{3.37}$$

is a linear isomorphism, then there is an open set $U' \subset U \subset \mathbb{R}^v$ containing x_0, an open set $V' \subset V \subset \mathbb{R}^w$ containing y_0, and a unique $\mathcal{C}^{n+1}$ function $g : U' \to V'$ such *that for all $x \in U'$*

$$g\left(x_0\right) = y_0 \textit{ and } f\left(x, g\left(x\right)\right) = 0. \tag{3.38}$$

By combining this fundamental result of calculus with the presented methods for enclosing inverse functions, we are able to compute Taylor models for implicitly described functions g as above. Suppose that f satisfies the conditions of the Implicit Function Theorem. Then we define a function $\Phi : U' \times V' :\to \mathbb{R}^w$ by

$$\Phi \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ f(x,y) \end{pmatrix}. \tag{3.39}$$

Since f satisfies the Implicit Function Theorem, it follows that Φ satisfies the conditions of the Inverse Function Theorem around $(x_0, y_0) \in U' \times V'$. Thus, there are neighborhoods U'' of x_0 and V'' of y_0 such that the inverse $\Phi^{-1} : U'' \times V'' \to \mathbb{R}^w$ exists and is well defined. Then we use Φ^{-1} to compute an explicit expression for the implicitly described function g:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \Phi^{-1} \begin{pmatrix} x \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ g(x) \end{pmatrix}. \tag{3.40}$$

Thus, in the semi-algebraic framework of Taylor models, we can use an inversion algorithm to obtain rigorous enclosures of implicitly prescribed functions. The only drawback of this method is that augmenting the system requires additional variables that might not be available due to memory constraints. However, in many important applications like prescribed path control [58], the variable x represents one-dimensional time. Thus, the augmented system is often enlarged by only a single variable, which is usually within the workable range of the Taylor model methods.

We also mention that the technique of augmenting implicit systems has been applied extensively to first order in bifurcation theory [72] and to higher order for symplectic integration [11]. Other combinations of the Implicit Function Theorem and first order automatic differentiation [111, 52, 112] in the field of control theory are discussed by Evtushenko [42].

3.5.1 Curves, Surfaces, Manifolds

Hyperbolas are conic sections with an eccentricity $\varepsilon > 1$. Denoting the foci of a hyperbola by f_1 and f_2. Hyperbola with a semi-major axis of a are described by

$$|f_1 - P| - |f_2 - P| = \pm 2a, \tag{3.41}$$

where the two different signs denote the two branches of the hyperbola [61]. If we denote the distance between the origin and the foci by e; i.e., $e = |f_1|$, the eccentricity ε and the semi-major axis a are related by:

$$a \cdot \varepsilon = e. \tag{3.42}$$

Given these relations and notation, the semi-minor axis b and the parameter p are connected via

$$b^2 = e^2 - a^2 \quad \text{and} \quad pa = b^2. \tag{3.43}$$

We denote the angle between the x-axis and the vector connecting the focal point f_1 with the point (x, y) on the positive branch of the hyperbola by ϕ. Then in the special case of $a = e = 1$, the cartesian coordinates x and y and the parameters ε and ϕ are connected by

$$h_1(\varepsilon, \phi, x, y) = x^2 - \frac{y^2}{\varepsilon^2 - 1} - 1 = 0 \tag{3.44a}$$

$$h_2(\varepsilon, \phi, x, y) = (1 - \varepsilon \cos(\phi)) \cdot \sqrt{(x - \varepsilon)^2 + y^2} - \left(\varepsilon^2 - 1\right) = 0 \tag{3.44b}$$

In the remainder of this section, we use the approach outlined in Eqns. (3.39) and (3.40) to compute Taylor models containing charts for the cartesian coordinates

x and y as functions of the parameters ε and ϕ. The results show that the method can indeed be used to solve implicit equations and to compute parameterizations. Besides its applicability to static problems as discussed here, this method has an important application in prescribed path control, where the implicit relation connecting coordinates and control functions is obtained as the result of an ODE integration step [58].

To compute charts for x and y from Eqns. (3.44), we start with initial conditions of $\varepsilon_0 = 2$ and $\phi_0 = \pi/2$ and use a standard Newton method to determine *consistent initial conditions* x_0 and y_0 such that

$$h_1(\varepsilon_0, \phi_0, x_0, y_0) = h_2(\varepsilon_0, \phi_0, x_0, y_0) = 0. \tag{3.45}$$

Once the consistent initial conditions $x_0 = 2$ and $y_0 = 3$ have been determined to machine precision, we model the functions h_1 and h_2 by 19-th order Taylor models T_1 and T_2 with reference point $(\varepsilon_0, \phi_0, x_0, y_0)$ and domain

$$\boldsymbol{D} = [1.99, 2.01] \times [1.56079633, 1.58079633] \times [1.95, 2.05] \times [2.88, 3.12]. \tag{3.46}$$

The resulting Taylor models have remainder bounds with a width in the order of the machine precision 10^{-16} and their reference polynomials have 59 and 4284 coefficients, respectively.

After augmenting the system and defining the function Φ according to Eqn. (3.39), we use Taylor model inversion and Eqn. (3.40) to obtain Taylor models for x and y as functions of ε and ϕ. The resulting Taylor models are defined over the domain

$$\boldsymbol{D'} = [1.99, 2.01] \times [1.56079633, 1.58079633], \tag{3.47}$$

have reference polynomials with 210 and 205 non-vanishing coefficients, and remainder bounds

$$R_1 = [-0.635775672959 \times 10^{-8}, 0.635776786515 \times 10^{-8}], \tag{3.48a}$$

$$R_2 = [-0.537150447777 \times 10^{-8}, 0.537151554117 \times 10^{-8}]. \tag{3.48b}$$

Fig. 3.10 shows the resulting surfaces $x(\varepsilon, \phi)$ and $y(\varepsilon, \phi)$ over the domain box $\boldsymbol{D}'$.

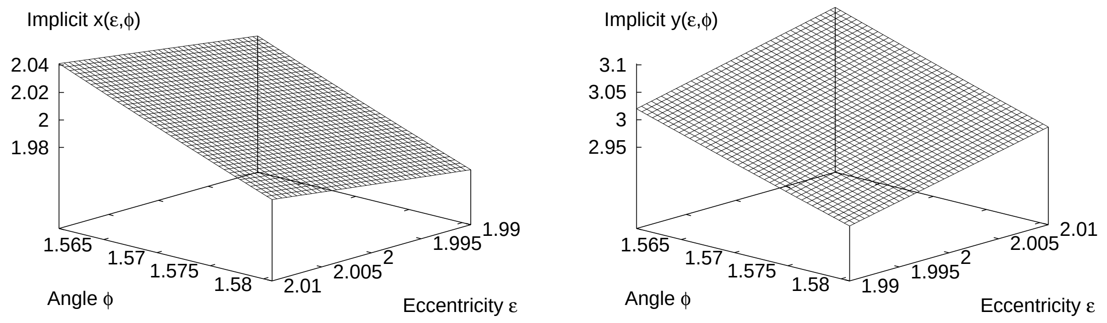


Figure 3.10: Charts for the cartesian coordinates x and y of the two-parameter description Eqns. (3.44) for hyperbolas. Remainder bounds are in the order of 10^{-8}.

This example demonstrates how the Taylor model inversion tools can indeed be used for the computations of implicit functions and charts of smooth manifolds. While the Implicit Function Theorem can guarantee the existence of local parameterizations and is therefore of great theoretical value, the presented method allows rigorous computations of these parameterizations. As illustrated by the example, the resulting Taylor models offer a highly accurate description of the charts of the implicitly described manifolds. The ability to rigorously solve implicit equations with small overestimation is of great interest to fields as diverse as prescribed path control [58] and differential algebraic equations [59].

3.6 Analysis of Inversion Algorithms

Before we close this chapter on Taylor model based inversion methods, we use the previously presented examples to study some of the complexity and performance characteristics that distinguish conventional interval methods and Taylor model techniques. While the theoretical limits that govern Taylor model computations have already been discussed in Sec. 2.3, here we focus on practical results to underline and illustrate these aspects.

3.6.1 Space and Time Requirements

The computational complexity of individual Taylor model operations exceeds the cost of the corresponding standard interval arithmetic by a wide margin. For example, the storage size S of an n-th order Taylor model in v variables scales approximately as

$$S = \begin{pmatrix} n+v \\ v \end{pmatrix} = \frac{(n+v)!}{v!\,n!}. \tag{3.49}$$

Moreover, while the cost of simple operations like addition and assignment scales linear with S, multiplication and intrinsic functions scale with higher orders of S [83, 22, 82]. However, since Taylor models can often avoid domain splitting, their real strength lies in high-dimensional problems with large domains. As an example, consider enclosing the inverse of a one-dimensional function as illustrated in Fig. 3.11. Using only intervals, this requires domains of the size of the desired accuracy, since the number of sub-intervals has to scale with the inverse of the desired accuracy.

As another example, consider the six-dimensional exponential function presented in Sec. 3.3.2. The computational cost of the 8-th order Taylor models is about 1500 times higher than that of conventional intervals. However, achieving a uniform inclusion accuracy of 10^{-11} with intervals, as is often needed for root finding problems, would require approximately 10^9 sub-intervals in each dimension: resulting in 10^{54} separate interval evaluations, which would make it impossible to use conventional interval methods for that task. Thus, the advantages of Taylor models often outweigh their additional costs over naive interval techniques if enclosures of high precision are

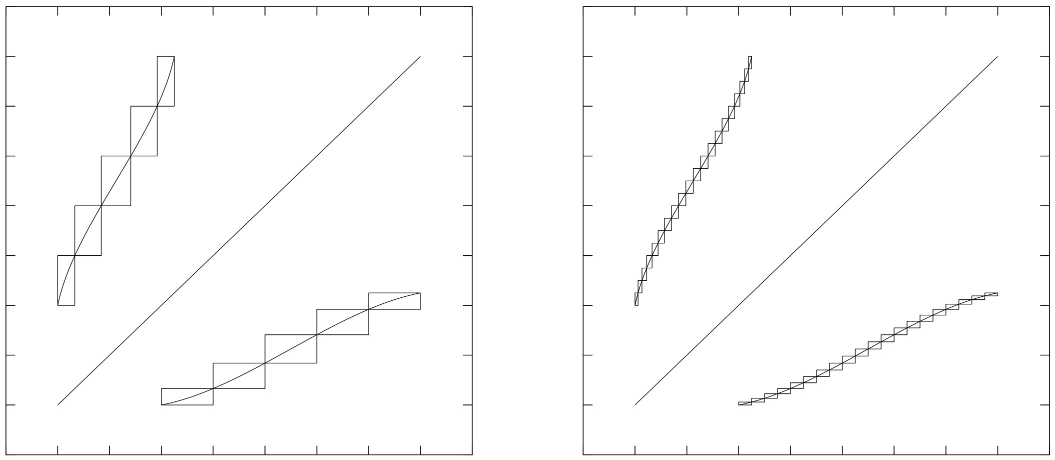


Figure 3.11: Illustration of verified inversion with intervals. The sharpness of the enclosures scales linearly with the magnitude of the domain sub-intervals.

desired.

As a general rule of thumb, if conventional interval methods and Taylor models compete over the same domain and box-splitting is not required to achieve the necessary sharpness, the simple interval methods will almost always be more efficient. But if using intervals requires splitting of the domains, as is frequently the case, the use of Taylor models will generally become favorable.

3.6.2 Order Dependence

This final example of the chapter illustrates how the accuracy of Taylor model approaches indeed scales with the $(n+1)$-st order of the domain size as shown in Thm. 2.6. For the six-dimensional exponential function defined in Sec. 3.3.2, Fig. 3.12 shows the magnitude of $\boldsymbol{D}^{(1)}$ after just one iteration of the 6-th order Taylor model Newton method as a function of $\boldsymbol{D}^{(0)} = [-0.25, 0.25]$.

The data illustrate how that accuracy of the method scales with the 7-th order of the initial domain $\boldsymbol{D}^{(0)}$. For purposes of illustration, a line of slope seven has been fitted to the data points. Deviations from that theoretical line for small domains are due to limits imposed on the verified computations by the machine epsilon. For large initial domains on the other hand, $\boldsymbol{D}^{(1)}$ is bigger than the theoretical line would imply, caused by a decreasing accuracy of the 6-th order Taylor polynomial approximations over the initial domain. Hence, for large domains the Taylor models operate in a region where the rules of $(n+1)$-st order convergence fail to apply.

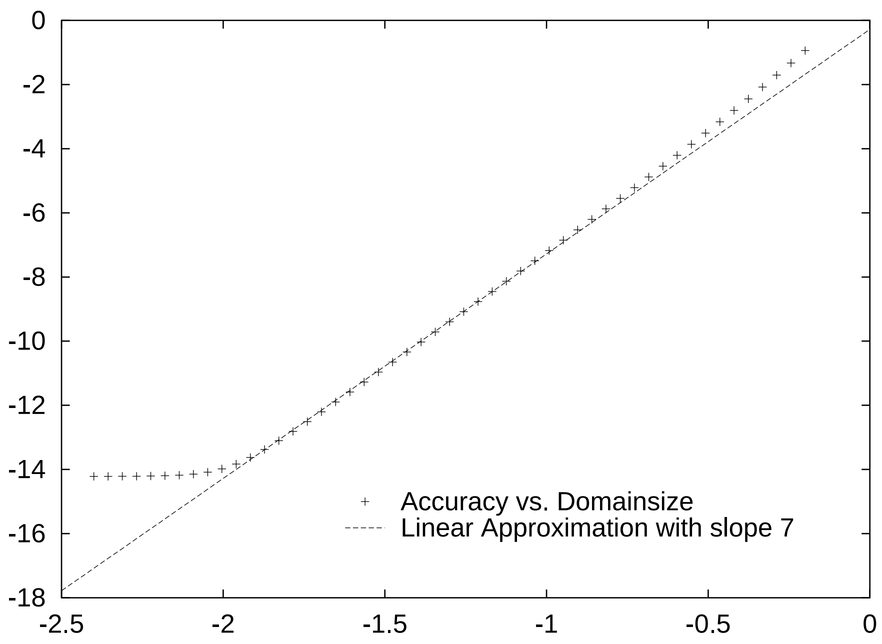


Figure 3.12: Double-logarithmic plot of the accuracy of the first Taylor model Newton step as a function of initial domain size; results for 6-th order Taylor models.

3.7 Summary

In this chapter we have presented new Taylor model based methods to rigorously answer the question of whether a function is invertible over a certain domain of interest. This method uses only first derivatives and as such it can be implemented with little computational expense.

As a next step, we developed a new method for computing rigorous enclosures of inverse functions of arbitrary complicated multidimensional functions. Its applicability and performance has been demonstrated in the computation of verified enclosures of the inverse function for given invertible functions. The method combines Taylor models with the method for proving invertibility and efficient differential algebra algorithms that allow the determination of exact n-th order polynomial inverses in finitely many steps.

Using Taylor models, we have been able to overcome some limitations of conventional interval based methods to determine invertibility. Namely we have successfully shown that Taylor model based methods can model computationally complex functions much more accurately than interval methods. Since the accuracy of Taylor models scales with the $(n+1)$-st order of the domain size, we have shown that the new methods work over significantly larger domains and scale better to high dimensional problems than interval based methods. Moreover, the newly developed method can handle complicated functions by using Taylor models which have been shown to control the dependency and cancellation problems inherent in regular interval arithmetic [85]. This is especially important in the establishment of invertibility prior to the computation of the inverse Taylor model itself.

As an immediate application of the new method, an extension of the conventional interval Newton method has been presented. It has been demonstrated to converge much faster than the regular interval-based version of that algorithm. Moreover, combining the inversion tools with a fresh look at the Implicit Function Theorem resulted in a new method for the rigorous computation of implicitly described functions. Important applications of the new methods are in the computation of solutions of differential algebraic equations (c.f., Chap. 4 and [4, 107, 59]), the determination of existence of generating functions (c.f., Sec. 5.2), and the stability analysis of asteroid orbits near the Lagrangian points [133].

Chapter 4

Differential Algebraic Equations

Under certain conditions, the solutions of ordinary differential equations (ODEs) and differential algebraic equations (DAEs) can be expanded in Taylor series in the independent variable and the initial conditions. In these cases, we can obtain good approximations of the solutions by computing the respective Taylor series [31, 32]. Moreover, the Taylor model approach often allows us to compute rigorous enclosures of the solutions of initial value problems [24, 82].

By using structural analysis [105, 108] and differentiation, it is often possible to transform a given DAE into an equivalent system of implicit ODEs. If the derived system is described by a Taylor model, representing each derivative by an independent variable, verified inversion methods discussed in the previous chapter can be utilized to solve for the highest derivatives as functions of lower order ones. The resulting Taylor model forms an enclosure of the right hand side of an explicit ODE initial value problem that is equivalent to the original DAE. While this explicit system is suitable for integration with Taylor model solvers [24], the intermediate inversion often requires a substantial increase in the dimensionality of the problem, limiting the approach to relatively small systems. An application of this inversion-based integration of differential equations has been discussed in [59].

In this chapter we derive a method for the verified integration of general implicit ODEs that is based on the observation that solutions can be obtained as fixed points of a certain operator containing the antiderivation. We show that this operator is particularly well suited for practical applications in Taylor model settings since its restriction to DA vectors is guaranteed to converge to the exact solution in at most $n+1$ steps, where $n \in \mathbb{N}$ is the order of the Taylor models.

While sophisticated methods have been developed for the numerical integration of DAEs [51, 30, 53, 4], they are usually based on multistep methods that generally do not provide any means of verification and validation. However, since the new method also allows the computation of the differentiation index ν_d of DAEs [4], it can be utilized for a verified index analysis. Combining this with a more traditional structural analysis of DAEs [105, 108], we get a scheme for transforming DAEs into implicit ODEs, which can then be solved with the new method. Since this combination uses verified Taylor models at every stage of the computation, it can be used to compute Taylor model enclosures of the solutions of DAEs. Additionally, by utilizing high order Taylor models ($n > 20$ is not uncommon), the scheme can even be applied to high-index problems that pose serious challenges to existing non-verified DAE integration methods.

4.1 Background and Motivation

The standard first order initial value problem is usually associated with an explicit ordinary differential equation of the form

$$x' = f(t, x). \tag{4.1}$$

However, a more general first order ODE problem is given by the implicit form

$$F(t, x, x') = 0, \tag{4.2}$$

where the Jacobian

$$\frac{\partial F(t,u,v)}{\partial v} \tag{4.3}$$

is assumed to be non-singular for all arguments in an appropriate domain. According to the Implicit Function Theorem, it is then possible to turn the problem into an explicit first order system that is as smooth as the original problem. Thus, if F was sufficiently smooth to begin with, the implicit system is guaranteed to have a smooth solution. However, the solutions of implicit ODEs are not guaranteed to be unique. To illustrate this, consider the following initial value problem for an implicit ODE

$$(x')^2 = x^2, \quad x(0) = 1. \tag{4.4}$$

Obviously $x_+(t) = e^t$ and $x_-(t) = e^{-t}$ are both valid solutions to this problem, illustrating that smooth implicit systems may indeed have multiple smooth solutions. We will later address the implications of this phenomenon by introducing the concept of *consistent points*, which will enable us to guarantee the local uniqueness of solutions of implicit ODEs and DAEs.

4.1.1 First Order Differential Algebraic Equations

ODEs with constraints are another extension of the regular explicit first order ODE problem

$$x' = f(t,x,z) \tag{4.5a}$$

$$0 = g(t,x,z). \tag{4.5b}$$

Unlike in the case of standard ODEs, $x(t)$ is not only a solution to the differential part (4.5a), but is also forced to satisfy the algebraic constraint condition (4.5b). While the system (4.5) is given in the so-called semi-explicit form, it can easily be rewritten in an implicit form by introducing the variable $\xi = (x, z)^T$.

$$F(t, \xi, \xi') = \begin{pmatrix} x' - f(t, x, z) \\ g(t, x, z) \end{pmatrix} = 0. \tag{4.6}$$

However, unlike in the case of implicit ODEs, the resulting Jacobian matrix

$$\frac{\partial F(t, u, v)}{\partial v} = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \tag{4.7}$$

is no longer regular.

Generalizing the previous example and the definition of the general first order implicit ODE (4.2), we write the general first order differential algebraic equation (DAE) as

$$F(t, x, x') = 0, \tag{4.8}$$

where F is a sufficiently smooth function defined in an appropriate domain. In the case of DAEs we do not impose any additional restrictions on the regularity of the Jacobian (4.3). In fact, the rank and the structure of the Jacobian may even depend on the solution $x(t)$. However, for simplicity we will always assume that the rank and the structure of the Jacobian along a single solution $x(t)$ are independent of t.

The class of DAEs contains all ODEs and all algebraic equations, as well as a large number of problems between these two extremes. As such, DAEs can generally not be solved by simple algebraic means. Moreover, unlike the case of ODEs, even smooth DAEs do not necessarily have solutions. It is interesting that the first results on the existence and uniqueness of solutions of general DAEs have only recently appeared [109, 110]. While these results are based on theoretical arguments of differential geometry, in Sec. 4.3 we will present results that guarantee the existence of solutions to a large class of DAEs based on direct computations.

The most common approach of determining the solution of a given DAE is to differentiate the system until we can pick v equations from the enlarged system such that the Jacobian (4.3) of this new system is regular. The resulting implicit ODE system can often be solved for a solution of the original DAE problem. This basic idea is formalized by the definition of the index [4]:

Definition 4.1 (Differentiation Index). *For the general DAE system (4.8), the* differentiation index ν_d *along a solution* $x(t)$ *is the minimum number of differentiations of the system which would be required to solve for* x' *uniquely in terms of the dependent variable* x *and the independent variable* t*.*

Thus, the index ν_d is defined in terms of the overdetermined system

$$\begin{aligned} F(t,x,x') &= 0 \\ F'(t,x,x',x'') &= 0 \\ &\vdots \\ F^{(p)}(t,x,,x',\ldots,x^{(p+1)}) &= 0 \end{aligned} \tag{4.9}$$

to be the smallest integer p so that (4.9) can be solved for the highest derivatives in terms of lower order derivatives and t. In the following, the index will indicate the differentiation index unless we compare to other concepts of indices in which case we will use the fully qualified name.

It might be somewhat of a surprise that the definition of the index explicitly refers to a solution of (4.8). However, while we will always assume that the index does not change along a single solution; i.e., is independent of the independent variable t, a given system can have different indices along different solutions. To illustrate this, consider the system

$$0 = x' - z \tag{4.10a}$$

$$0 = y(1-y) \tag{4.10b}$$

$$0 = xy + z(1-y) - t \tag{4.10c}$$

Assuming continuous solutions, the second equation permits exactly two solutions: $y_\alpha(t) = 0$ and $y_\beta(t) = 1$. Depending on which of the solutions is compatible with the initial conditions, we get two different scenarios:

α: The system reduces to $x' = z$ and $0 = z - t$; it has index 1 with the solution $(x_0 + t^2/2, 0, t)^T$.

β: The system reduces to $x' = z$ and $0 = x - t$; it has index 2 with the solution $(t, 1, 1)^T$.

While this example shows that the index does indeed depend on the solution of the DAE, it also illustrates how the solution manifold of a DAE can vary with the index. While any sufficiently smooth first order ODE system of size v has exactly v degrees of freedom, the solution manifold of the DAE system (4.8) can have any dimensionality between 0 and v.

By permitting only discrete values for y, the previous example highlights how the constrained nature of DAEs even extends to the initial conditions. While the initial conditions for ODEs are usually free to vary over large ranges, the initial conditions of DAE problems have to be consistent with the constraints. However, frequently the consistent initialization of a DAE problem is not obvious: problems with an index larger than one have *hidden constraints* which are only revealed by the intermediate derivatives in (4.9). Thus, while the highest derivatives are needed to transform the system into a solvable ODE, all lower order derivatives are describing the constraint manifold and the consistent initial conditions.

Standard DAE Problems

Short of testing all possible combinations of derivatives, Def. 4.1 provides us with no efficient algorithm for computing the index of the DAE (4.8). While we will revisit this issue in Sec. 4.3, many practical DAEs can be written in standard Hessenberg forms with known index, separating the ODE part and the algebraic constraints.

Definition 4.2 (Hessenberg Index-1). *The differential algebraic equation*

$$x' = f(t, x, z) \tag{4.11a}$$

$$0 = g(t, x, z), \tag{4.11b}$$

with the Jacobian matrix function $\partial g/\partial z$ assumed to be regular for all t, is in Hessenberg Index-1 *form.*

DAEs in Hessenberg Index-1 form are also called *semi-explicit index-1* systems. Since the constraints can in principle be solved for the algebraic variable z, these systems are closely related to implicit ODE problems. Another important class of DAEs is given by the Hessenberg Index-2 problems.

Definition 4.3 (Hessenberg Index-2). *The differential algebraic equation*

$$x' = f(t, x, z) \tag{4.12a}$$

$$0 = g(t, x), \tag{4.12b}$$

with the product of the Jacobian matrices $(\partial g/\partial x)\,(\partial f/\partial z)$ assumed to be regular for all t, is in Hessenberg Index-2 *form.*

We note that in many index-2 problems the algebraic quantity z plays the role of a Lagrange multiplier. Thus, many problems of classical mechanics can easily be written in the form of index-2 DAEs. An example of this will be discussed in Sec. 4.4.2.

4.1.2 General Differential Equations

Up to this point, we have exclusively dealt with systems of first order differential equations. This is commonly justified by the observation that any n-th order differential equation in v variables can be transformed into an equivalent first order systems with up to $n \times v$ variables. This approach is known as *order reduction* and has many advantages for the theoretical analysis of general ODE problems. However, the increased size of the problems often challenges verified numerical integration methods. In the remainder of this chapter we will therefore focus on methods for the direct numerical integration of general high-order differential equations.

For a definition of the most general DAE problem, we consider the independent scalar variable t, the v dependent variables $x_j = x_j(t)$, and sufficiently smooth functions f_i for $i = 1, \dots, v$, and form the system

$$\begin{aligned} f_1\left(t, x_1, \dots, x_1^{(\xi_{11})}, \dots, x_v, \dots, x_v^{(\xi_{1v})}\right) &= 0 \\ &\vdots \\ f_v\left(t, x_1, \dots, x_1^{(\xi_{v1})}, \dots, x_v, \dots, x_v^{(\xi_{vv})}\right) &= 0. \end{aligned} \tag{4.13}$$

For a given j, the i-th equation of this system does not necessarily depend on x_j, the derivatives $x_j^{(\eta)}$ for $\eta < \xi_{ij}$, or even any of its derivatives. However, if there is a dependence on at least one of the derivatives (including the 0-th derivative $x_j^{(0)} = x_j$), we denote the order of that highest derivative by ξ_{ij}.

4.2 Verified Integration of Implicit ODEs

While sophisticated general-purpose methods for the verified integration of explicit first order ODEs have been developed [79, 78, 82, 24, 97, 99], none of these can be readily used for the verified integration of implicit ODEs, let alone differential algebraic equations. As a first step toward the verified integration of general DAEs, we consider the problem of integrating the implicit ODE initial value problem (4.2). Unlike in standard ODE integration methods, we will not limit ourselves to first order problems, but will eventually be able to integrate the arbitrary order problem

$$F\left(t, x, x', \dots, x^{(p)}\right), \tag{4.14}$$

where the derivative of F with respect to the highest derivatives in each of the components is assumed to be non-singular for all argument values in an appropriate domain.

The new integration method is based on an extended use of antiderivation. To motivate the combination of high order methods and antiderivation for the verified integration of general differential equations, consider the explicit second order ODE initial value problem

$$x'' = f\left(x, x', t\right), \quad x(t_0) = x_0, \quad x'(t_0) = x'_0. \tag{4.15}$$

While the conventional approach to solving this system is based on order reduction to a two-dimensional first order problem, in the framework of Taylor models we can use the intrinsic antiderivation and substitute $\xi = x'$; i.e.,

$$x(t) = x_0 + \int_{t_0}^{t} \xi(\tau) d\tau. \tag{4.16}$$

After inserting the expanded expression for $x(t)$ into (4.15), we obtain an explicit first order ODE for ξ.

$$\xi' = F\left(\xi, t\right) = f\left(x_0 + \int_{t_0}^{t} \xi(\tau) d\tau, \xi, t\right). \tag{4.17}$$

This system can readily be integrated with the existing Taylor model based integration scheme discussed in Sec. 2.3.3, and the solution $x(t)$ of Eqn. (4.15) can be computed from the Taylor model for $\xi(t)$ by a final application of antiderivation.

$$x = x_0 + \int_{t_0}^{t} \xi(\tau) d\tau. \tag{4.18}$$

While this approach seems obvious from a theoretical point of view, numerical analysis has traditionally avoided explicit references to antiderivation in its algorithms since antiderivation is usually not available in the classical computational frameworks. Only recently, with an increased use of symbolic and semi-symbolic tools like Mathematica and COSY Infinity, has the explicit use of antiderivation in numerical computations become feasible. From a practical point of view, this approach has the advantage of reducing the computational complexity of the right hand side of the ODE (4.15), often allowing for favorable computations.

4.2.1 High Order Taylor Model Solutions

In this section we present a Taylor model based algorithm for the verified integration of the general first order ODE initial value problem

$$F(t, x, x') = 0, \quad x(t_0) = x_0. \tag{4.19}$$

While we will later extend the algorithm to higher order ODEs, for now we assume that the problem is stated as an implicit first order system with an $(n + 2)$-times continuously differentiable v-dimensional function F and regular Jacobian matrix

$$\frac{\partial F(t, u, v)}{\partial v} \tag{4.20}$$

in appropriate domains.

The method discussed in this section uses techniques and tools based on the differential algebra ${}_nD_v$ and the Taylor model approach presented in Chap. 2. Utilizing DA vectors and Taylor model methods for the verified integration of initial value problems allows the propagation of initial conditions by not only expanding the solution in time, but also in the transverse variables [24]. By representing the initial conditions as additional DA variables, their dependence can be propagated through the integration process, allowing Taylor model based integration schemes to reduce the wrapping effect to high orders [86]. Moreover, in the context of the new algorithm, expanding the consistent initial derivative in the transverse variables further reduces the wrapping effect and allows the system to be rewritten in a derivative-free, origin preserving form suitable for verified integration with Taylor models.

Algorithm: Taylor Model Integration of Implicit ODEs

For notational convenience, we write the initial condition of the implicit ODE problem (4.19) as x_0, and denote functional dependence on the initial condition x_0 in a suitable domain by y. With these conventions, a single n-th order integration step of the implicit first order ODE (4.19) consists of the following sub-steps:

1. Using a suitable numerical method like the Newton method discussed in Sec. 3.4, solve the implicit system
$$F(t_0, y, x') = 0 \tag{4.21}$$
for a consistent initial condition $x'(t_0, y) = x'_0(y)$.

2. Utilizing antiderivation, rewrite the original problem in a derivative-free form
$$\Phi(t, y, \xi) = F\left(t, y + \int_{t_0}^{t} \xi(\tau, y) d\tau, \xi\right) = 0, \tag{4.22}$$
where $\xi = \xi(t, y) = x'(t, y)$ has been substituted for the derivative of x.

3. Translate the problem into an origin-preserving form by substituting $\zeta(t, y) = \xi(t, y) - x'_0(y)$ to obtain a new function
$$\Psi(t, y, \zeta) = \Phi\left(t, y, \zeta(t, y) + x'_0(y)\right). \tag{4.23}$$

4. Since $\Psi(t_0, x_0, 0) = 0$, within the differential algebraic framework it is possible to write the first order truncation of Ψ without the constant part as
$$\Psi(t, y, \zeta) =_1 L_\zeta(\zeta) + L_R(t, y), \tag{4.24}$$
where L_ζ and L_R denote the linear parts in ζ and (t, y), respectively.

5. If L_ζ is regular, transform the previous expression into an equivalent fixed point formulation for ζ.

$$\zeta(t, y) = \mathcal{H}(\zeta) = -L_\zeta^{-1} \left(\Psi(t, y, \zeta) - L_\zeta(\zeta)\right). \tag{4.25}$$

6. Using the operator $\mathcal{H}$, define a sequence (a_ν) of DA vectors in ${}_nD_{1+v}$ by $a_0 = 0$ and

$$a_{\nu+1} = \mathcal{H}(a_\nu). \tag{4.26}$$

   Then define the polynomial $P(t, y) = a_{n+1}$.

7. Construct a Taylor model T with the reference polynomial P over an appropriate domain $\boldsymbol{T} \times \boldsymbol{D} \subset \mathbb{R} \times \mathbb{R}^v$ containing the reference point (t_0, x_0) such that

$$\mathcal{H}(T) \subset T. \tag{4.27}$$

8. Compute a Taylor model X from T by using the relation

$$x(t, y) = y + \int_{t_0}^{t} \left(\zeta(\tau, y) + x_0'(y)\right) d\tau. \tag{4.28}$$

As we will see in the remainder of this section, the final Taylor model X is in fact a Taylor model for the flow $x(t, y)$ of the original ODE problem (4.19) for the particular choice of the initial derivative.

4.2.2 Mathematical Background

The algorithm presented in the previous section rests on several non-trivial assertions that will be proven here. We provide its mathematical foundation and establish the basis for the discussion in the next subsection. The proofs can be split into two groups: the differential algebraic part of the algorithm and the Taylor model results. Implementation and user interface issues of the algorithm will be discussed in the next section.

Differential Algebraic Results

In this section we discuss the differential algebraic results needed to justify the presented algorithm for the verified integration of implicit ODEs. First, we show that the operator $\mathcal{H}$ introduced above is well defined and DA-contracting. We then show that up to an additive constant, its unique fixed point lies in the same equivalence class as the derivative of the flow of the original ODE problem.

Lemma 4.1. *The operator $\mathcal{H}$ given by Eqn. 4.25 is a contracting operator and self map on the set $M = \{a \in {}_nD_{1+v} \mid \lambda(a) > 0\}$.*

Proof. To first order regularity of L_ζ is equivalent to the assumed regularity of the Jacobian $\partial F(t,u,v)/\partial v$.

$$\frac{\partial\Psi}{\partial\zeta}(t_0, y_0, \zeta) = \frac{\partial\Phi}{\partial\zeta}(t_0, y_0, \zeta + x_0') =_1 \frac{\partial F}{\partial\zeta}(t_0, x_0, \zeta) \tag{4.29}$$

Thus by assumption, the linear map L_ζ is regular in a neighborhood of the initial conditions and the operator $\mathcal{H}$ is therefore well defined on all of ${}_nD_{1+v}$.

To show that $\mathcal{H}$ is contracting on the subset M of nilpotent DA vectors, let $a, b \in M$ be given and assume that a and b agree up to order k. Since L_ζ^{-1} is invertible, it suffices to show that

$$\lambda\left((\Psi(t_0, y_0, a) - L_\zeta(a)) - (\Psi(t_0, y_0, b) - L_\zeta(b))\right) > k. \tag{4.30}$$

Since $\Psi(t_0, x_0, 0) = 0$, the map $\Psi(t_0, x_0, \zeta)$ is origin-preserving and can be written as

$$\Psi(t_0, x_0, \zeta) = L_\zeta(\zeta) + L_R(t_0, x_0) + \mathcal{N}(t_0, x_0, \zeta), \tag{4.31}$$

where $\mathcal{N}$ is a purely non-linear function. Thus, it suffices to show that $\mathcal{N}$ is contracting. However, if a and b agree up to order k, their images $\mathcal{N}(t_0, x_0, a)$ and $\mathcal{N}(t_0, x_0, b)$ trivially agree up to order $k+1$. Finally, since Ψ is origin-preserving, $\mathcal{H}$ is indeed a self map of M, and therefore a contracting operator on M. $\square$

While this lemma guarantees the existence of a unique fixed point of the operator $\mathcal{H}$, the next theorem summarizes the main result of the DA part of the presented algorithm.

Theorem 4.1. *If we denote the flow of the implicit first order ODE initial value problem (4.19) by $x(t, y)$, then the fixed point of $\mathcal{H}$ is a representative for*

$$[x'(t, y) - x_0'(y)]_n \,. \tag{4.32}$$

around the expansion point (t_0, x_0).

Proof. In principle, this assertion follows from the construction of the operator $\mathcal{H}$. However, in the following we summarize observations on existence, uniqueness, and smoothness of the solutions to the IVP (4.19) that are required for a full justification.

Since the original function F is of class $\mathcal{C}^{n+2}$ and its Jacobian matrix is assumed to be non-singular over a suitable region containing (t_0, x_0, x_0'), at least one solution to the initial value problem exists. Moreover, once a consistent initial derivative x_0' has been fixed, the Inverse Function Theorem guarantees the existence of a unique solution $x(t, y)$ in a neighborhood of (t_0, x_0, x_0'). The solution is the unique solution to an explicit first order system, which can in principle be derived from the original implicit problem. Since the Inverse Function Theorem guarantees that the explicit system is also of class $\mathcal{C}^{n+2}$ in a neighborhood of the consistent initial conditions, Thm. 5.2 ensures that the flow $x(t, y)$ is a $\mathcal{C}^{n+2}$ function of its variables. Thus, the equivalence class of its derivative is well defined in $\mathcal{C}^{n+1}$, and since the solution ζ is unique, the fixed point of $\mathcal{H}$ is indeed a representative for that class. $\square$

Taylor Model Results

In this section we prove the main Taylor model result needed for the presented algorithm: the self-inclusion of the Taylor model in (4.27) is a sufficient condition to guarantee the enclosure of a fixed point of Eqn. (4.25).

Definition 4.4 (L-Taylor Model). *Let $T = (P, x_0, \boldsymbol{D}, R)$ be an n-th order Taylor model and let $L > 0$ be a Lipschitz constant for P over $\boldsymbol{D}$. Then the L-Taylor model T_L is the set of all functions $f \in \mathcal{C}^0(\boldsymbol{D}, \mathbb{R}^w)$ such that*

1. *$f(x) - P(x) \in R$ for all $x \in \boldsymbol{D}$,*
2. *$f(x_0) = P(x_0)$,*
3. *$|f(x) - f(y)| \leq L|x - y|$ for all $x, y \in \boldsymbol{D}$.*

First, it should be noted that this gives a well defined and non-empty set of functions, since $P \in T_L$. And while there is an obvious connection to normal Taylor models, L-Taylor models contain a different set of functions:

1. Taylor models are "more restrictive" than the corresponding L-Taylor models, since the latter may contain non-differentiable functions.
2. Taylor models are also "less restrictive" than the corresponding L-Taylor models, since they do not pose any limits on the Lipschitz constant of its members.

Finally, unlike standard Taylor models, L-Taylor models are subsets of a Banach space, namely $\mathcal{C}^0(\boldsymbol{D}, \mathbb{R}^w)$.

Lemma 4.2. *The L-Taylor model T_L defined as above is a convex subset of the Banach space $\mathcal{C}^0(\boldsymbol{D}, \mathbb{R}^w)$ [82].*

Proof. Given two functions f_0 and f_1 in T_L and $t \in [0,1]$, define $f_t = t \cdot f_1 + (1-t) \cdot f_0$. Since the sum of continuous functions is continuous, $f_t \in \mathcal{C}^0(\boldsymbol{D}, \mathbb{R}^w)$ for any $t \in [0,1]$. Moreover, by convexity of R

$$f_t(x) - P(x) = (t \cdot f_1(x) + (1-t) \cdot f_0(x)) - P(x) \in R \tag{4.33}$$

for any $t \in [0,1]$. Since

$$\begin{aligned} |f_t(x) - f_t(y)| &= |t \cdot f_1(x) + (1-t) \cdot f_2(x) - t \cdot f_1(y) - (1-t) \cdot f_2(y)| \\ &\leq t \cdot |f_1(x) - f_1(y)| + (1-t) \cdot |f_0(x) - f_0(y)| \qquad (4.34) \\ &\leq t \cdot L + (1-t) \cdot L = L, \end{aligned}$$

T_L is indeed a convex subset of the Banach space $\mathcal{C}^0(\boldsymbol{D}, \mathbb{R}^w)$. □

Lemma 4.3. *The L-Taylor model T_L defined as above is a compact subset of the Banach space $\mathcal{C}^0(\boldsymbol{D}, \mathbb{R}^w)$ [82].*

Proof. It suffices to show that every sequence (f_ν) in T_L has at least one limit point in T_L. According to the Arzela-Ascoli Theorem it suffices to show that (f_ν) is uniformly bounded and equicontinuous.

We will first show that the sequence is uniformly bounded. Since P is a polynomial and therefore continuous on $\boldsymbol{D}$, $|P|$ assumes its finite maximum M_1 over $\boldsymbol{D}$ at some point $\tilde{x}$:

$$M_1 = |P(\tilde{x})| = \max\{|P(x)| \,|\, x \in \boldsymbol{D}\}. \qquad (4.35)$$

Since R is bounded, there is a constant $M_2 > 0$ such that

$$y \in R \;\Rightarrow\; |y| \leq M_2. \qquad (4.36)$$

After defining $\delta_\nu = f_\nu - P$, $\delta_\nu(x) \in R$ for all $x \in \boldsymbol{D}$. Thus for any $x \in \boldsymbol{D}$ and any $\nu \in \mathbb{N}$

$$|f_\nu(x)| \leq |P(x)| + |\delta_\nu(x)| < M_1 + M_2 < \infty. \qquad (4.37)$$

Hence, the sequence is indeed uniformly bounded. Moreover, since by definition the sequence is also Lipschitzian with uniform Lipschitz constant L that is independent is ν, it follows that it also equicontinuous.

Thus, the L-Taylor model T_L is a compact subset of the Banach space $\mathcal{C}^0(\boldsymbol{D}, \mathbb{R}^w)$. □

The main result of this section is that the self-inclusion given in Eqn. (4.27) in indeed sufficient to guarantee that the fixed point of $\mathcal{H}$ is contained in the Taylor model T. The proof of this assertion is based on the next theorem, which is an almost immediate consequence of the previous two lemmas.

Theorem 4.2. *Let $\boldsymbol{T}$ and $\boldsymbol{D}$ be interval domains containing the points t_0 and x_0 respectively. Let $L > 0$ be a Lipschitz constant for P and define the L-Taylor model*

$$T_L = (P, (t_0, x_0), \boldsymbol{T} \times \boldsymbol{D}, R). \tag{4.38}$$

If $\mathcal{H}(T_L) \subset T_L$, then the L-Taylor model T_L contains a fixed point of $\mathcal{H}$.

Proof. Since $\mathcal{H}$ is a continuous operator on $\mathcal{C}^0(\boldsymbol{T} \times \boldsymbol{D}, \mathbb{R}^v)$, the assertion follows from the last two lemmas and the Schauder Fixed Point Theorem. □

According to this theorem, a fixed point of $\mathcal{H}$ is contained in the L-Taylor model T_L. However, as indicated earlier, one of the fixed points is in fact equal to $x'(t,y) - x'_0(y)$ and is therefore of class $\mathcal{C}^{n+1}$. Moreover, according to Thm. 4.1, the n-th order Taylor expansion of the fixed point around (t_0, x_0) equals P. Thus, if $|\mathcal{H}| < 1$, the fixed point is also contained in the regular Taylor model

$$T = (P, (t_0, x_0), \boldsymbol{T} \times \boldsymbol{D}, R). \tag{4.39}$$

We also note that since the reference polynomial P is already a very good approximation of the mathematically correct fixed point. In practice the self-inclusion of the image can almost always be achieved by choosing sufficiently small domains $\boldsymbol{T}$ and $\boldsymbol{D}$ around the reference points t_0 and x_0.

4.2.3 Discussion of the Algorithm

After having established the mathematical foundations of the algorithm in the previous subsection, we will now comment on the individual steps of the basic method. In the following discussion we focus on how each stage can be performed automatically in a computer environment, with only limited need for manual user interventions.

1. In the integration of explicit ODEs, the initial derivative $x'_0 = x'(t_0)$ is uniquely determined by the systems initial conditions. In the context of implicit ODE on the other hand, the solution to an initial value problem is not guaranteed to be unique. Thus, the consistent initial condition x'_0 has to be obtained during a pre-analysis step. And since the consistent initial condition may not be unique, verified methods have to be used for an exhaustive global search. To simplify this, the user should be able to supply initial search regions for x'_0.

   In addition to the example given in Eqn. (4.4), the following example illustrates how the solutions of implicit ODE initial value problems are not necessarily unique.

   $$\left(x'\left(t\right)\right)^2 + \left(\sin\left(t\right)\right)^2 = 1 \text{ and } x(0) = 0 \tag{4.40}$$

   Obviously, $x_-(t) = -\sin(t)$ and $x_+(t) = +\sin(t)$ are two distinct smooth solutions of the problem with different initial derivatives -1 and $+1$.

   Since the following stages of the algorithm require an expansion of the consistent initial velocity in the transverse variables, the use of high-order Newton methods or the inversion schemes discussed in Sec. 3.5 are warranted at this point. This will compute both a Taylor model and a suitable DA vector for the following stages of the algorithm.

   As part of the first step, the regularity of the Jacobian matrix (4.20) should also be verified, and rigorous bounds on the sizes of the domains of regularity have to be established for t, x, and x'. Naturally, all subsequent steps then have to ensure, by possibly shrinking the domains for the independent variable t and the initial conditions, that all computed quantities stay within these upper bounds.

2. From a user's perspective, it is important that by combining a suitable user interface with a dynamically typed runtime environment like COSY Infinity, the substitution of the variables with antiderivatives can be performed automatically, and there is no need for the user to rewrite any of the equations by hand.

3. By shifting to coordinates that are relative to the consistent initial condition x_0', the solution space is restricted to the set $M = \{a \in {}_nD_v : \lambda(a) \geq 1\}$ of nilpotent DA vectors. In step 6, this allows the definition of a DA-contracting operator, and the application of the DA Fixed Point Theorem. Again, this coordinate shift can be performed automatically within the semi-algebraic DA framework of COSY Infinity.

4. As in the previous two steps, the semi-symbolic nature of the DA framework allows the linear part L_ζ to be extracted accurately and automatically, since the linearizations are computed automatically and are readily available in DA and Taylor model based methods. It should also be noted that the existence of the inverse over the domains of interest has already been established in the first step, since the regularity of L_ζ follows from the regularity of the Jacobian (4.20).

5. Frequently the solution of a system can be specified as the fixed point of a sufficiently smooth operator. Here we have the derivative of the solution of an implicit ODE initial value problem given as the fixed point of such an operator. Moreover, according to Thm. 4.1, the fixed point does indeed represent the desired solution.

6. According to Lem. 4.1, $\mathcal{H}$ is well defined and DA-contracting on the set M of nilpotent DA vectors. Hence the DA Fixed Point Theorem guarantees that the sequence (a_ν) of iterates converges in at most $n+1$ steps to the n-th order

solution polynomial.

$$a_{n+1} = P(t, y) =_n \zeta(t, y). \tag{4.41}$$

It should however be noted that within a computer environment the implementation and iteration of $\mathcal{H}$ finds only a floating point polynomial which is a fixed point of the floating point operator $\mathcal{H}$. While the coefficients of the polynomial might differ from the mathematically exact n-th order expansion by the machine epsilon, it is in fact sufficient to find a fixed point of $\mathcal{H}$ only to machine precision, since deviations from the exact result will be accounted for in the remainder bound of the Taylor models.

7. It has been shown that for explicit ODEs and the Picard operator $\mathcal{P}$ defined in Eqn. 2.44, inclusion is guaranteed if the solution Taylor model T satisfies $\mathcal{P}(T) \subset T$. Although $\mathcal{H}$ differs from $\mathcal{P}$, based on the proof sketched in [82], we have been able to establish a similar result in Thm. 4.2.

   While all previous steps are guaranteed to succeed whenever at least one consistent x_0' can be found for which the linear part is regular, practical applications of this step of the algorithm can fail if no suitable Taylor model can be constructed, although decreasing the size of the domains will generally lead to a successful inclusion. Details on suitable strategies for the construction of the Taylor model T can be found in [82, 24]. Lastly, it should be noted that this step of the algorithm requires a guaranteed computation of $\mathcal{H}$, using Taylor model enclosures for x_0' and L_ζ.

8. This final step computes an enclosure of the solution to the original problem from the computed Taylor model containing the derivative of the actual solution, and it relies on antiderivation being inclusion-preserving. To maintain full verification, the Taylor model enclosure of x_0' has to be added to the result of

the integration.

Similar to the example presented in (4.15), the new method also allows the direct integration of higher order implicit ODE initial value problems without explicit order reduction. While this approach does not reduce the dimensionality of the Taylor models if dependence on initial conditions is desired, it can however improve the sharpness of the final remainder bounds. Since the magnitude of the time domain $\boldsymbol{T}$ is usually smaller than one, Def. 2.8 shows that the remainder bounds of the actual solution will often be smaller than the ones of the computed highest derivative.

To illustrate how the algorithm can be adapted to higher order ODEs, consider the general second order implicit ODE initial value problem

$$G(t, x, x', x'') = 0, \quad x(t_0) = x_0, \quad x'(t_0) = x'_0. \tag{4.42}$$

If we assume that the Jacobian matrix $\partial G(t, u, v, w)/\partial w$ is non-singular in a suitable domain, this can be written as

$$\Phi(t, \xi) = G\left(x_0 + \int_{t_0}^{t} \left(x'_0 + \int_{t_0}^{\tau} \xi(\sigma) d\sigma\right) d\tau, x'_0 + \int_{t_0}^{t} \xi(\tau) d\tau, \xi, t\right) = 0, \tag{4.43}$$

and the algorithm works with only minor adjustments. Similar arguments can be made for more general higher order ODEs. The performance of this approach will be illustrated in the next section with the direct integration of a second order system.

4.2.4 A Second Order Example

To illustrate how the new algorithm can be used for the direct integration of higher order problems and to demonstrate how the method works in practice, consider the implicit second order ODE initial value problem

$$e^{x''} + x'' + x = 0, \tag{4.44a}$$

$$x(0) = x_0 = 1, \tag{4.44b}$$

$$x'(0) = x'_0 = 0. \tag{4.44c}$$

While the demonstration in this section uses explicit algebraic transformations for illustrative purposes, it is important to note that the actual implementation uses the DA framework and does not rely on such explicit manipulations. We also mention that for purposes of illustration, this example does not expand the solution in the transverse variables.

1. Compute a consistent initial value for $x''_0 = x''(0)$ such that $e^{x''_0} + x''_0 + x_0 = 0$. A simple interval Newton method, with a starting value of 0, finds an enclosure of the unique solution $x''_0 = -1.278464542761074$ in just a few steps.

2. Rewrite the original ODE in a derivative-free form by substituting $\xi = x''$.
$$\Phi(\xi, t) = e^{\xi(t)} + \xi(t) + \left(x_0 + \int_0^t \left(x'_0 + \int_0^\tau \xi(\sigma)d\sigma\right) d\tau\right) = 0. \tag{4.45}$$

3. Define the new dependent variable ζ as the relative distance of ξ to its consistent initial value and substitute $\zeta = \xi - x''_0$ in Φ to obtain the new function Ψ given by
$$\Psi(\zeta, t) = \zeta + x''_0 + e^{x''_0} e^{\zeta} + 1 + \frac{x''_0}{2} t^2 + \int_0^t \int_0^\tau \zeta(\sigma) d\sigma d\tau = 0. \tag{4.46}$$

4. The linear part $L_\zeta(\zeta)$ of Ψ is $1 + e^{x''_0}$ where the 1 is the constant coefficient and $e^{x''_0}$ results from the linear part of the exponential function e^ζ.

5. With L_ζ from the previous step, the solution ζ is a fixed point of the DA-contracting operator $\mathcal{H}$ defined by
$$\mathcal{H}(\zeta) = \frac{1}{1 + e^{x''_0}} \left(e^{x''_0}\left(\zeta - e^\zeta\right) - x''_0 - 1 - \frac{x''_0}{2} t^2 - \int_0^t \int_0^\tau \zeta\left(\sigma\right) d\sigma d\tau\right). \tag{4.47}$$

6. Starting with an initial value of $\zeta^{(0)} = 0$, the n-th order expansion P of ζ is obtained in exactly n steps.

$$\zeta^{(k+1)} = \mathcal{H}\left(\zeta^{(k)}\right). \tag{4.48}$$

7. The result is verified by constructing a Taylor model T with the computed reference polynomial P such that $\mathcal{H}(T) \subset T$. With the Taylor model

$$T = \left(P, 0, [0, 0.5], \left[-10^{-14}, 10^{-14}\right]\right), \tag{4.49}$$

$$\mathcal{H}(T) = \left(P, 0, [0, 0.5], \left[-0.659807722 \cdot 10^{-14}, 0.659857319 \cdot 10^{-14}\right]\right). \tag{4.50}$$

Since P is a fixed point of $\mathcal{H}$, the inclusion $\mathcal{H}(T) \subset T$ can be checked by simply comparing the remainder bounds of T and $\mathcal{H}(T)$; the inclusion requirement is satisfied for the constructed T.

8. A Taylor model for x is obtained by using the antiderivation of Taylor models according to

$$x(t) = x_0 + \int_0^t \left(x_0' + \int_0^\tau \left(x_0'' + \zeta(\sigma)\right) d\sigma\right) d\tau. \tag{4.51}$$

The listing in Tab. 4.1 shows the resulting Taylor model of order 25 computed by COSY Infinity

This example demonstrates how the new method can be used for the verified integration of implicit ODE initial value problems to high accuracy. In this context it is important to note that the width of the final enclosure of the solution is in the order of 10^{-14} for a relatively large time step of $h = 0.5$.

4.3 Verified Integration of DAEs

In this section we revisit the question of how a given DAE initial value problem can be transformed into an equivalent system of implicit ODEs. Since no general purpose

```
         RDA VARIABLE:   NO=  25, NV=     1
             I  COEFFICIENT            ORDER
             1   1.000000000000000       0
             2  -.6392322713805370       2
             3  0.4166666666666668E-01   4
             4  -.1993921404777223E-02   6
             5  0.6314945441169959E-04   8
             6  0.2635524930464548E-05  10
             7  -.4411105791086625E-06  12
             8  -.1533094467519992E-07  14
             9  0.8104707776528831E-08  16
            10  -.3384116382961162E-09  18
            11  -.1389729003787960E-09  20
            12  0.1981078695604361E-10  22
            13  0.1549987273495670E-11  24
            ----------------------------------
 VAR    REFERENCE POINT        DOMAIN INTERVAL
   1  0.000000000000000     [0.00000, 0.50000]
             REMAINDER BOUND INTERVAL
   R     [-.2500253775762034E-014,0.2500000000000003E-014]
 **********************************************************
```

Table 4.1: Taylor model for the solution of the implicit second order ODE initial value problem given by Eqn. (4.44).

methods for the direct integration of DAEs exist, this transformation is in general the only available method of computing solutions to the initial value problems of differential algebraic equations.

We first summarize the main result of the Σ-method by J. D. Pryce [107, 108], which uses a structural analysis of DAEs to establish the existence of solutions. Additionally, the method finds an upper bound on the problem's differentiation index ν_d. Moreover, and more importantly, it also determines an explicit scheme for the transformation of the DAE system into a solvable system of implicit ODEs.

However, within the previously presented algorithm for the integration of ODEs, the regularity of L_ς already offers a sufficient criterion for the solvability of the derived ODEs. While the linear map L_ς will generally be singular, by repeatedly differentiating the individual equations of the DAE, we will eventually obtain a regular linear map L_ς. Additionally, once the consistent initial derivative is computed, the minimum number of differentiations gives the differentiation index ν_d of the DAE.

4.3.1 Structural Analysis

Here we summarize the main results of the structural analysis of DAEs developed by J. D. Pryce [105, 107, 108]. The signature method, or Σ-method, is used to decide whether a given DAE possesses a unique solution and to transform it into an equivalent system of implicit ODEs. The method considers the general DAE problem

$$\begin{aligned} f_1\left(t, x_1, \ldots, x_1^{(\xi_{11})}, \ldots, x_v, \ldots, x_v^{(\xi_{1v})}\right) &= 0 \\ &\vdots \\ f_v\left(t, x_1, \ldots, x_1^{(\xi_{v1})}, \ldots, x_v, \ldots, x_v^{(\xi_{vv})}\right) &= 0. \end{aligned} \tag{4.52}$$

We recall that for a given j, the i-th equation of this system does not necessarily depend on x_j, the derivatives $x_j^{(\eta)}$ for $\eta < \xi_{ij}$, or even any of its derivatives. However, if there is a dependence on at least one of the derivatives, including the zeroth order derivative $x_j^{(0)} = x_j$, we denote the order of that highest derivative by ξ_{ij}. We formally define the $v \times v$ matrix $\Sigma = (\sigma_{ij})$ by

$$\sigma_{ij} = \begin{cases} -\infty & \text{if the } j\text{-th variable doesn't occur in } f_i \\ \xi_{ij} & \text{otherwise.} \end{cases} \tag{4.53}$$

The matrix Σ is called the *signature matrix* of the problem (4.52) and it forms the foundation of the structural analysis.

We denote the set of permutations of length v by $\mathcal{P}_v$ and consider the assignment problem [9] of finding a maximal transversal $T \in P_v$ of the signature matrix Σ.

$$\text{Maximize } \|T\| = \textstyle\sum_{i=1}^{v} \sigma_{i,T(i)} \text{ with } \sigma_{i,T(i)} \geq 0. \tag{4.54}$$

Generally, an assignment problem is the task of matching v workers with v jobs. The entries of the assignment matrix Σ measure the effectiveness of the person i for the job j. An effectiveness of $-\infty$ indicates the inability of performing a particular task. By requiring the entries of the transversal T to be finite, we effectively limit the problem to the sparsity pattern

$$S = \{(i,j)\,|\sigma_{ij} > -\infty\}\,. \tag{4.55}$$

It should be noted that this analysis of the sparsity structure can be ill-posed and the assignment problem does not have any feasible solutions. More importantly, it has been shown by Pantelides that the assignment problem might be ill-posed even for solvable DAE problems. While this is one of the largest drawbacks of the structural analysis, the method works for many important DAE problems, including DAEs of index 0 and DAEs in Hessenberg form [108].

If a maximal transversal exists, we can consider the linear programming problem in the variables $(c_i), (d_j) \in \mathbb{Z}^v$ defined by

$$\text{Minimize } \bar{z} = \sum_{j=1}^{v} d_j - \sum_{i=1}^{v} c_i, \tag{4.56}$$

subject to the restrictions

$$d_j - c_i \geq \sigma_{ij}\ \forall\,(i,j) \in S,\ \ c_i \geq 0\ \forall\, i = 1, \ldots, v. \tag{4.57}$$

While this problem does not have a unique solution, there is a uniquely determined smallest solution [108], and the smallest $(c_i), (d_j) \in \mathbb{Z}^v$ are called the *offsets* of the problem.

Once the offsets have been determined, we form the $v \times v$ *system Jacobian* matrix $J = (J_{ij})$ with entries

$$J_{ij} = \frac{\partial f_i}{\partial\left(x_j^{(d_j - c_i)}\right)}, \tag{4.58}$$

where $J_{ij} = 0$ if the derivative is not present or if $d_j < c_i$. We will later see that in the framework of structural analysis the system Jacobian plays the role of the Jacobian matrix (4.20) in conventional index analysis of DAEs.

Next we consider the collection of equations obtained by taking derivatives of the f_i with respect to the independent variable t:

$$\begin{aligned} f_1^{(0)} = f_1^{(1)} = \ldots = f_1^{(c_1)} = 0, \\ \vdots \\ f_v^{(0)} = f_v^{(1)} = \ldots = f_v^{(c_v)} = 0. \end{aligned} \tag{4.59}$$

By definition of σ_{ij} and Eqn. (4.57), the derivatives of the x_j that appear in the equations (4.59) are all in the set

$$\left\{x_1^{(0)}, x_1^{(1)}, \ldots, x_1^{(d_1)}, \ldots, x_v^{(0)}, x_v^{(1)}, \ldots, x_1^{(d_v)}\right\}. \tag{4.60}$$

If we write the set (4.60) as a vector X and collect all the equations (4.59) in the function F, we can write the derived system as

$$F\left(t, X\right) = 0, \tag{4.61}$$

where F and X have $M = v + \sum c_i$ and $N = v + \sum d_j$ components, respectively.

Within the framework of structural analysis of DAEs, a consistent point of (4.52) is defined as a scalar t_0 and a set of scalars ξ_{jl} indexed over a finite set $\mathcal{I}$ of indices (j, l) such that there is a *unique* solution of (4.52) near $t = t_0$ such that $x_j^{(l)}(t_0) = \xi_{jl}$. However, the set $\mathcal{I}$ is not required to be a minimal index set, and it has been shown that it generally will not be minimal [116]. The concept of consistent points allows a simple success check of the structural analysis. If we view (4.61) as M algebraic equations in N variables and assume that the system has a solution (t^*, X^*), then if the system Jacobian (4.58) is regular at the point (t^*, X^*), it is a consistent point and the system (4.52) has a unique solution near $t = t^*$. Moreover, in a neighborhood of (t^*, X^*) the system has D degrees of freedom, where D is the common optimal value of (4.54) and (4.57),

$$D = \|T\| = \bar{z} = \sum_{j=1}^{v} d_j - \sum_{i=1}^{v} c_i. \tag{4.62}$$

The structural analysis also provides an upper bound for the differentiation index ν_d of the system.

$$\nu_d \leq \max c_i + \begin{cases} 0 & \text{if all } d_j > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.63}$$

While it has been suggested that the index ν_d might equal the given expression, examples in [116] show that (4.63) provides only be an upper bound.

To utilize structural analysis for the numerical integration of the DAE (4.52) we use the subset of equations belonging to the highest derivatives in (4.59) and solve it as an implicit ODE initial value problem. Thus, we solve the v-dimensional ODE problem

$$f_1^{(c_1)} = \ldots = f_v^{(c_v)} = 0 \tag{4.64}$$

to find the unique solution of the original DAE problem. Moreover, all the intermediate equations

$$\begin{aligned} f_1^{(0)} = \ldots = f_1^{(c_1-1)} &= 0 \\ &\vdots \\ f_v^{(0)} = \ldots = f_v^{(c_v-1)} &= 0 \end{aligned} \tag{4.65}$$

express the obvious and hidden constraints. These equations can in principle be used to verify initial conditions and make a posteriori adjustments to the computed final coordinates.

Finally, it should be noted that the structural analysis is not infallible, since it relies on the presence of a substantial sparsity pattern. To illustrate this, consider some DAE system $F = 0$ for which the structural analysis succeeds and the c_i are not all zero. If we then premultiply the system by a random non-singular matrix, the resulting DAE is equivalent to the original one. However, the signature matrix of the new system will generally be constant in each column, since the system is unlikely to exhibit any significant sparsity patterns. Thus, all the offsets will be $c_i = 0$ and the system Jacobian is identically singular.

4.3.2 Verified Index Analysis

The applicability of the structural analysis summarized in the previous section is often limited by its inherent limitation; namely, the Σ-method fails if the system has no obvious sparsity pattern. Moreover, it has been shown that index one problems can have arbitrary large structural and Taylor indices, rendering the method unpractical in these cases [116]. More formally, the fact that the index set $\mathcal{I}$ is not necessarily minimal, may lead to arbitrarily large indices and unnecessary computational overhead.

Within the DA framework on the other hand, the regularity of L_ζ in the neighborhood of a consistent point is a sufficient criterion for the existence and uniqueness of solutions. Moreover, since the regularity of L_ζ is equivalent to the regularity of the Jacobian (4.20), this approach allows a rigorous determination of the differentiation index ν_d. In practical terms, this approach is simplified by the availability of the differentiation in the DA framework of COSY Infinity. After determination of a consistent initial velocity, a conventional DA analysis can easily determine the differentiation index of the system, provided it is in fact smaller than the order of the DA vectors. High order Taylor model methods can then be used to rigorously guarantee regularity of the resulting system Jacobian and determine the existence of a consistent point for the initial conditions.

Clearly this approach benefits from the availability of high order derivatives in the DA framework. However, to fully automate this method and avoid the need for hand-coded derivatives, it requires the additional availability of the derivative operation on Taylor models. Its utilization can completely eliminate the need for manual user intervention from the whole algorithm, which has repeatedly been recognized as one of the most important aspects of successful user interface design for verified integration schemes [135, 34, 65].

4.4 Examples and Applications

To further illustrate the concept of DAEs and to show the wide range of applications in which they arise naturally we discuss two important representatives of general DAE problems in this section. Moreover, as an interesting application of DAEs, we will demonstrate how prescribed path control problems can be formulated as DAE problems. In light of the Taylor model approach the method has distinct advantages over other existing methods for verified feedforward control problems; namely, a reduction in computational complexity and a broader range of applications, since it is not limited to explicit first order systems.

4.4.1 Kirchhoff's Law and Electric Circuits

Differential algebraic equations are a natural way of modeling systems that obey intrinsic conservation laws. Frequently these systems can be written in either Hessenberg Index-1 or Index-2 form. While we will encounter an index two problem in the next section, here we illustrate the concept of DAEs with an index one example from of electrical engineering. Consider the electric circuit shown in Fig. 4.1, which combines resistors, capacitors, and transistors.

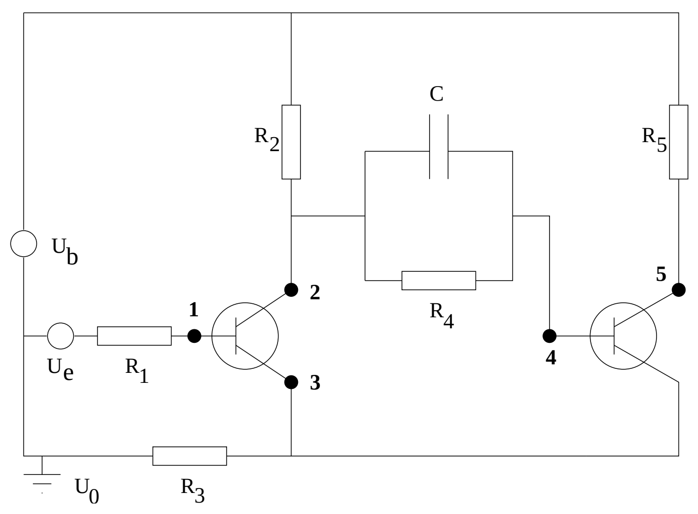


Figure 4.1: Illustration of an electric circuit described by the index-1 DAE (4.69). U_b is the operating voltage and U_e is the driving voltage of the circuit.

Ohm's Law describes the simple linear relationship between the voltage U and the current I at a resistor of resistance R by

$$U = RI. \tag{4.66}$$

At a transistor on the other hand, the voltage and the currents at base, emitter, and collector are connected by nonlinear relationships of the form

$$I_E = f(U) = \beta\left(e^{U/U_F} - 1\right) \tag{4.67a}$$

$$I_C = -\alpha I_E \tag{4.67b}$$

$$I_B = (\alpha - 1)\, I_E, \tag{4.67c}$$

where U_F, α, and β are characteristic quantities of the transistor. Across a capacitor C the connection between voltage and current is given by a simple first order differential equation

$$I = CU'. \tag{4.68}$$

If we apply Kirchhoff's Law, which formalizes the conservation of current, to the five points marked in Fig. 4.1, we get the following set of equations:

$$\begin{aligned}
0 &= \frac{U_1 - U_e}{R_1} - (\alpha - 1)\, f\,(U_1 - U_3) && (4.69\text{a}) \\
C\,(U_2' - U_4') &= \frac{U_b - U_2}{R_2} + \frac{U_4 - U_2}{R_4} - \alpha f\,(U_1 - U_3) && (4.69\text{b}) \\
0 &= \frac{U_3 - U_0}{R_3} - f\,(U_1 - U_3) - f\,(U_4 - U_3) && (4.69\text{c}) \\
C\,(U_4' - U_2') &= (\alpha - 1)\, f\,(U_4 - U_3) - \frac{U_4 - U_2}{R_4} && (4.69\text{d}) \\
0 &= \frac{U_4 - U_b}{R_5} + \alpha f\,(U_4 - U_3)\,. && (4.69\text{e})
\end{aligned}$$

Thus, the response of the system to the driving voltage $U_e(t)$ is determined by a differential algebraic equation with index one. The system consists of four algebraic constraints. Since the two differential equations are linearly dependent, they amount to one differential and one algebraic equation. It is easy to see that the problem has only one degree of freedom, since only the difference $U_2(0) - U_4(0)$ can be chosen freely.

This circuit is a typical example for a large class of engineering problems that have conservation laws at their core. Most, if not all of these dynamical systems are naturally described by differential algebraic equations, providing an almost infinite number of interesting and important DAE problems, many of which can and should be solved with verified methods.

4.4.2 Constrained Mechanical Systems

Constrained mechanical systems can often be written as differential algebraic equations and are typically of the form

$$M(x) \cdot x'' = f(t, x, x') + G(x)\lambda \tag{4.70a}$$

$$\Phi(x) = 0 \tag{4.70b}$$

where $x \in \mathbb{R}^v$ is the state vector, $\lambda \in \mathbb{R}^w$ is the Lagrange multiplier, and $\Phi' = \partial\Phi/\partial q = G^T$. We will generally assume that M is positive-definite symmetric; so its diagonal is non-zero. Moreover we will also assume that Φ' has full row rank, implying that $v \geq w$.

Extensive theories have been developed that give *cookbook recipes* for solving these problems by using the constraint conditions (4.70b) to introduce new variables that reduce the problem's dimensionality [49, 61]. However, these schemes usually rely, to a certain degree, on the user's intuition and often require substantial arithmetic to reorganize and simplify the resulting ODEs into explicit first order systems. Moreover, while the use of the constraints (4.70b) does lead to simplified ODEs, it generally does not reveal the hidden constraints of the system.

Within the context of differential algebraic equations, the regular and the hidden constraints are easily accessible. Moreover, the availability of automated integration schemes for DAE initial value problems removes the need for intuition and physical insight into the problems and allows the automated integration of these important systems without user intervention.

Example: Double Pendulum

As a prototypical example for constrained mechanical systems, consider a planar pair of connected pendulums in a frictionless environment. Assume that the pendulums

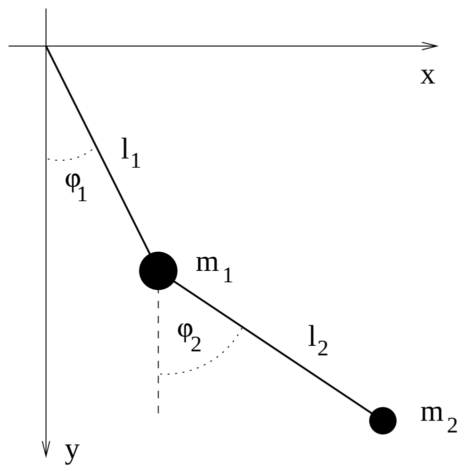


Figure 4.2: Illustration of a double pendulum with masses m_1 and m_2 connected by massless rods of lengths l_1 and l_2, respectively.

are massless and inextensible, with point masses on the ends, as illustrated in Fig. 4.2.

If we denote the tensions in the strings by λ_1 and λ_2, they take the roles of the Lagrange multipliers in this problem and the equations of motion, expressed in the Cartesian coordinates x_1, y_1, x_2, y_2, are given by

$$\begin{aligned}
m_1 x_1'' + \lambda_1 x_1/l_1 - \lambda_2(x_2 - x_1)/l_2 &= 0 \\
m_2 y_1'' + \lambda_1 y_1/l_1 - \lambda_2(y_2 - y_1)/l_2 - m_1 g &= 0 \\
m_2 x_2'' + \lambda_2(x_2 - x_1)/l_2 &= 0 \\
m_2 y_2'' + \lambda_2(y_2 - y_1)/l_2 - m_2 g &= 0 \\
x_1^2 + y_1^2 - l_1^2 &= 0 \\
(x_2 - x_1)^2 + (y_2 - y_1)^2 - l_2^2 &= 0.
\end{aligned} \tag{4.71}$$

While it is common practice in classical mechanics to reduce this problem to a two dimensional ODE in the variables ϕ_1 and ϕ_2, here the focus is on treating this problem as a differential algebraic initial value problem in the six variables x_1, y_1, x_2, y_2 and λ_1, λ_2.

The Σ matrix of the system (4.71), with the entries forming a maximal transversal in bold face, is given by

$$\Sigma = \begin{pmatrix} 2 & -1 & 0 & -1 & \mathbf{0} & 0 \\ -1 & \mathbf{2} & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & -1 & \mathbf{0} \\ -1 & 0 & -1 & \mathbf{2} & -1 & 0 \\ \mathbf{0} & 0 & -1 & -1 & -1 & -1 \\ 0 & 0 & \mathbf{0} & 0 & -1 & -1 \end{pmatrix}. \tag{4.72}$$

Further analysis gives the offsets $c = (0,0,0,0,2,2)$ and $d = (2,2,2,2,0,0)$ and reveals that the system has four degrees of freedom, a structural index of three and a differentiation index ν_d of two.

Based on this analysis, the implicit ODE problem that can be derived from the DAE (4.71) is given by

$$\begin{aligned} m_1 x_1'' + \lambda_1 x_1/l_1 - \lambda_2(x_2 - x_1)/l_2 &= 0 \\ m_2 y_1'' + \lambda_1 y_1/l_1 - \lambda_2(y_2 - y_1)/l_2 - m_1 g &= 0 \\ m_2 x_2'' + \lambda_2(x_2 - x_1)/l_2 &= 0 \\ m_2 y_2'' + \lambda_2(y_2 - y_1)/l_2 - m_2 g &= 0 \\ 2\left({x_1'}^2 + x_1 x_1'' + {y_1'}^2 + y_1 y_1''\right) &= 0 \\ 2\left((x_2' - x_1')^2 + (y_2' - y_1')^2 + (x_2 - x_1)(x_2'' - x_1'') + (y_2 - y_1)(y_2'' - y_1'')\right) &= 0. \end{aligned} \tag{4.73}$$

Additionally, the complete set of obvious and hidden constraint conditions for this system is given by

$$\begin{aligned} x_1^2 + y_1^2 - l_1^2 &= 0 \\ (x_2 - x_1)^2 + (y_2 - y_1)^2 - l_2^2 &= 0 \\ 2(x_1 x_1' + y y_1') &= 0 \\ 2((x_2 - x_1)(x_2' - x_1') + (y_2 - y_1)(y_2' - y_1')) &= 0 \end{aligned} \tag{4.74}$$

The first two constraints express that the lengths l_1 and l_2 are constant, and the

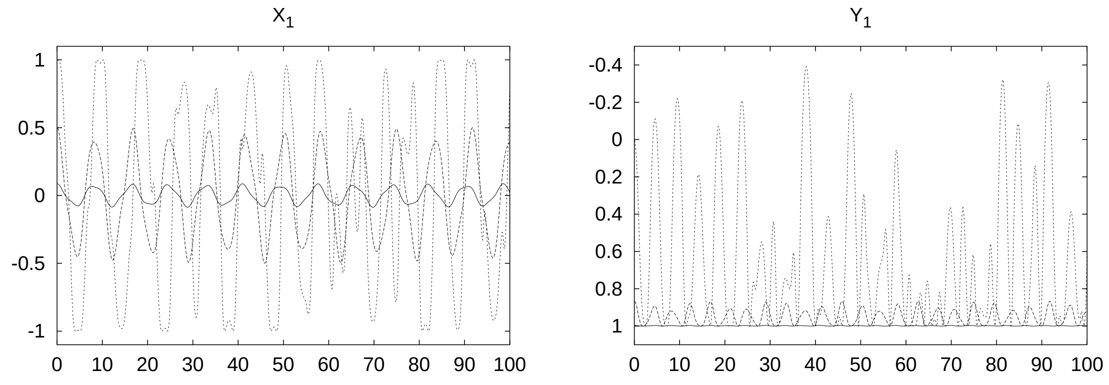


Figure 4.3: Coordinates x_1 and y_1 of the double pendulum for $\varphi_1(0) = \varphi_2(0) = 5°, 30°, 90°$, $x_1'(0) = x_2'(0) = y_1'(0) = y_2'(0) = 0$ and $0 \leq t \leq 100$ (integrated in order 6, $h = 0.1$, $g = 1$, $l_1 = l_2 = 1$, $m_1 = m_2 = 1$).

hidden constraints reveal that the velocities of the masses are tangent to the circles on which the masses move.

To illustrate the motion of the double pendulum, Figs. 4.3 to 4.5 show results obtained from various integrations of the double pendulum with different initial values. All plots highlight the oscillatory motion of the system with an energy transfer between the two masses. Fig. 4.3 shows the coordinates of the upper mass m_1 for various initial energies. While the system follows an almost periodic path for small energies, it shows signs of chaotic motion for high energies.

The phase space projections of the second mass m_2 shown in Fig. 4.4 indicate that the system moves on almost periodic orbits on its oscillating orbit. Finally, Fig. 4.5 shows the time evolution of the coordinates x_1 and y_1 for the case where the upper mass is 100 times heavier than the lower one. In this case, we see slow oscillations superimposed with the small and fast oscillations of the lower mass. Moreover, the evolution of y_1 indicates slow but noticeable energy transfers between the two masses.

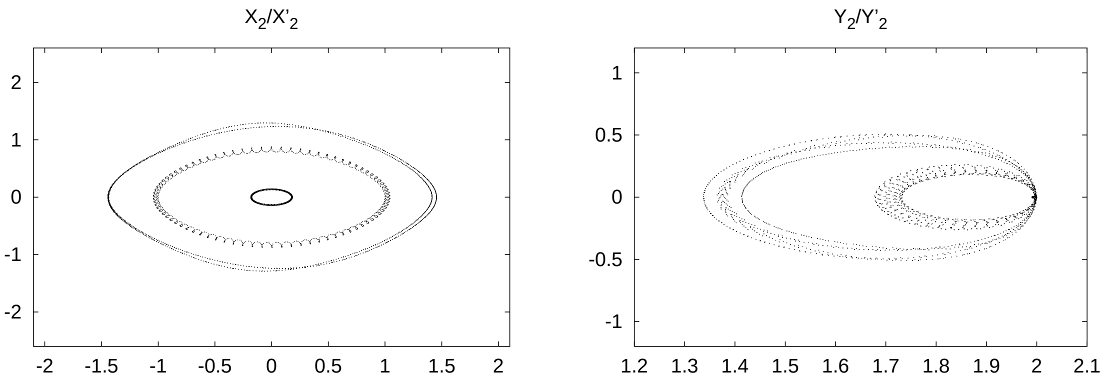


Figure 4.4: x_2- and y_2-phase space projections for the double pendulum with for $\varphi_1(0) = \varphi_2(0) = 5°, 30°, 45°$, $x_1'(0) = x_2'(0) = y_1'(0) = y_2'(0) = 0$ and $0 \le t \le 100$ (integrated in order 6, $h = 0.1$, $g = 1$, $l_1 = l_2 = 1$, $m_1 = m_2 = 1$).

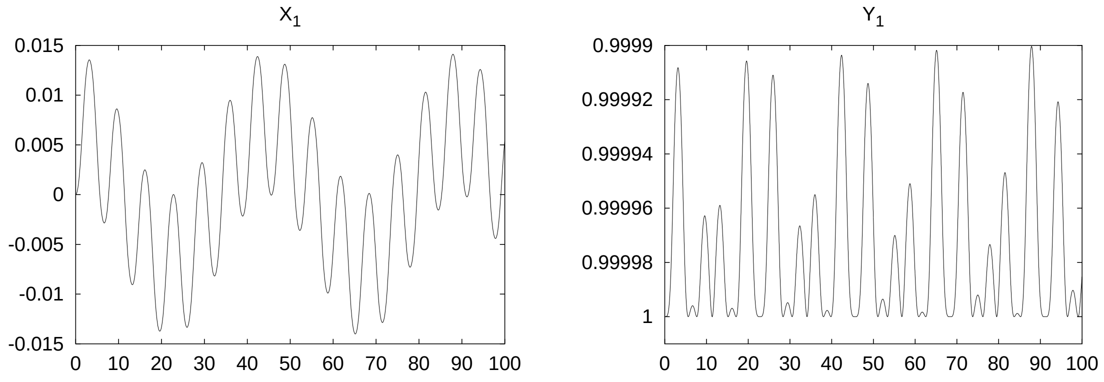


Figure 4.5: Coordinates x_1 and y_1 of the double pendulum for $\varphi_1(0) = 0$, $\varphi_2(0) = 45°$, $x_1'(0) = x_2'(0) = y_1'(0) = y_2'(0) = 0$ and $0 \le t \le 100$ (integrated in order 6, $h = 0.1$, $g = 1$, $l_1 = l_2 = 1$, $m_1 = 100$, $m_2 = 1$).

4.4.3 Optimal Prescribed Path Control

Prescribed path control problems for dynamical systems are traditionally solved by using feedback control methods. Feedback control is based on the idea that we can let the system operate with a pre-determined control function λ for a small time interval h and adjust the control function at the end of the time interval, depending on the deviation of the system from its prescribed path. While feedback control is usually easy to implement, its applicability is often limited to non-stiff problems that vary only slowly over time.

However, it is also possible to handle prescribed path problems in *feedforward mode*, where the appropriate control function is determined according to the desired future behavior of the system. Feedforward control tends to be more accurate in following the prescribed path with larger time steps h. Thus, feedforward control can often result in more accurate modeling with less computational cost. If we denote the independent variable by $t \in [t_i, t_{i+1}]$, the state of the system by $x(t)$ and the control input by $\lambda(t)$, the general prescribed path problem can be written as

$$F(t, x, x', \lambda) = 0 \;\; \forall\, t \in [t_i, t_{i+1}]. \tag{4.75}$$

Using the Taylor model approach for the integration of implicit ODEs, problems of this form can be solved for the system's state as a function of both the independent variable t and the control function λ. Thus, we can obtain a Taylor model with an explicit dependence on the control function λ,

$$x(t, \lambda). \tag{4.76}$$

Then, given a prescribed path $\psi(t)$, we define a new function, $\Delta = \Delta(t, \lambda)$, as the difference of the solution $x(t, \lambda)$ and the prescribed path $\psi(t)$.

$$\Delta(t, \lambda) = x(t, \lambda) - \psi(t). \tag{4.77}$$

The function Δ can be viewed as an implicit description of the control function $\lambda(t)$ such that the system follows the prescribed path. We may assume that $x(t_i) = \psi(t_i)$ and that we have an initial control $\lambda(t_i)$ such that

$$\left(\frac{\partial \Delta(t,\lambda)}{\partial \lambda}\right)(t_i, \lambda(t_i))\,. \tag{4.78}$$

is regular. Then, the Implicit Function Theorem guarantees the existence of a function $\hat{\lambda}(t)$ such that

$$\Delta(t, \hat{\lambda}) = 0 \tag{4.79}$$

in a neighborhood of t_i. Since the partial inversion discussed in Sec. 3.5 allows the determination of a Taylor model for the optimal control function $\hat{\lambda}$ from this expression, at this point the problem of verified prescribed path control can be considered solved. However, the potentially large number of additional variables required in solving the implicit equation (4.79) often renders this method unfeasible. With the availability of general DAE integration tools on the other hand, we can view (4.75) as a DAE problem and use methods for the verified integration of high order DAE initial value problems to compute appropriate control functions for the system (4.75).

In comparison to traditional methods of prescribed path control, the virtue of this method lies in the fact it requires only a single ODE integration for a rather long time step. Existing methods for prescribed path control optimize the control function through a series of repeated integrations or use extremely short time steps with the assumption that the control function stays constant over a single integration interval. The DAE approach on the other hand, allows the determination of a smooth and explicit expression for the control function to very high order. Moreover, by utilizing Taylor models at each step of the algorithm, the new method even allows the computation of rigorous error bounds on the computed control functions, providing the operators of the system with guaranteed and trustworthy results.

Example: Two-Link Robot Motion

To illustrate the formulation of prescribed path control problems as DAE initial value problems, the following system describes a slightly simplified version of the equations for the prescribed path control of a two-link robot arm [107]. With sufficiently smooth and highly non-linear functions f_1, f_2, and f_3, the system has a differentiation index of 5 and is given by

$$\begin{aligned}
x_1'' &= f_1(t, x_2, x_3, x_1', x_3', u_1, u_2) && \text{(4.80a)}\\
x_2'' &= f_2(t, x_2, x_3, x_1', x_3', u_1, u_2) && \text{(4.80b)}\\
x_3'' &= f_3(t, x_2, x_3, x_1', x_3', u_1, u_2) && \text{(4.80c)}\\
0 &= \cos(x_1) + \cos(x_1 + x_3) - \cos(e^t - 1) - \cos(t - 1) && \text{(4.80d)}\\
0 &= \sin(x_1) + \sin(x_1 + x_3) - \sin(1 - e^t) - \sin(1 - t). && \text{(4.80e)}
\end{aligned}$$

The prescribed path is expressed in the constraint conditions (4.80d) and (4.80e) and requires valid solutions x_1 and x_3 to be of the form

$$x_1(t) = 1 - e^t \tag{4.81a}$$

$$x_3(t) = e^t - t \tag{4.81b}$$

4.5 Summary

We have developed a method for the verified integration of general ODE initial value problems. It is based on a combination of high order Taylor models and the antiderivation, and it allows the automatic computation of tight solution enclosures for a large class of solvable initial value problems. The new approach avoids the wrapping effect to very high orders by expanding the solution not only in the independent variable t, but also in the transverse variables. Thus, the method computes accurate enclosures of the *flow* of ODE systems. Additionally, by using the method, or alternatively a more traditional structural analysis, for the index-analysis of differential algebraic equations, it can be extended to the integration of the large and important class of DAE initial value problems.

Finally, we point out that although we deliberately restricted the discussion to DAE problems with constant indices, many interesting practical problems have in fact non-constant index: valves in chemical processes may open and close, surfaces in mechanical devices may touch. However, a change in index is usually also associated with a non-smooth change in the equations, which is something the Taylor model approach cannot handle well. But the method can still be used in these cases by dynamically changing the equations *between* time steps and using the Taylor model describing the final coordinates at the end of the previous step as the initial condition for the next step.

Chapter 5

Applications of High Order Taylor Model Methods

In this chapter we present applications in which the availability of Taylor model methods allows the use of computers to obtain rigorous results. These applications require the high order Taylor model approach for two reasons: standard floating point methods fail to provide the required mathematical rigor, and conventional interval techniques fail to adequately address the problems of wrapping and dependency. Since the systems discussed in this chapter are not necessarily available for direct experiments, the Taylor model approach is the only method to study them rigorously.

First we study the verified integration of near-earth asteroid orbits. This problem has long been seen as a significant challenge for interval techniques, since it involves large initial sets, long integration intervals, and highly non-linear equations of motion, resulting in significant wrapping and dependency problems. However, we will show that the Taylor model approach can overcome all of these obstacles and return practical results. Unlike in the case of conventional floating point methods, the computed results are guaranteed enclosures of future positions of the asteroids, accounting for the effects of uncertain initial conditions, measured data, and mathematical truncation.

In a second application we use high order Taylor models to prove the existence of generating functions for Hamiltonian systems over extended domains. The approach combines the mathematical rigor of the invertibility tests discussed in Sec. 3.1 with Taylor model based integration of initial value problems. This novel combination provides the rigorous justification for the method of symplectic tracking of Hamiltonian systems, which has important applications in the study and the design of particle accelerators.

5.1 Solar System Dynamics and Asteroid Orbits

Spurred by recent media attention, the possibility and implications of asteroid impacts on Earth have recently come to the public's attention. Indeed, the potentially devastating consequences of such events justify research in preventing them, including possibly by altering the orbits of near-earth asteroids on collision courses. But protective measures cannot be taken unless near-earth asteroids are detected and their orbits are predicted. Moreover, for an increased response time it is essential to detect hazardous asteroids as early as possible and perform long-term integration of orbits from known observational data. Since the results of these predictions will potentially justify substantial preventive efforts, it is of utmost importance to actually get them *right*. Thus, the integration of asteroid orbits from initial measurements is an obvious application of verified ordinary differential equation (ODE) integration methods. Due to the computationally challenging models and the large uncertainties, however, so far all such computations have been performed using conventional high order floating point integration schemes with subsequent non-verified error analysis [73].

The integration of the motion of asteroids in the solar system poses several challenges to verified integration methods, mostly due to the large uncertainties in the initial conditions, measured physical constants (e.g. the gravitational constant is one the least accurately known constants [62]), and strong non-linearity in the mathematical models. Therefore, verified tools have not found widespread application for these problems. However, due to the potential seriousness of the outcome of the integration, they are important applications and test cases of verified methods [90, 135].

The major problem of integrating solar system dynamics lies in the relatively large uncertainties in the positions and velocities of the objects involved. This problem particularly hurts conventional verified integration schemes that are prone to the wrapping effect discussed in Sec. 2.2.2. It leads to a substantial overestimation on the range of final positions and velocities, and this overestimation often results in situations that do not permit reasonable conclusions, since the bounds are overly pessimistic. The other main obstacle to verified integration of solar system dynamics lies in the strong non-linearity of the equations of motion (5.3). Together with the uncertainties in the positions and velocities, this lets conventional verified integration suffer from cancellation problems and makes it almost impossible to obtain usable long-term integration results.

We will demonstrate, however, that the use of Taylor model based ODE integration methods [83, 85] can successfully alleviate these problems. In fact, the Taylor model approach can indeed be used to perform verified long-term integration of asteroid orbits in the solar system. We will demonstrate the performance of the method with several examples obtained from orbit integrations of the asteroid 1997 XF11, which is predicted to have several nearby passages of Earth between the years 2000 and 2040.

5.1.1 Physical Background

Before we discuss any details of rigorous asteroid integrations, in this section we summarize the physical principles that govern the motion of planets and asteroids in the solar system. We give an overview of the solar system and present an accurate formulation of the equations of motion for a point mass in it. Some of the notation introduced deviate from the standard conventions used in the International System of Units (SI) and will be used throughout the text.

Physical Units

The basic physical quantities relevant for the mathematical description of planetary motion are *length*, *time*, and *mass*. They are usually expressed in terms of the SI units Meter, Second, and Kilogram.

Quantity	*SI Unit*	*Symbol*
Length	Meter	m
Time	Second	s
Mass	Kilogram	kg

Table 5.1: SI units for length, time, and mass.

For practical calculations it is sometimes convenient to use different units than the ones listed in Tab. 5.1, mostly to avoid excessively large or small values of physical quantities. The following list gives a short description of the commonly used astronomical units of time, mass, and length:

- The astronomical unit of time is the time interval of one Earth-day; an interval of 36525 days is called a Julian century.
- The astronomical unit of mass is the mass of the Sun.
- The astronomical unit of length (AU) is the radius of a circular orbit in which a body of negligible mass, and free of perturbations, would revolve around the Sun in 365.25692 days, and its value is almost equal to the mean distance between the Earth and the Sun [125].

The rules required for a conversion of these units into the SI system are given in Tab. 5.2. Several other physical quantities, namely the *velocity*, the *acceleration*, and the *force*), are expressed in derived units; Tab. 5.3 lists them in terms of the fundamental quantities length, mass, and time.

Quantity	*Unit*	*Symbols*	*Conversion*
Time	Day	d	1 d = 86400 s
	Julian Year	J	1 J = 365.25 d
Mass	Sun Mass	M	1 M = 1.9891×10^{30} kg
Length	Astronomical Unit	AU	1 AU = 1.496×10^{11} m

Table 5.2: Astronomical units and their conversions to SI units.

Quantity	*Unit*
Velocity	Length/Time
Acceleration	Length/Time^2
Force	Length $\times$ Mass/Time^2

Table 5.3: Units of the derived quantities velocity, acceleration, and force.

Astronomical calculations also make frequent use of angles, which are commonly measured in degrees (360 deg is equivalent to a full circle) or radians (2π rad $\equiv$ 360 deg). Finally, practical computations often require the use of ad hoc units to satisfy requirements of the algorithms used, and these units will be introduced as needed.

Gravitation

Gravitation is an attractive force between masses, and for the purpose of gravitational interactions in the solar system, all masses can be treated as point masses; i.e., as if their mass was concentrated in the center of mass. Classically, the force inflicted by a point mass m_2 on a mass m_1 acts along the line connecting the two points and is given by

$$\vec{F}_{12} = G\,\frac{m_1 m_2}{\|\vec{r}_{12}\|^3}\,\vec{r}_{12}, \tag{5.1}$$

where G denotes the *gravitational constant* and $\vec{r}_{12}$ is the vector connecting m_1 with m_2. Eqn. (5.1) is due to Newton, who derived it from observational data of planetary motion and Kepler's Laws. Since $\vec{F}_{12} = -\vec{F}_{21}$, gravitation satisfies Newton's Third Law: *actio = reactio*.

Predicting the motion of two point masses m_1 and m_2 under their mutual Newtonian gravitational influence, neglecting all other forces and interactions, is known as the *two-body problem* and has been studied extensively. It can be shown that the barycenter of the two masses moves with constant velocity and that the two masses move on one of the *conic sections* circles, ellipses, parabolas, or hyperbolas with the common barycenter in the focus [49, 61]. Consequently, in a barycentric coordinate system, the two masses move on planar conic sections around the origin. Details of the motion on conic sections will be discussed in Sec. 5.1.2.

Within the framework of classical celestial mechanics, the superposition of the gravitational force is linear. Hence, the total force $\vec{F}_T$ on a mass M induced by N other masses m_i is merely the vector sum of the individual forces $\vec{F}_i$:

$$\vec{F}_T = \sum_{i=1}^{N} \vec{F}_i = \sum_{i=1}^{N} G\,\frac{M m_i}{\|r_i\|^3}\,\vec{r}_i \tag{5.2}$$

A more general and more complete model for the force on a point mass under the influence of other masses will be discussed in Sec. 5.1.1.

The Solar System

The solar system is the dynamical system consisting of the objects that are confined to orbits around the Sun. The largest members of the solar system are the Sun, the nine major planets, some of which are orbited by moons, and a large number of asteroids and comets.

Fig. 5.1 shows the orbits of the four innermost planets: Mercury, Venus, Earth, and Mars. As shown in Tab. A.1, the orbital ellipse of Mercury has a noticeable eccentricity. Fig. 5.2 shows the orbits of all nine major planets; the x/y-plane coincides

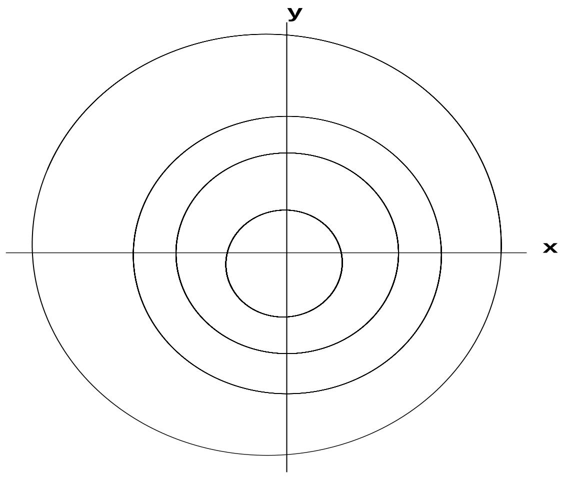


Figure 5.1: Orbits of the inner planets Mercury, Venus, Earth, and Mars around the Sun.

with the ecliptic; i.e., the plane of Earth's orbit, of the year 2000. The up-to-scale inclusion of the orbits of the inner planets illustrates the size of the solar system: Pluto's perihelion is almost 50 astronomical units away from the Sun, and it takes the light from the Sun almost seven hours to reach Pluto.

The Sun is by far the heaviest and largest object in the solar system, as its mass of approximately 1.9891×10^{30} kg exceeds the masses of the other planets by at least three orders of magnitude. As an important consequence of this dominance, the maximal distance between the solar system barycenter and the Sun's center of mass is only 1.5×10^{9} m, or about twice the radius of the Sun. For most practical purposes it is therefore sufficient to assume that all planets orbit the fixed Sun. Tab. 5.4 lists the masses of the nine major planets with their moons as fractions of the Sun mass, their mean distance to the Sun, and the average times needed for one revolution around

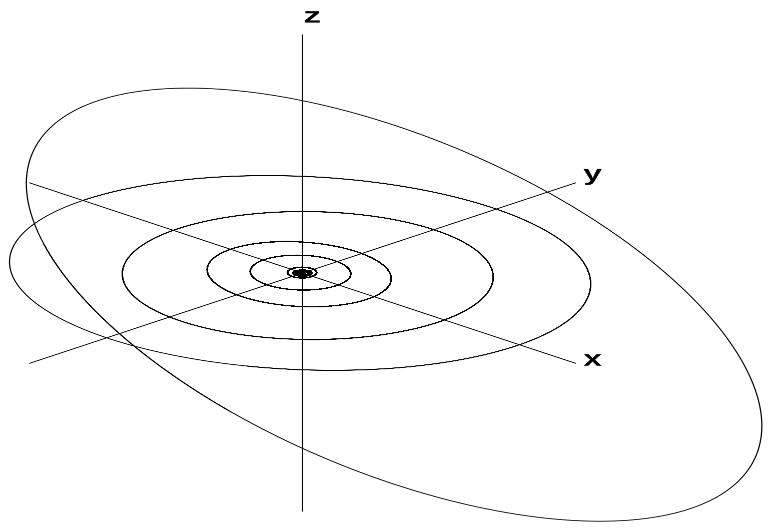


Figure 5.2: Orbits of the outer planets Jupiter, Saturn, Uranus, Neptune, and Pluto around the Sun.

the Sun. It should be noted that, according to *Kepler's Third Law*, the period T and the mean distances a are related via $T = a^{3/2}$.

Planet	*Mass (m)*	*Dist. a*	$a^{3/2}$	*Time T*
Mercury	1/6023600	0.39	0.2436	0.240
Venus	1/408523.5	0.72	0.6109	0.615
Earth	1/328900.5	1.00	1.0000	0.999
Mars	1/3098710	1.52	1.8739	1.881
Jupiter	1/1047.355	5.20	11.858	11.856
Saturn	1/3498.5	9.54	29.466	29.424
Uranus	1/22869	19.19	84.064	83.747
Neptune	1/19314	30.07	164.89	163.723
Pluto	1/3000000	39.48	248.07	248.021

Table 5.4: Masses, mean distances, and periods of the major planets. Masses are given as fractions of the Sun mass M, mean distances in astronomical units, and times in Julian years [125].

In addition to the Sun, the nine major planets and their moons, the solar system contains numerous other small objects, mostly asteroids and comets, the largest of which are commonly referred to as *minor planets* [134]. The sixteen largest asteroids in the solar-system have diameters between 240 km and 1000 km and are part of the main asteroid belt, which consists of several thousand objects of non-negligible mass and size and is located between the planets Earth and Mars. During past space craft missions, the asteroids Ceres, Pallas, Vesta, Iris, and Bamberga, all of which are orbiting the Sun in the main asteroid belt, have been found to have a noticeable gravitational effect on the motion of objects in the Earth-Mars range [125]. Hence, an accurate formulation of the equations of motion of objects on near-earth orbits has to include their gravitational effects. We will revisit this when we discuss an accurate mathematical model for the motion of near-earth asteroids in Sec. 5.1.1. Finally, it should be noted that the term asteroid belt is misleading, since it is by no means a dense object. In fact, several spacecrafts have passed it undamaged and the mean distance between non-negligible objects in the main asteroid belt is several million kilometers.

The Equations of Motion

In the previous sections we have given an overview of the solar system and the classical description of gravitation. Now we present an accurate mathematical model [139, 125, 44, 76] describing the motion of point masses in a solar system barycentric coordinate system by a second order differential equation. The model includes Newtonian effects of the Sun and the major and minor planets and relativistic corrections to the classical Newtonian description of gravitation of the Sun and the major planets.

$\vec{r}_j$, $\dot{\vec{r}}_j$, and $\ddot{\vec{r}}_j$ denote the solar-system barycentric position, velocity, and acceleration of the j-th major planet, where $j = 0$ denotes the Sun. γ and β are the post-Newtonian parameters measuring space curvature produced by the unit rest mass and nonlinearity in the superposition of gravity; in accordance with general relativity, both are assumed to be one. The masses have been combined with the gravitational constant into the coefficients $\mu_j = Gm_j$, and the position of the point mass is given by $\vec{r}$. The distances between point mass and major planets are expressed by $r_j = \|\vec{r} - \vec{r_j}\|_2$, the respective velocities are denoted by $v_j = \|\dot{\vec{r_j}}\|_2$ and $v = \|\dot{\vec{r}}\|_2$. N is the number of minor planets that are included in the computation and c is the velocity of light.

$$\begin{aligned}
\ddot{\vec{r}} \; = \; & \sum_{j=0}^{9} \frac{\mu_j\left(\vec{r_j} - \vec{r}\right)}{r_j^3} \left\{ 1 - \frac{2\left(\beta + \gamma\right)}{c^2} \sum_{k=0}^{9} \frac{\mu_k}{r_k} - \frac{\left(2\beta - 1\right)}{c^2} \sum_{k \neq j} \frac{\mu_k}{r_{jk}} + \left(1 + \gamma\right) \left(\frac{v_j}{c}\right)^2 \right. \\
& \left. + \gamma \left(\frac{v}{c}\right)^2 - \frac{2\left(1 + \gamma\right)}{c^2} \dot{\vec{r}} \cdot \dot{\vec{r_j}} - \frac{3}{2c^2} \left(\frac{\left(\vec{r} - \vec{r_j}\right) \cdot \dot{\vec{r_j}}}{r_j} \right)^2 + \frac{1}{2c^2} \left(\vec{r_j} - \vec{r}\right) \cdot \ddot{\vec{r_j}} \right\} \\
& + \frac{1}{c^2} \sum_{j=0}^{9} \frac{\mu_j}{r_j^3} \left(\left(\vec{r} - \vec{r_j}\right) \cdot \left(\left(2 + 2\gamma\right) \dot{\vec{r}} - \left(1 + 2\gamma\right) \dot{\vec{r_j}} \right) \right) \left(\dot{\vec{r}} - \dot{\vec{r_j}} \right) \\
& + \frac{3 + 4\gamma}{2c^2} \sum_{j=0}^{9} \frac{\mu_j}{r_j} \ddot{\vec{r_j}} + \sum_{\nu=1}^{N} \frac{\mu_\nu \left(\vec{r_\nu} - \vec{r}\right)}{r_\nu^3}
\end{aligned} \tag{5.3}$$

This differential equation shows that the force on a point mass is essentially given by the Newton force inflicted by all other major objects in the solar system and some correction terms in the order of $1/c^2$. Although Eqn. (5.3) gives only a partial description of the acceleration exerted on a point mass in the solar system, it provides a highly accurate description of the motion, and it has been used for the preparation of the most accurate ephemeris system so far [125].

5.1.2 Geometric Description of Kepler Orbits

As mentioned earlier, the Kepler problem of predicting the motion of two point masses under their mutual Newtonian gravitation permits only conic sections as orbits, with the barycenter being the focus. The orbits lie in planes that are perpendicular to the system's angular momentum and pass through the barycenter. In general, the shape of conic sections is determined by the *polar equation*, which relates distance to the barycenter r and the angle in the plane of motion φ via

$$r\left(1+\varepsilon\cos(\varphi)\right) = a\left(1-\varepsilon^2\right). \tag{5.4}$$

Depending on the value of the *eccentricity* ε, the orbit is given by one of the following geometric figures:

Eccentricity	*Orbit*
$\varepsilon = 0$	Circle
$0 < \varepsilon < 1$	Ellipse
$\varepsilon = 1$	Parabola
$\varepsilon > 1$	Hyperbola

Table 5.5: Geometric shape of conic sections, depending on the eccentricity ε.

In this section we summarize how Kepler ellipses can be described in terms of *orbital elements* and how this description can be used to compute the positions and velocities of planets in a barycentric cartesian coordinate system whose x/y-plane coincides with the ecliptic of the Earth.

Orbital Elements

The shape and orientation of Kepler ellipses in the solar system can be described unambiguously by six parameters, the so-called orbital elements. While several equivalent choices for these parameters exist, the following list shows one particular selection that significantly simplifies the subsequent computations:

- *Longitude of ascending node Ω:* The angle, along the ecliptic, from the vernal point to the ascending node, which is the intersection between the orbit and the ecliptic, where the planet moves from south of to north of the ecliptic; i.e., from negative to positive latitudes.

- *Inclination i:* The tilt of the orbit plane relative to the ecliptic.
- *Argument of perihelion ω:* The angle from the ascending node to the perihelion, along the orbit.
- *Semi-major axis a:* The mean distance to the Sun.
- *Eccentricity ε:* The parameter describing the shape of the orbit (c.f. Tab. 5.5).
- *Mean anomaly M:* The product of the *mean motion* of the orbiting body and the time since the body passed the *pericenter* at some specific point in time.

The mean anomaly is best described if one imagines a small circle within the orbit, and centered on the focus f. Then M is the angle between the perihelion P, the focus f, and the position the planet would occupy if it was traveling at a constant angular speed. While the last three elements describe the orbit ellipse themselves, Ω, i, and ω describe the orientation of this ellipse in space. Other equivalent choices of the orbital elements include the *mean longitude* $L = M + \omega + \Omega$, the *time at perihelion*, and the *mean daily motion* n. For reference, Appendix A lists the orbital elements and daily rates of change of the major planets.

Cartesian Coordinates

Since the equations of motion (5.3) describing the acceleration of a point mass in the solar system are only valid in a cartesian system of inertia, for practical computations it is usually necessary to convert orbital elements into cartesian coordinates. This transformation is based on the *eccentric anomaly* E, which is due to Kepler and connects the cartesian coordinates of a planet with its orbital elements.

Consider a large circle of radius equal to the semi-major axis of the orbit. Let the center of this circle be at c. Now, on the diameter of the circle which passes through the long axis of the ellipse, raise a perpendicular which meets the planet at r on the ellipse. Continue this perpendicular until it crosses the large circle and call the point where the perpendicular crosses the large circle Q. Then the angle E is called the *eccentric anomaly* (c.f. Fig. 5.3). The mean and eccentric anomalies are related by

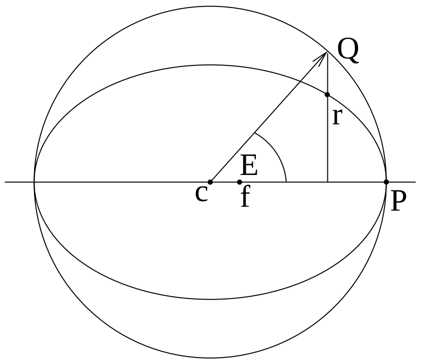


Figure 5.3: Illustration of the eccentric anomaly E

the implicit equation

$$M = E - \varepsilon \cdot \sin(E). \tag{5.5}$$

If the eccentricity ε is small, a Newton method can easily find E for given values of M and ε. Once E has been determined, the cartesian positions $\vec{r} = (x, y, z)$ and velocities $\dot{\vec{r}} = (\dot{x}, \dot{y}, \dot{z})$ are computed from the following equations [118, 93]:

$$n = \sqrt{\frac{G\,(M+m)}{a}}\,\frac{1}{(1-\varepsilon\cos(E))} \tag{5.6a}$$

$$\alpha = a\,(\cos(E) - \varepsilon) \tag{5.6b}$$

$$\beta = a\sqrt{1-\varepsilon^2}\,\sin(E) \tag{5.6c}$$

$$\gamma = -n\sin(E) \tag{5.6d}$$

$$\delta = n\sqrt{1-\varepsilon^2}\,\cos(E) \tag{5.6e}$$

$$p_x = \cos(\omega)\cos(\Omega) - \sin(\Omega)\sin(\omega)\cos(i) \tag{5.7a}$$

$$p_y = \cos(\omega)\sin(\Omega) + \cos(\Omega)\sin(\omega)\cos(i) \tag{5.7b}$$

$$p_z = \sin(\omega)\sin(i) \tag{5.7c}$$

$$q_x = -\sin(\omega)\cos(\Omega) - \sin(\Omega)\cos(\omega)\cos(i) \tag{5.7d}$$

$$q_y = -\sin(\omega)\sin(\Omega) + \cos(\Omega)\cos(\omega)\cos(i) \tag{5.7e}$$

$$q_z = \cos(\omega)\sin(i) \tag{5.7f}$$

$$x = \alpha \cdot p_x + \beta \cdot q_x \tag{5.8a}$$

$$y = \alpha \cdot p_y + \beta \cdot q_y \tag{5.8b}$$

$$z = \alpha \cdot p_z + \beta \cdot q_z \tag{5.8c}$$

$$\dot{x} = \gamma \cdot p_x + \delta \cdot q_x \tag{5.8d}$$

$$\dot{y} = \gamma \cdot p_y + \delta \cdot q_y \tag{5.8e}$$

$$\dot{z} = \gamma \cdot p_z + \delta \cdot q_z \tag{5.8f}$$

The origin of this coordinate system coincides with the barycenter of the combined two-body system of the Sun and planet, but the coordinates can easily be transformed into a solar system barycentric coordinate system by translation.

Planetary Motion and Perturbations

For the two-body problem, the six orbital elements are constants and are determined uniquely by the six initial conditions of the position $\vec{r}$ and the velocity $\dot{\vec{r}}$. Due to the overwhelming mass of the Sun and the large distances between the planets, even in the presence of other planets, the orbits of the planets are very close to being ellipses. Minor corrections to the two-body Kepler ellipses of the planets are due to:

- influences from planets other than the Sun,

- deviations of the force from the simple inverse square law (relativistic corrections).

It is convenient to treat the orbits as instantaneous ellipses whose parameters are defined by the instantaneous values of the position and velocity vectors, since for small perturbations the orbits are approximately ellipses. However, these perturbations cause the six formerly constant parameters to vary slowly, and the instantaneous perturbed orbits are called *osculating ellipses*: the osculating ellipses are the elliptical orbits that would be assumed by the bodies if all the perturbing forces and corrections were suddenly turned off.

This perturbation theoretical approach of describing planet orbits relative to slowly varying elliptical *reference orbits* has proven itself extremely successful in particle optical systems [17, 87], and based on the arbitrary order approach of COSY Infinity [23, 26], sufficiently accurate smooth description of planet orbits can be computed even for long-term integrations. Moreover, in light of the Taylor model approach, this strategy has the particular advantage that the orbital elements of planets are almost linear functions of time, which eases the description by Taylor models and increases the accuracy of the enclosures.

5.1.3 Verified Orbit Integration

In the previous sections we summarized the concepts that govern the motion of masses in the solar system and presented the principle of describing nearly closed orbits by perturbations to osculating ellipses. Here we show how these concepts can be used for the verified integration of asteroid orbits. Since this involves highly non-linear equations, large sets of initial conditions, and substantial uncertainties in the mathematical description, we have found conventional verified integration schemes [79, 99] unable to handle the problem of creating a guaranteed ephemeris for asteroids in the solar system. As indicated earlier however, Taylor model based integration schemes [83, 85] can offer accurate long-term verification even under these challenging conditions. To illustrate this, we have based the integration on Taylor model methods, Eqn. (5.3), and the ephemeris DE405 [130].

The basic idea of the integration is that the presence of a relatively small asteroid will not change the orbits described by the DE405 ephemeris. While this is clearly only an approximation, due to the mass ratio between the planets and asteroids and due to the relatively large distances separating them, the deviations from this approximation are in fact extremely small and will be accounted for through general error bounds on the planets' positions and velocities. Thus, instead of integrating the full $(60 + 3 \times N)$-dimensional system involving the Sun, the nine major planets, and the N minor planets as Eqn. (5.3) would suggest, we use osculating ellipses together with the corresponding errors of the ephemeris to obtain Taylor models for the positions and velocities of the planets. These Taylor models describing positions and velocities of the planets and interval enclosures for effects not described by Eqn. (5.3) are then used for a Taylor model based integration of the six-dimensional equations of motions for the asteroid alone.

This particular approach allows a substantial reduction of the problem's dimensionality and makes its verified integration feasible. Utilizing high-order Taylor models limits the wrapping effect and enables us to obtain long-term predictions, even for large sets of initial conditions. Moreover, the approach does not lead to a dramatic inflation in the size of enclosures of the final coordinates. In the remainder of this section we present a detailed discussion of how the verified integration of Eqn. (5.3) can be achieved and results will be presented in Sec. 5.1.4.

Accuracy of the Ephemeris

The mathematical models and concepts presented so far allow the computation of accurate planet orbits. For the computation of the ephemeris DE405 [125, 130], which has been used to compute the motion of asteroids, rockets, and satellites in the solar system [142], the following effects have also been included:

- Newtonian gravitation of the Sun and the major planets
- relativistic effects of orders up to $1/c^2$
- Newtonian effects of the 300 largest minor planets

Since planets are not rigid spheres with a uniform mass density, an accurate integration includes the deviations from the simple point-mass concept. The following factors have been included in the preparation of the DE405/LE405 ephemeris:

- the difference between geometric centers and barycenters (figure effects)
- Earth tide effects
- physical librations of moons

Finally, the results of the numerical integration have been adjusted to past measurements. Measurements are, among other things, based on the following techniques:

- meridian transits (i.e., optical triangulation),
- satellite astronomy,
- radar ranging to planet surfaces (including surface models),
- radar ranging to spacecrafts,

- laser ranging to lunar reflectors.

Residuals of the ephemeris DE405/LE405 have been presented in [130] and it has been shown that the ephemeris DE405 is indeed suitable to accurately predict the positions and velocities of massive objects in the solar system [142]. Since Jupiter has the second largest mass of all objects in the solar system, its positional errors translate into potentially large errors in the integration results of asteroids and its residuals with respect to DE405 are therefore of great importance. Fig. 5.4 shows these residuals with respect to DE405, obtained via *Very Long Baseline Interferometry* observations from the Galileo spacecraft.

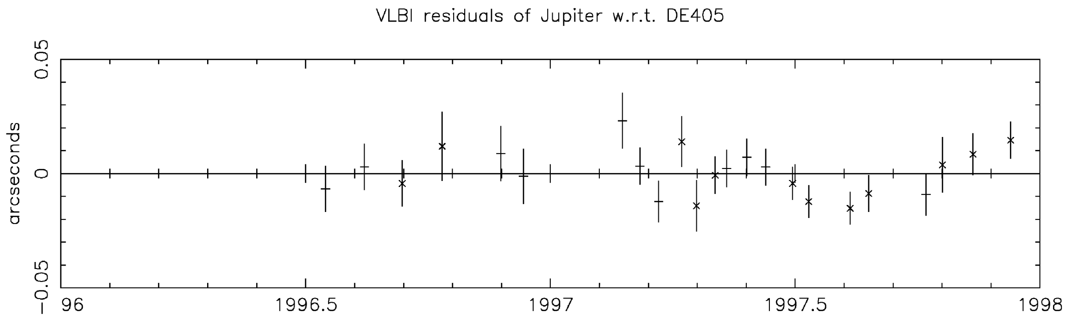


Figure 5.4: Residuals of Jupiter; two Very Long Baseline Interferometry observations from the Galileo spacecraft; courtesy of [130].

The residuals shown in Fig. 5.4 translate into positional uncertainties of only about 100 km. Combining this with even higher accuracies for the inner planets (the ephemeris system DE405 is believed to be accurate to 0.001 arc seconds [130]), it is possible to predict the positions and velocities of the Sun and the major and minor planets quite accurately for the next 50 to 100 years. As a result, the data of DE405 can be used to obtain time-dependent descriptions of the orbital elements. These can then be used to compute Taylor model descriptions of the barycentric cartesian coordinates of the planets that are sufficiently accurate for verified long-term integrations. Consequently, Taylor model integration schemes can be used to obtain verified long-term results for the motion of small objects in the solar system.

In addition to the effects included in Eqn. (5.3), our integration also considers the acceleration due to the rotation of the solar system barycenter around the galaxy; i.e., the deviation of the solar system from an inertial frame of reference. While this contribution amounts only to about $5{\cdot}10^{-8}$ [7], it is in the same order as the relativistic corrections and can therefore not be neglected. By determining the direction of this acceleration, we include it in the accurate model of determining the acceleration on a point mass in the solar system given by Eqn. (5.3).

The center of our galaxy is believed to coincide with *Sgr A** and its ecliptic latitude β and longitude λ at the year 2000 are given by [127]:

$$\beta = 266.85 \text{ deg}, \tag{5.9}$$

$$\lambda = -5.61 \text{ deg}. \tag{5.10}$$

Translating these into cartesian coordinates allows the determination of a vector describing the acceleration of objects in the solar system due to the rotation of the solar system barycenter around the center of the galaxy. Since this vector varies over time, and since its direction and length are based on measurements, it is necessary to include error bounds on it. However, since these errors are several orders of magnitude smaller than other errors, they do not challenge the rigorous integration.

Additional Effects

Traditional numerical integration schemes, as used in the preparation of the ephemeris DE405, can only include quantifiable effects. Moreover, in the framework of traditional floating point integration schemes it is useless to account for quantities that have smaller effects on the result than the known uncertainties in other more influential parameters. However, some of these omitted effects are actually indirectly accounted for, since the results of the ephemeris have been fitted to observational data by modifying the underlying physics [125]. The most important of the omitted forces in the preparation of the ephemeris DE405 and Eqn. (5.3) are:

- relativistic effects of orders higher than $1/c^2$,
- relativistic corrections for the minor planets,
- non-gravitational forces (e. g. friction),
- static background forces.

Verified integration methods, however, cannot ignore these contributions, regardless of their actual influence. The easiest, and usually sufficiently accurate, way of including them in the mathematical models is to add error intervals to the right hand side of the equations of motion (5.3).

Taylor Models for Orbits

To obtain an enclosure for the acceleration of a point mass in the presence of the Sun and the planets, Eqn. (5.3) requires verified descriptions of the positions, velocities, and accelerations of the Sun and planets. As outlined before, we obtain these Taylor models from the time dependent descriptions of the osculating ellipses. The orbital elements of the planets are slowly time-dependent functions with a dominating linear part. As such they can easily be modeled within the differential algebraic framework presented in Sec. 2.1. Within the differential algebraic framework of COSY Infinity, these polynomials can be carried through Equations (5.6 – 5.8) to obtain time dependent polynomials describing the positions and velocities of the individual planets and the Sun. Taylor models for the cartesian coordinates are then constructed by utilizing bounds obtained from the residual analysis of the DE405 ephemeris [130] and an accuracy analysis of the polynomial descriptions.

It is important to underscore that we currently do not have long-term data for the orbital elements for which the described procedure yields polynomial descriptions that are equivalent in accuracy to DE405. However, the system provides the framework necessary to include time dependent descriptions of the orbital elements that allow the computation of such orbits. Currently, the implementation assumes a linear time-dependence of the orbital elements as shown in Appendix A. Thus, while the results presented in Sec. 5.1.4 are not verified with respect to the actual solar system, they are consistent and accurate within the given description of the orbits. Moreover, our results indicate that a more accurate description of the orbital elements will neither degrade the performance nor the sharpness of our integration method.

Initial Conditions and Observational Data

A major challenge for verified integration of asteroid orbits is the problem of obtaining guaranteed sets of initial conditions for the asteroid at some time. Since actual measurements are the main source for initial conditions, they do not have rigorous error bounds, but come with *confidence intervals* instead. This is based on the assumption that, in the absence of systematic errors, the measurements have a Gaussian distribution around the correct value. Hence, the quantity that describes the width of the confidence interval most naturally is σ, the parameter describing the width of the Gaussian curve. While this description leaves the possibility of undetected systematic errors, measurements are frequently confirmed using independent methods. Thus, it is usually appropriate to assume that initial conditions obtained from a 3σ neighborhood are good enclosures of the actual initial conditions.

Due to the numerous measurements of the inner planets, their positions and velocities are known to a very high degree of accuracy. For asteroids on the other hand, often only a few measurements are available. However, if the asteroid ever comes close to Earth, high accuracy measurements using radar Doppler effects and radar delays will yield accurate results with small uncertainties for the initial conditions [142].

In the computation of the results shown in Sec. 5.1.4, we assumed that the initial positions and velocities of the asteroid of interest can be enclosed in cartesian boxes of approximately ± 75 km and $\pm 2 \cdot 10^{-5}$ km/s. Since current sigma position uncertainties for near Earth asteroids tend to be between 10 km and 10^3 km [142], obtaining enclosures this sharp poses a significant challenge for astronomers. However, the actual results shown in Fig. 5.6 indicate that the Taylor model methods can successfully handle initial condition sets as wide as 10^{-5} AU, or approximately 1500 km. Thus, for the performance of the actual integration with algorithms based on Taylor models, the width of the enclosure boxes is not a significant factor.

5.1.4 Results and Discussion

After combining the theory of Taylor model based ODE integration with the detailed model of the underlying physical concepts presented earlier, we have performed several verified integrations of asteroid orbits using Taylor model methods. Here, we show results from these integrations to illustrate the performance and applicability of these integration tools.

All computations presented in this section are based on special units for the distance and time that aid the integration process:

- Time is measured in *time units (TU)*, where one time unit equals $365.25/2\pi \approx 58.131$ days.
- Distance is measured in astronomical units (AU).

Since the Earth moves on an almost circular orbit, it travels approximately 2π astronomical units in 2π time units. Hence, in these units the velocity of the Earth on its orbit is approximately $v = (2\pi \text{ AU})/(2\pi \text{ TU}) = 1 \text{ AU/ TU}$. Since in these units both the distance of the Earth to the coordinate center and its orbital velocity are close to unity, all components of the right hand side of the differential equation describing the asteroid's orbit are of similar magnitude. This simplifies the error analysis of the integration tools, since it removes the necessity of weighting the errors of the individual components against each other. Moreover, this aids the step size control of the integrators, since it turns the potentially stiff system, with different step sizes for different components of the system, into a non-stiff one.

Integration Results for 1997 XF11

In this section we present results obtained from integrations of the near-Earth asteroid 1997 XF11, which has a high eccentricity of $\varepsilon = 0.48$, low-inclination orbit with $i = 4.10$, and a diameter of several kilometers. It orbits the Sun with a period of approximately 1.7 years. Since one of the two intersection points of its orbit with the ecliptic is in the vicinity of the Earth's orbit ellipse, there is a non-vanishing probability for close encounters between the asteroid and Earth. Tab. 5.6 lists predictions for the close encounters between the years 2000 and 2040: the closest approach distance is predicted to be 0.006 astronomical units in the year 2028. While this is about one quarter of the distance between the Earth and the Moon, it is close enough to warrant further research into the future orbit of the asteroid 1997 XF11. In fact, 1997 XF11 is an important test case for verified solar system integrations, since a successful integration would be able to verify the non-impact in the year 2028.

The integrations discussed in this section are based on initial conditions obtained from the HORIZONS system [128]. As indicated in Sec. 5.1.3, we assumed the initial conditions to be in boxes of 10^{-7} and 10^{-6} for the positions and velocities. The exact position $\vec{r_0}$ and velocity $\dot{\vec{r_0}}$ at the initial time t_0 are given by:

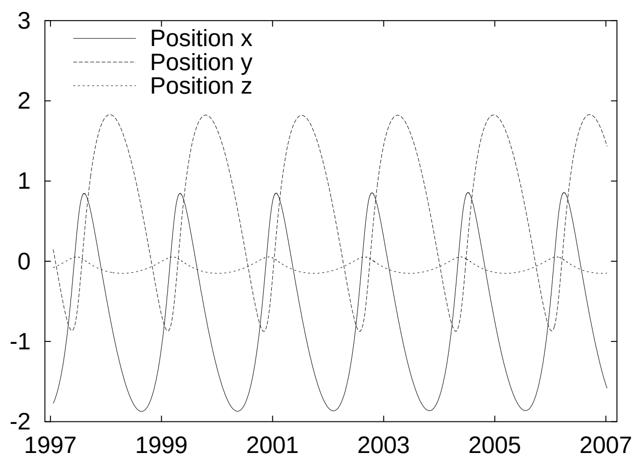
$$\begin{aligned} t_0 &= \text{JD2450465.5} \approx \text{January } 17, 1997 \\ \vec{r_0} &\in (-1.772691\ldots, +0.148722\ldots, -0.079284\ldots) \pm 0.5 \cdot \left(10^{-7}, 10^{-7}, 10^{-7}\right) \\ \dot{\vec{r_0}} &\in (+0.237203\ldots, -0.612525\ldots, +0.045832\ldots) \pm 0.5 \cdot \left(10^{-6}, 10^{-6}, 10^{-6}\right). \end{aligned} \tag{5.11}$$

The cartesian positions and velocities of the asteroid 1997 XF11 during the integration interval starting at t_0 are shown in Fig. 5.5 and were obtained from the relativistic integration discussed in Sec. 5.1.4. Since this problem is dominated by the two-body interaction between the Sun and the asteroid, the evolution of the coordinates exhibits the periodic behavior of elliptical orbits with high eccentricity.

In the remainder of this section we present results obtained from two different integrations of 1997 XF11's orbit. The first integration uses Eqn. (5.3), while the mathematical model of the second integration does not include the relativistic corrections to the classical Newton gravitation. As such, the two integrations allow us to measure the actual influence of the correction terms on the asteroid's final coordinates, which gives an estimate on the computational cost and final benefit associated with these corrections.

Integration with Relativistic Corrections

In this section we present results from a ten year integration of the asteroid 1997 XF11, using Eqn. (5.3) with relativistic corrections and initial conditions boxes given by Eqn. (5.11). The asteroid's positions obtained from this integration are shown in Fig. 5.5. As indicated earlier, the results demonstrate that the asteroid is moving on an almost elliptical orbit around the Sun. More interestingly though, Fig. 5.6 gives the size of the enclosures for the coordinate x and the velocity $\dot{x}$. It is important



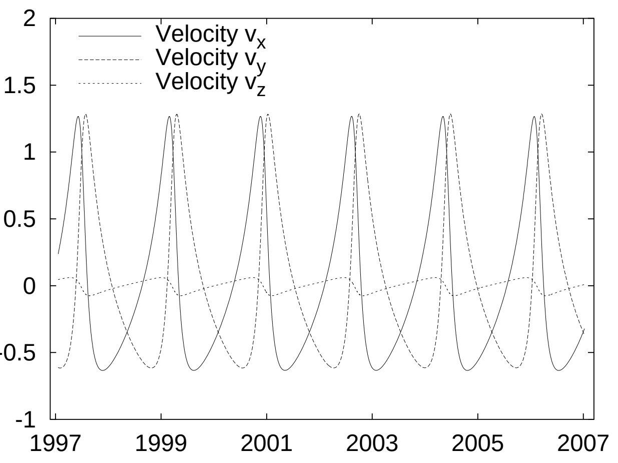
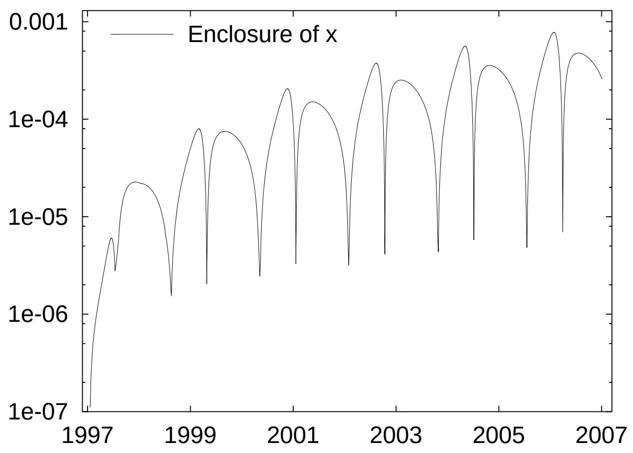


Figure 5.5: Cartesian positions (in AU) and velocities (in AU/TU) of 1997 XF11 during the ten year integration interval.

to note that the sizes of these enclosures do not just grow monotonically with time, but even decreases at certain points. Moreover, the enclosures have a very small average growth rate, which is the main reason for the ability of the Taylor model based integrator to propagate initial conditions over large time intervals.

Tab. 5.6 lists the closest approach distance between 1997 XF11 and the Earth in the year 2028 as 0.006 AU. By extrapolating the enclosures that can be obtained from the Taylor model integration on the other hand, we observe that the enclosures at that point will likely be smaller than 0.006 AU. Since this is sufficiently accurate to guarantee a non-impact, it shows that the Taylor model approach can indeed be used



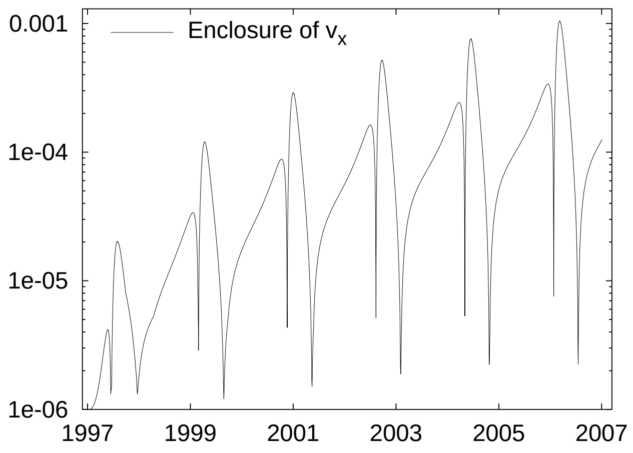


Figure 5.6: Logarithmic plot of the diameters of enclosures for the positions (in AU) and velocities (in AU/TU) of 1997 XF11 during the ten year integration interval.

to obtain meaningful results from verified long-term integrations of real world systems. In Sec. 5.1.5 we will compare this with similar results obtained from conventional verified integration tools.

To illustrate the technical aspects of the integration method, Fig. 5.7 shows how the automatic error and step size control of the Taylor model integration method work in practice. The maximally allowed local error was set to 10^{-9} and Fig. 5.7 shows that the integrator has been able to meet this goal by reducing the step sizes accordingly. A comparison between Figs. 5.7 and 5.5 shows a strong correlation

Date	*Min. Dist. (AU)*
October 31, 2002	0.064
June 10, 2016	0.180
October 26, 2028	0.006
June 8, 2035	0.174
November 4, 2040	0.150

Table 5.6: Dates and minimal distances of the predicted closest Earth approach distances for the near-Earth asteroid 1997 XF11 between the years 2000 and 2040 [73].

between the local error and the current location of the asteroid on its orbit: the local error tends to increase, with a corresponding reduction in the step size, whenever the asteroid is closest to the Sun. In these regions its acceleration is at a maximum and the asteroid moves with its largest orbital velocity. It should be noted that a significant contribution to the local errors stems from the uncertainties of measured quantities and the planets' orbits. Thus, the main source for local error can therefore not be subject to verified error control. Consequently, the desired local error is close to the optimum for this particular problem.

Integration without Relativistic Corrections

Here we present results of a ten year integration similar to the one discussed in the previous section. However, this integration is based on the classical Newton gravitation without any of the relativistic corrections included in Eqn (5.3). This allows us to quantify the actual effects of the relativistic corrections on the computed orbit of 1997 XF11. Hence, the analysis determines the importance of a complete model including the relativistic corrections given by Eqn. 5.1 and puts a measure on the computational overhead of these corrections.

For this integration, we used the same initial conditions as in the previous example. Moreover, the automatic step size control of the Taylor model integrator chooses

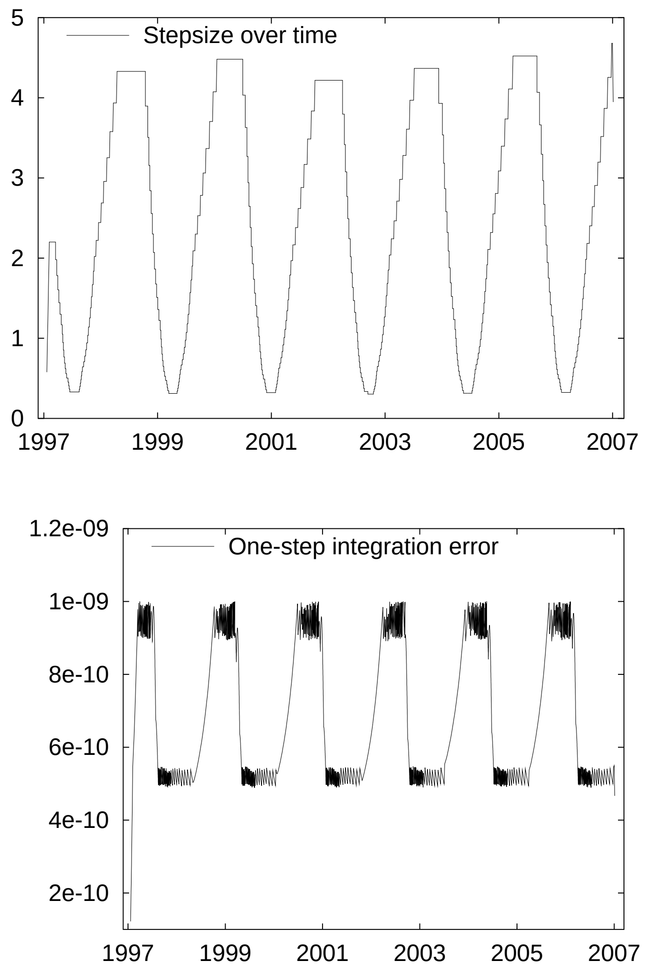
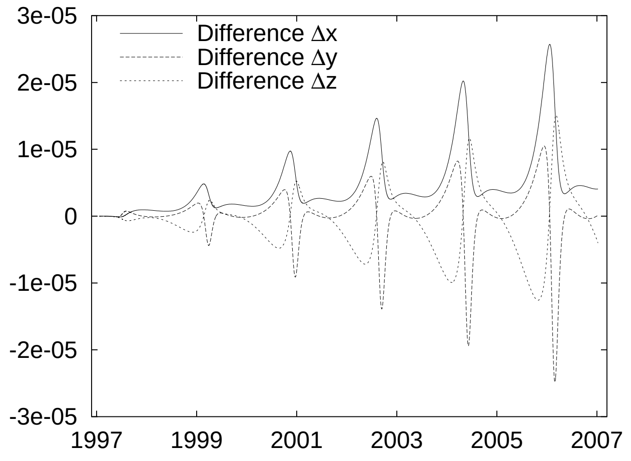


Figure 5.7: Evolution of the step size and one-step integration errors during the ten year integration of 1997 XF11.

the same step sizes as in the previous computation, indicating that the relativistic corrections have only a small influence on the final result.

Firstly, it should be noted that the integration without the relativistic corrections requires only about one tenth of the CPU time necessary for the accurate integration discussed before. Secondly, the enclosures of the computed final positions and velocities are virtually indistinguishable from the ones shown in Fig. 5.6. The differences between the positions and velocities computed by the corrected and the uncorrected integrations are shown in Fig. 5.8. Since the differences are much smaller than the sizes of the enclosure boxes, the performance of verified solar system integration could



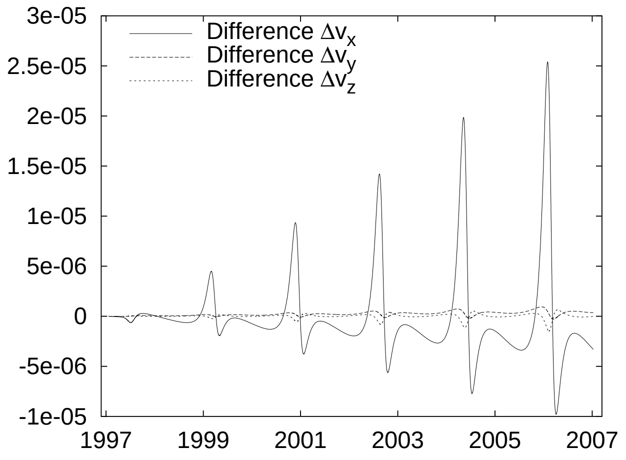


Figure 5.8: Differences in the computed positions and velocities of the asteroid 1997 XF11 between the ten year integrations with and without relativistic corrections.

be improved significantly by finding guaranteed enclosures of the relativistic corrections in Eqn. (5.3) that are easier to compute than the corrections themselves and are not overly pessimistic. The net effect of this approach would be a relatively small tradeoff in accuracy for a substantial gain in integration speed. However, such rigorous estimates on the size of the relativistic corrections are not known and it is not clear if they could be rigorously computed with an effort that is significantly smaller than the one needed to compute the corrections proper.

Shrink Wrapping

An important aspect of the Taylor model based integration method is the ability to dynamically shrink the size of the remainder bounds during the integration. This is extremely useful since it delays the exponential inflation of errors by several orders in time. This dynamic reduction of errors is called *shrink wrapping* [28].

The main idea behind shrink wrapping is illustrated in Fig. 5.9: at certain points during the integration, the linear part of the Taylor model M propagating the initial conditions is enlarged, resulting in a new Taylor model M_S. The polynomial part of this new Taylor model maps the set of initial conditions to a set that properly contains the original enclosure of the final coordinates plus the remainder bounds. At this point, the remainder bounds of M_S can be shrunk to zero width, since the polynomial map of M_S already ensures containment of the image of the initial conditions in the final enclosure. However, this parts with the Taylor model approach viewing the reference polynomial as the Taylor expansion of the *flow* of the differential equation. On the other hand, shrink wrapping enables Taylor model based integration methods to obtain long-term results without an overly pessimistic enclosure of the final positions and velocities. Since the errors of the right hand side of Eqn. (5.3) are relatively large, shrink wrapping has been performed after each integration step in the computation of the results presented in Sec. 5.1.4.

The *shrink factor* is the factor by which the linear parts of the Taylor models' reference polynomials are enlarged. We distinguish two kinds of shrink factors:

- The one-step shrink factor determines the amount of scaling in a single shrink wrap operation.
- The accumulated shrink factor gives the amount by which the linear parts have

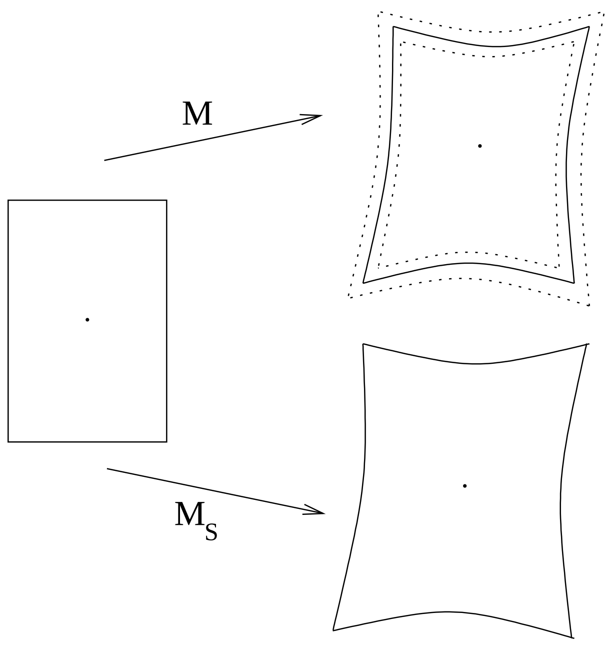


Figure 5.9: Illustration of the shrink wrapping method used in the verified integration of asteroid orbits.

been scaled over the whole integration at this point.

Fig. 5.10 shows the values of both shrink factors, computed during the ten year integration with relativistic corrections presented in Sec. 5.1.4. The figures illustrate two important and somewhat surprising aspects of the shrink wrapping method: on average, the one-step shrink factor decreases exponentially over time and the total shrink factor grows only linearly and not exponentially. This behavior is particularly helpful for the computation of long term results, since it prevents an exponential inflation of the enclosures of the final positions and velocities over the integration interval. Comparisons between integrations with and without shrink wrapping show that the availability of the method is a key ingredient of an overall strategy geared towards verified long-term integration results [86].

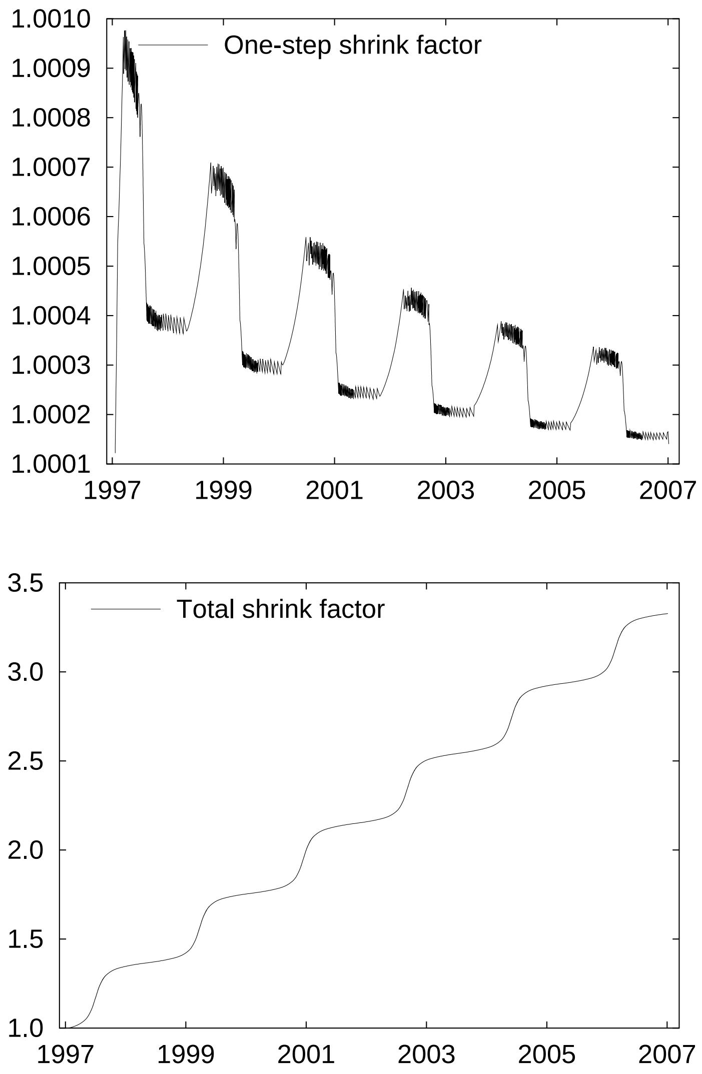


Figure 5.10: One-step and total shrink factors obtained during the ten year integration of the asteroid 1997 XF11.

5.1.5 Comparison with AWA

During the last decade, AWA [79, 80] has become a standard benchmark for verified ODE integration methods. While it tends to be overly pessimistic for complicated problems, the method works reasonably well for converging linear problems. In this section we compare the performance of AWA with the Taylor model based integration schemes used before, and we will demonstrate that the Taylor model approach performs favorably when compared with the conventional interval techniques used in

AWA.

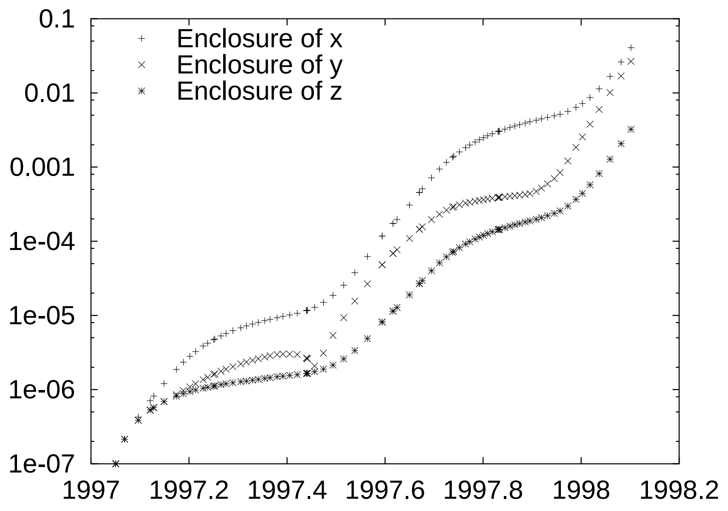
Since AWA lacks advanced control structures which would aid the implementation of the right hand side as given by Eqn. (5.3), and cannot handle uncertainties in the right hand side of the differential equations very well, we have only used it for the integration of the two-body problem of the Sun and the asteroid 1997 XF11.

$$\ddot{\vec{r}} = -\frac{\gamma}{\|r\|^3}\vec{r}. \tag{5.12}$$

$\gamma = 0.9986$ is the value of the product of gravitational constant G and the Sun mass M in the units that have been used for all previous integrations. It should be noted that Eqn. (5.12) represents a much simpler problem than the one previously discussed, since it contains only one planet orbiting the Sun, no relativistic corrections, no corrections accounting for the solar system being a rotating coordinate system, and no uncertainties for the positions and velocities of the Sun.

As in the previous cases, the initial conditions are given by Eqn. (5.11), and AWA has been set to integrate the system in order 18 with an initial step size of 0.0001. For comparison, the same integration has been performed with Taylor models of order ten and an initial step size of 0.1. In both cases the maximum local errors were set to 10^{-11}. It should be noted that the model discussed in this section is an extremely simplified version of the real problem described by Eqn. (5.3) and has therefore no relevance for the verified integration of solar system dynamics. Also, while we would have liked to perform the integrations over time intervals spanning more than one year, AWA was unable to integrate the system further, since the wrapping effect led to an exponential inflation of the enclosures. (The time evolution of the position and velocity enclosures is shown in Fig. 5.11).

The results given in Fig. 5.11 show that the Taylor model approach allows the computation of much sharper bounds on the final coordinates than conventional interval



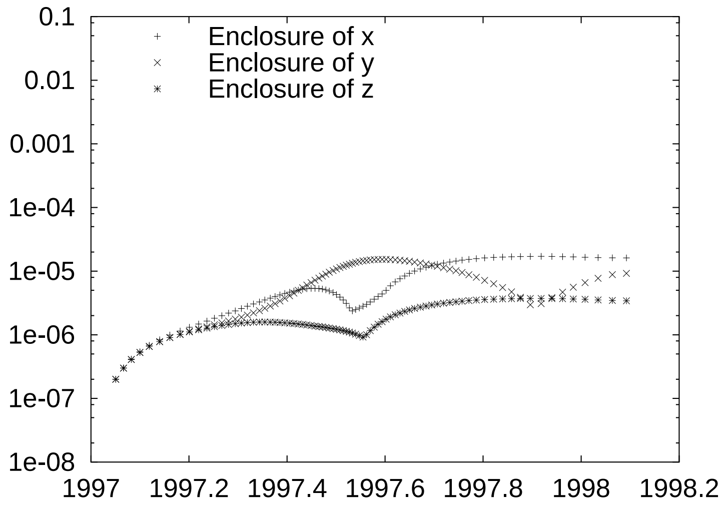


Figure 5.11: Logarithmic plot of the diameters of the enclosures for positions obtained by AWA (upper) and the Taylor model based integrator.

techniques, represented by AWA. At the point where AWA terminates the integration, the Taylor model enclosures are almost four orders of magnitude sharper than the corresponding results obtained by AWA. Moreover, at the final point of integration, the enclosures obtained by AWA exhibit a large exponential growth while the Taylor model results show a much smaller growth rate and even decrease at certain times. This smaller rate of increase gives the Taylor model methods the ability to integrate over much larger time intervals than conventional interval based tools.

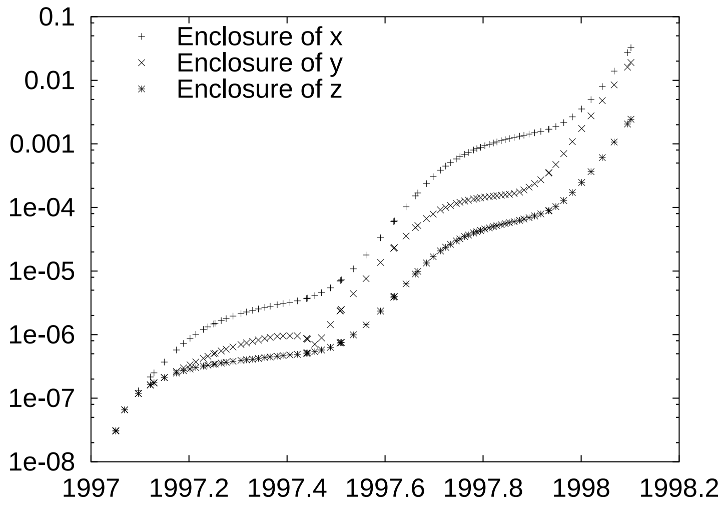


Figure 5.12: Logarithmic plot of the diameters of the enclosures for positions obtained by AWA for an initial box reduced by a factor of 3.275 in each direction.

Since the Taylor model approach is much more computationally complex than regular interval arithmetic, it is natural to ask if we could split the original problem for AWA in such a way that the total CPU time used is the same for both AWA and the Taylor model integration. Using AWA, the computation of the results shown in Fig. 5.11 took 4.5 seconds of CPU time. On the same machine, the Taylor model based integration took more than 92 minutes, or about 1234 times longer. Since

$$(3.275)^6 \approx 1234,$$

we then split the boxes of initial conditions into boxes that are 3.275 times smaller in each coordinate direction and ran AWA with initial conditions given by these smaller boxes. Fig. 5.12 shows the resulting enclosures of the final positions for the same integration period as before. It is striking that even if AWA uses the same amount of CPU time as the Taylor model based integrator, there is hardly any change in the resulting size of the enclosures. Moreover, the sharpness of the results given by AWA is still not comparable with the one computed from Taylor models.

As a final test, we tried to determine the size of initial condition boxes for which

AWA can compute a final enclosure comparable in sharpness to the Taylor model results. However, even the smallest possible initial sets; i.e., boxes with a magnitude of the machine epsilon of 10^{-15} in each of the six coordinate directions, do not allow the computation of final enclosures comparable to the Taylor model results. Details of this integration are shown Fig. 5.13 and it is important to note that even with these tiny initial boxes, the final enclosures are virtually indistinguishable from the ones obtained with larger initial sets. This indicates that the main challenge for the verified integration of this system lies in the wrapping effect, which is known to limit the applicability of conventional interval techniques.

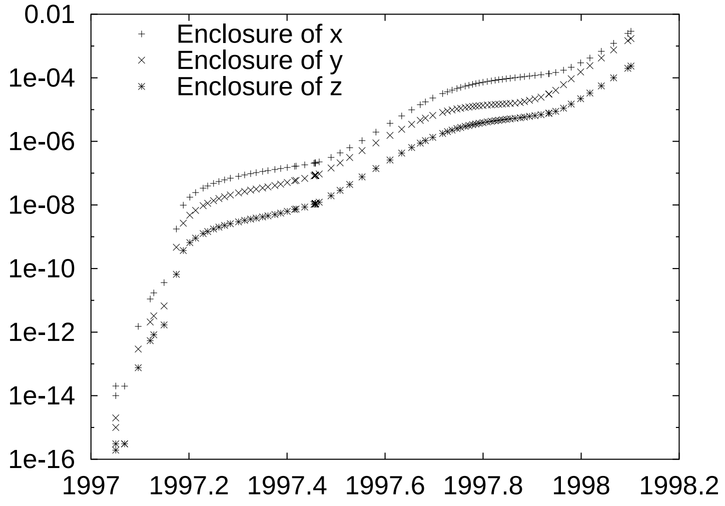


Figure 5.13: Logarithmic plot of the diameters of the enclosures for positions obtained by AWA for initial condition enclosures of 10^{-15}.

While normal interval techniques will almost always outperform Taylor model methods for problems that do not require any domain splitting, it has been shown in Sec. 3.6.1 that if domain splitting becomes necessary, the additional expense of using Taylor models is more than compensated for by the increased accuracy. Here we even have a problem for which the normal interval methods, represented by AWA, cannot compete with Taylor models in terms of sharpness regardless of the amount of domain splitting. Lastly we mention that the integration of a two-body system could have been implemented more favorably for AWA, since the problem can be reduced to four dimensions. However, the main purpose of these computations was to gauge the applicability of conventional interval techniques to the six-dimensional problem of solar system dynamics and not to fabricate a simple test case.

5.1.6 Summary

The results presented in Sec. 5.1.4 and Sec. 5.1.5 show that the Taylor model approach can indeed be used for the verified integration of solar system dynamics. And it has been shown that the Taylor model approach is capable of propagating large sets of initial conditions of large time spans without falling prey to the wrapping effect that often leads to significant overestimations of the final enclosures in verified integrations of ODE initial value problems.

Moreover, by comparing the results from the Taylor model based integration method with similar results obtained from tools utilizing conventional interval techniques, it has to be concluded that the Taylor model approach is the only method that allows the verified long-term integration of asteroid orbits and other real world problems. Compared to conventional interval techniques, the Taylor model approach gives several orders of magnitude sharper results and can propagate initial conditions over orders of magnitude larger time spans.

5.2 Existence of Generating Functions

In this section we discuss a particularly interesting application of Taylor model methods to the theory of generating functions of canonical transformations. We show that using high-order Taylor models enables us to rigorously establish lower bounds on the size of the domains of definition of any type of generating function. Moreover, we will see that the computed domains often enclose the dynamic apertures of the examples studied. The results of this analysis have far reaching implications in the study of particle accelerators and symplectic tracking.

5.2.1 Introduction

It is well known that coordinate transformations of Hamiltonian systems that keep the Hamiltonian structure intact are called canonical transformations, or, in more modern terminology, symplectic maps [17, 49, 123]; the time evolutions of Hamiltonian systems are also symplectic. Any symplectic map can be represented in terms of a single scalar function, called the generating function. Until recently, only the four conventional Goldstein-type generators were well known [49]. However, following the introduction of extended generating functions, it has been shown that to each symplectic map, infinitely many generating function types can be constructed [40]. In certain applications, such as long term simulation of accelerators and other Hamiltonian systems, it is important to maintain symplecticity during tracking [47, 1, 40]; one available method to achieve this is the generating function based symplectic tracking [14, 40, 138].

In principle, any of the valid generators could be used for tracking of a given Hamiltonian system. Although it can be shown that for any globally defined symplectic map, global generators can always be found [40], their construction is however usually not straightforward. Indeed, as shown in [17, 40], the representations of symplectic maps through generators commonly available in practice are often only locally valid. Frequently, the purpose of tracking simulations is to estimate the region of space where stable particle orbits can be found, the so-called non-wandering set, or dynamic aperture in the beam dynamics terminology. Hence, a sizable phase space region must be covered by tracking, and if the generating function method is used, the generating function must be defined at least in that region. However, so far nothing has been known about the size of the domains of definition of generating functions, and the necessity to study this interesting problem has been recognized in [14, 138, 33, 40].

5.2.2 Theory of Extended Generating Functions

Following the exposition of [40], we regard every map as a column vector. Let

$$\alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} \tag{5.13}$$

be a diffeomorphism from the open set $U \subset \mathbb{R}^{4w}$ onto its image; α_1 and α_2 are first and second $2w$ components of α, respectively. In other words, for $i = 1, 2$, $\alpha_i : U \to V_i \subset \mathbb{R}^{2w}$. Let

$$\mathrm{Jac}\,(\alpha) = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \tag{5.14}$$

be the $4w \times 4w$ Jacobian of α, split into $2w \times 2w$ blocks and define

$$\tilde{J}_{4w} = \begin{pmatrix} J_{2w} & 0_{2w} \\ 0_{2w} & -J_{2w} \end{pmatrix}, \tag{5.15}$$

where

$$J_{2w} = \begin{pmatrix} 0_w & I_w \\ -I_w & 0_w \end{pmatrix}. \tag{5.16}$$

In the last equation, I_w denotes the unit matrix of appropriate dimension. We call a map α conformal symplectic if its Jacobian satisfies

$$(\mathrm{Jac}\,(\alpha))^T \cdot J_{4w} \cdot \mathrm{Jac}\,(\alpha) = \mu \cdot \tilde{J}_{4w}, \tag{5.17}$$

with some non-zero real constant μ [6]. Finally, a map $\mathcal{M}$ is called symplectic if its Jacobian M satisfies the symplectic condition [39]

$$M^T \cdot J \cdot M = J. \tag{5.18}$$

Henceforth, we will consider only symplectic maps that are origin preserving; i.e., maps around fixed points. We call a map a gradient map if it has a symmetric Jacobian N, and it is well known that, at least over simply connected subsets, gradient maps can be written as the gradient of a function, which is called the potential of the map [17]. With these preliminaries, the main result of the extended generating function theory is best expressed by the following theorem [40].

Theorem 5.1 (Existence of Extended Generating Functions). *Let $\mathcal{M}$ be a symplectic map and denote the identity map by $\mathcal{I}$. Then for every point z in the domain of $\mathcal{M}$, there is a neighborhood of z such that $\mathcal{M}$ can be represented by functions F via the relation*

$$(\nabla F)^T = \left(\alpha_1 \circ \begin{pmatrix} \mathcal{M} \\ \mathcal{I} \end{pmatrix}\right) \circ \left(\alpha_2 \circ \begin{pmatrix} \mathcal{M} \\ \mathcal{I} \end{pmatrix}\right)^{-1}, \tag{5.19}$$

where $\alpha = (\alpha_1, \alpha_2)^T$ is any conformal symplectic map such that

$$\det\left(C\left(\mathcal{M}\left(z\right), z\right) \cdot Mz + D\left(\mathcal{M}\left(z\right), z\right)\right) \neq 0. \tag{5.20}$$

Conversely, for any $\mathcal{C}^2$ function F the map $\mathcal{M}$ defined by

$$M = (NC - A)^{-1}(B - ND) \tag{5.21}$$

is symplectic, where the matrices $A, B, C, D, M,$ and N are defined as above.

The function F is called the generating function of type α of $\mathcal{M}$, and denoted by $F_{\alpha,\mathcal{M}}$. Theorem 5.1 says that, once the generator type α is fixed, locally there is a one-to-one correspondence between symplectic maps and scalar functions, which are unique up to an additive constant. The constant can be normalized to zero without loss of generality. Due to the fact that there exist uncountably many maps of the form (5.17), to each symplectic map one can construct infinitely many generating function types. But in any case, according to Eqn. (5.19), for a given $\mathcal{M}$ the domain of definition of the generator of type α equals the domain of invertibility of

$$\left(\alpha_2 \circ \begin{pmatrix} \mathcal{M} \\ \mathcal{I} \end{pmatrix}\right). \tag{5.22}$$

Thus, finding the domain of definition of a generator is equivalent to finding the domain of invertibility of (5.22). A large class of practically used generator types are the generators associated with linear maps α. These can be organized into equivalence classes $[S]$, associated with

$$\alpha = \begin{pmatrix} -JL^{-1} & J \\ \frac{1}{2}(I+JS)L^{-1} & \frac{1}{2}(I-JS) \end{pmatrix}, \tag{5.23}$$

and represented by arbitrary symmetric matrices S [40] where the matrix L denotes the linear part of $\mathcal{M}$.

Finally, we note that the size of the domains of definition of the extended generating functions is directly related to the problem of optimal symplectic approximation of Hamiltonian flows [40]. A very general theoretical argumentation based on Hofer's metric states that in general the generating function associated with $S = 0$ gives the optimal approximation. In the examples shown in Sec. 5.2.4 we will therefore concentrate on this particular choice of S.

5.2.3 Enclosing Derivatives of Flows

In order to prove the existence of a generating function over extended domains with Taylor model methods, we need to obtain Taylor models for the derivative of Eqn. (5.22). In the case of linear maps α, this reduces to the computation of Taylor models for the Jacobian of the map $\mathcal{M}$. While simple systems may allow the explicit determination of the symplectic map and its derivative, in the more common case of practical problems, the exactly known information are the Hamiltonian functions, or equivalently, the corresponding ordinary differential equations. Since closed form solutions of these are generally not available, numerical integration schemes are utilized to compute approximations to the mathematically correct solutions. If we use the Taylor model integration method discusses in Sec. 2.3.3, we can enclose the mathematically correct solution by a Taylor model. However, since the derivative operation does not extend to Taylor models, we cannot use this Taylor model to compute Taylor models for the derivates. But the following result from the theory of ordinary differential equations gives us the means to compute Taylor models for the Jacobians of flows [54].

Theorem 5.2 (Smooth Dependence on Initial Conditions). *Suppose that $f(t,x)$ is a $\mathcal{C}^{n+1}$ function defined on an open set $\Omega \subset \mathbb{R} \times \mathbb{R}^v$. Let $\phi(t,u,y)$ be the unique solution to the initial value problem*

$$x' = f(t,x), \quad x(u) = y. \tag{5.24}$$

Then the function $\phi(t,u,y)$ is a $\mathcal{C}^{n+1}$ function of the variables (t,u,y) and the partial derivative of the solution $\frac{\partial\phi}{\partial y}$ with respect to the spatial initial conditions satisfies the matrix initial value problem

$$Y' = \left(\frac{\partial f}{\partial x}\left(t, \phi\left(t,u,y\right)\right)\right) \cdot Y, \quad Y(u) = id. \tag{5.25}$$

Thus, by using Taylor model based integration techniques to rigorously enclose the solution of the enlarged ODE initial value problem

$$x' = f(t,x), \tag{5.26a}$$

$$Y' = \left(\frac{\partial f}{\partial x}\left(t,x\right)\right) \cdot Y, \tag{5.26b}$$

$$x(t_0) = x_0, \tag{5.26c}$$

$$Y(t_0) = I, \tag{5.26d}$$

we can compute Taylor models containing not only the map $\mathcal{M}(t_f, t_0, x_0)$ at the final time t_f, but also the Jacobian $\frac{\partial \mathcal{M}}{\partial x_0}(t_f, t_0, x_0)$ of the flow. In light of Eqn. (5.26), the symplectic condition (5.18) loses its differential algebraic nature and becomes a purely algebraic constraint. Thus, similar to energy conservation, the symplectic condition expresses in fact a conservation law, which can open the door for symplectic integration in the framework of as differential algebraic equations.

It should be noted that the augmented ODE system (5.26) is less smooth than the original problem, resulting in solutions that are also less smooth than before. However, since the systems of interest are commonly analytic over their domains, this is usually not a problem. Finally, it is important to note that while the augmented system is $v(v+1)$ dimensional, there is no need to propagate any dependency on the additional initial conditions in the augmented system. Thus, the integration of the augmented system can still be done with Taylor model in v variables, resulting in a system that is as manageable as the original problem.

5.2.4 Examples

To illustrate how the existence of generating functions can indeed be proved with full mathematical rigor, we will now discuss several examples of symplectic systems. We show that the Taylor model based invertibility tests can often guarantee invertibility of the maps over regions that properly enclose the dynamic aperture.

An Exactly Symplectic Map

As a first example we consider a two dimensional cubic polynomial, which is exactly symplectic and given by

$$\mathcal{M} = \mathcal{N} \circ \mathcal{L}, \tag{5.27}$$

where

$$\mathcal{L} = \begin{pmatrix} \cos\frac{\pi}{3} & \sin\frac{\pi}{3} \\ -\sin\frac{\pi}{3} & \cos\frac{\pi}{3} \end{pmatrix}, \tag{5.28}$$

and

$$\mathcal{N}\begin{pmatrix} q \\ p \end{pmatrix} = \begin{pmatrix} q - 3\,(q+p)^3 \\ p + 3\,(q+p)^3 \end{pmatrix}. \tag{5.29}$$

The main advantage of this relatively simple example over the more complicated examples discussed below is it allows an analytical treatment. According to Thm. 3.2, extended generating function exist over extended regions $\boldsymbol{D} \subset \mathbb{R}^2$ if the following polynomial expression for the determinant in the four variables q_1, p_1, q_2 and p_2 has no zeros in $\boldsymbol{D} \times \boldsymbol{D} \subset \mathbb{R}^4$:

$$\begin{aligned} 1 \quad &+ \quad 9\,\frac{\left[(1-\sqrt{3})\,q_1 + (1+\sqrt{3})\,p_1\right]^2}{8\left(1+\sqrt{3}\right)^2}\left[\left(2+\sqrt{3}\right)(s_{12}+1) + s_{22}\right] \\ &+ \quad 9\,\frac{\left[(1-\sqrt{3})\,q_1 + (1+\sqrt{3})\,p_1\right]^2}{8\left(1+\sqrt{3}\right)^2}\left[\left(2+\sqrt{3}\right)(s_{12}-1) + \left(7+4\sqrt{3}\right)s_{11}\right], \end{aligned} \tag{5.30}$$

where

$$S = \begin{pmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{pmatrix} \tag{5.31}$$

is any symmetric matrix with real entries. It is easy to see that Thm. 3.2 guarantees the existence of globally defined inverses if the entries of S satisfy:

$$-s_{22} \le \left(2+\sqrt{3}\right)(s_{12}+1) \tag{5.32a}$$

$$-s_{11} \le \frac{2+\sqrt{3}}{7+4\sqrt{3}}\,(s_{12}-1)\,. \tag{5.32b}$$

Thus, we can analytically prove the existence of a large class of globally defined generators for this symplectic map. However, while global domains of definition are of great theoretical value, in practical applications we are interested mainly in proving invertibility in finite regions that enclose the dynamic aperture. For these cases, the Taylor model based methods are well suited. For example, employing the Taylor

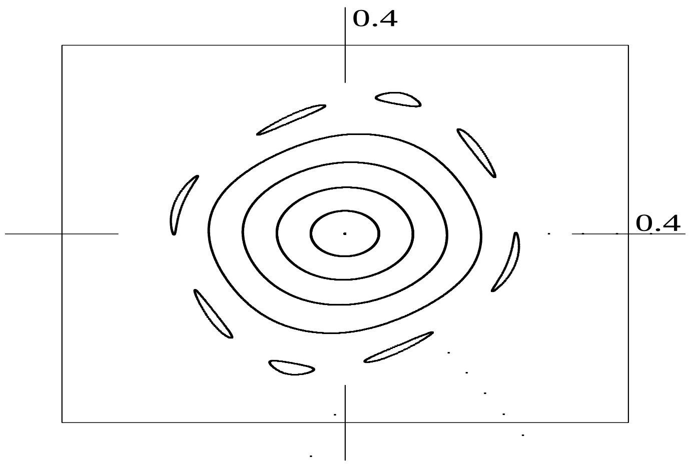


Figure 5.14: Tracking picture of the cubic two-dimensional symplectic map, and the box of guaranteed invertibility of the generator associated with $S = 0$.

method for $S = 0$, the tracking picture shown in Fig. 5.14 was obtained. The box surrounding the tracking is the region for which invertibility of the corresponding generator can be proved using the Taylor model approach. Clearly, it extends well beyond the dynamic aperture, which is approximately 0.2.

The Fermi-Pasta-Ulam System

Since the map of the previous system was given by an exactly symplectic polynomial, we were able to derive an analytic expression for the derivative of the map and compute a Taylor model for the Jacobian. This Taylor model allowed us to apply the invertibility test to prove the existence of the generating function over extended domains. Here and in the next example we study problems in which the maps are obtained by integrating an ODE initial value problem, requiring the application of Thm. 5.2 to obtain Taylor models for the Jacobians of the flows.

The Fermi-Pasta-Ulam system [45] is an extensively studied Hamiltonian system which is known to exhibit many interesting features and has stimulated important developments in nonlinear dynamics. The Fermi-Pasta-Ulam system consists of a finite series of springs; regular springs obeying a linear force law alternating with stiff non-linear springs. The general Hamiltonian function of the system is given by

$$H\left(\vec{q},\vec{p}\right)=\frac{1}{2}\sum_{i=1}^{n}\left(p_{2i-1}^{2}+p_{2i}^{2}\right)+\frac{k}{2}\sum_{i=1}^{n}\left(q_{2i}-q_{2i-1}\right)^{2}+\sum_{i=0}^{n}\left(q_{2i+1}-q_{2i}\right)^{4}. \tag{5.33}$$

For this example we considered the problem with $n=2$, $k=5000$, and $q_4=q_5=p_4=0$, resulting in the system shown in Fig. 5.15.

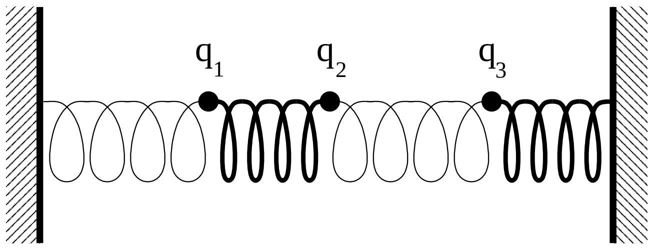


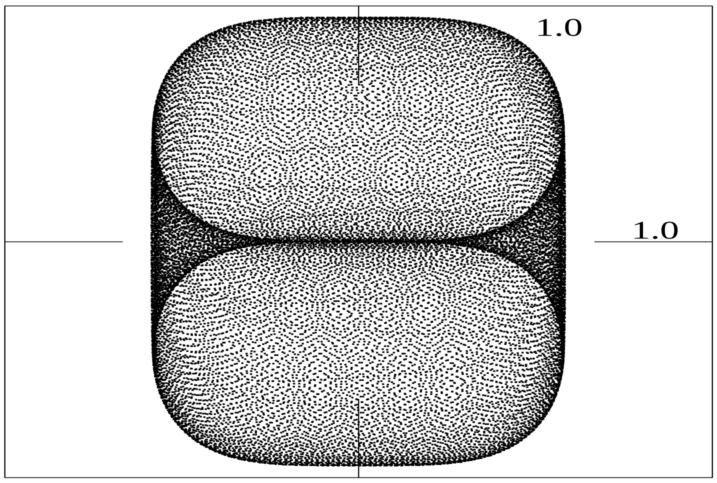
Figure 5.15: Illustration of the Fermi-Pasta-Ulam system with three masses and four springs.

By using the Taylor model based integration scheme, we obtained a Taylor model for the half-period map of the flow of (5.33), and then tracked several particles by iterating this map. Typical phase space plots are shown in Fig. 5.15. The domain of the generator associated with $S=0$ extends to at least $[-1,1]$ in every direction of the phase space. Although the concept of dynamic aperture does not apply in this example, since the system is bounded, the size of the resulting domains are comfortably large.

A Particle Accelerator Cell

We conclude the section with an example of practical interest in accelerator physics. The Hamiltonian representing an accelerator magnet with the arc length s as independent variable and on-energy particles is given by [17]

$$H\left(x,y,a,b\right)=-\left(1+\frac{x}{\rho}\right)\sqrt{1-a^{2}-b^{2}}-\frac{q}{p_{0}}\left(1+\frac{x}{\rho}\right)A_{s}\left(x,y\right). \tag{5.34}$$



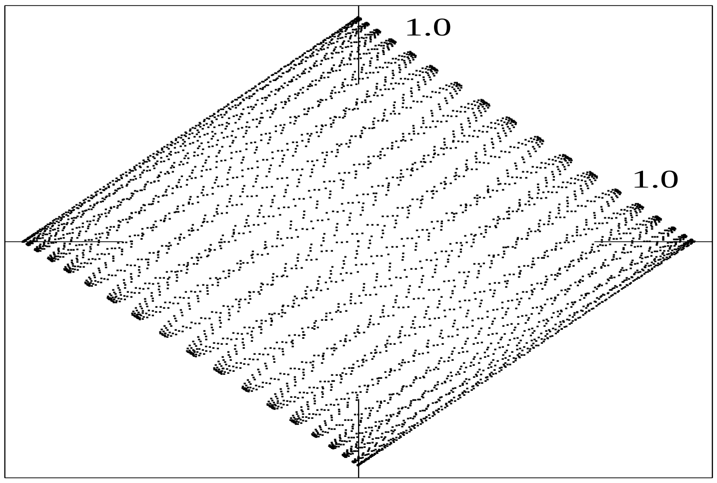


Figure 5.16: (q_1, p_1) and (q_1, q_2) tracking pictures of the half period map of the Fermi-Pasta-Ulam system for a particle launched along the q_1 axis. The box of guaranteed invertibility of the generator associated with $S = 0$ extends to at least $[-1, 1]$ in every direction.

Here ρ is the curvature radius of the magnet, A_s is the s component of the vector potential, and $((x, a)\,; (y, b))$ are two pairs of canonically conjugate variables. For the sake of computational simplicity, we assume that the magnetic fields are piecewise independent of the arc length, in particular we use the sharp cutoff approximation for fringe fields, as is commonly done in beam physics.

To find the Hamiltonian functions for a specific magnetic element or a field free regions of space, all that is necessary are the specific values of ρ and A_s. In field free space is obviously $A_s = 0$ and $\rho = \infty$. For a homogeneous dipole magnet, the vector potential and curvature radius are connected via

$$A_s = -\frac{B_0}{2}(x + \rho), \tag{5.35}$$

where $qB_0/p_0 = 1/\rho$. For a quadrupole magnet on the other hand $\rho = \infty$ and the vector potential is given by

$$A_s = -\frac{k}{2}\left(x^2 - y^2\right), \tag{5.36}$$

where k is the quadruple strength. The case $k > 0$ denotes a magnet that is focusing in the x direction, and correspondingly $k < 0$ indicates a magnet that is defocusing in the x direction and focusing in the y direction. Lastly, the Hamiltonian function of a sextupole magnet is determined by $\rho = \infty$ and

$$A_s = -\frac{h}{3}\left(x^3 - xy^2\right), \tag{5.37}$$

where h is the sextupole strength.

Combining these elements, we set up a cell consisting of the following sequence of elements: drift, defocusing quadrupole superimposed with a sextupole, drift, bending dipole, drift, defocusing quadrupole superimposed with a sextupole, and drift. The defocusing quadrupoles have the same strength $k = -0.0085$ and the sextupole strength is $h = 0.06$. The lengths are 1 meter for the drifts and 0.5 meter for the magnets; the dipole's curvature radius is $\rho = 2.5$ meter and the reference particle is a proton with an energy of 1 MeV. The resulting arrangement is illustrated in Fig. 5.17.

Integration of the system yielded a Taylor model containing the true solution of the corresponding differential equations. For illustration, Tab. 5.7 shows the first ten terms of the Taylor model containing the x-component of the flow, which is the function mapping the initial conditions x_0, a_0, y_0, and b_0 to the x position at the end of the accelerator cell. The full Taylor model is of order 17 and its reference

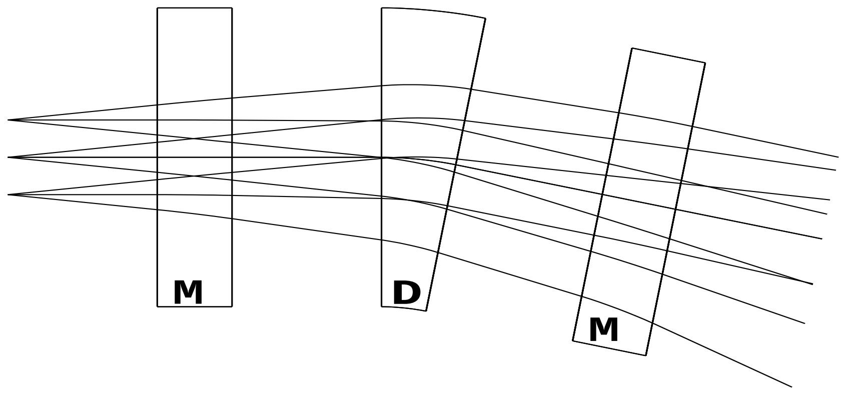


Figure 5.17: Illustration of the accelerator cell studied in Sec. 5.2.4

polynomial has 1933 non-vanishing coefficients. It should be noted that the Taylor model encloses the flow with a relative overestimation of only 10^{-10}, showing yet again that the Taylor model approach avoids the wrapping effect to very high orders and allows the computation of extremely tight solution enclosures.

According to the tracking results shown in Fig. 5.18, the dynamic aperture of this accelerator cell can be estimated to be $(x, y) = (0.03, 0.045)$. Using the Taylor model based invertibility tests with the results of the integration, we have been able to prove the existence of the generating function for $S = 0$ over domains that extend as far as ± 0.1 and therefore comfortably enclose the dynamic aperture of this system. Moreover, since ten centimeters corresponds to a typical accelerator magnet aperture, the results even bear practical relevance.

5.2.5 Rigorous Analysis of Symplectification Errors

Based on the results of summarized in Sec. 5.2.2, generating function based symplectic tracking has been developed for COSY Infinity by B. Erdélyi [40]. The basic ideas of the actual implementation are summarized below. With an arbitrary initial condition

```
RDA VARIABLE:   NO=  17, NV=     4
   I  COEFFICIENT            ORDER EXPONENTS
   1  0.1398389113940111       1   1 0  0 0
   2  0.1038317686361456       1   0 1  0 0
   3  -.2447944264979660E-01   2   2 0  0 0
   4  -.1183394850192213E-01   2   1 1  0 0
   5  -.2119694344941219E-02   2   0 2  0 0
   6  0.2361409770673162E-01   2   0 0  2 0
   7  0.1212766097410308E-01   2   0 0  1 1
   8  0.2185093364668458E-02   2   0 0  0 2
   9  0.9944830763540945E-03   3   3 0  0 0
  10  0.8715481705011164E-03   3   2 1  0 0
---------------------------------------------
VAR    REFERENCE POINT                  DOMAIN INTERVAL
  1  0.000000000000000      [-1.00000000 , 1.00000000 ]
  2  0.000000000000000      [-1.00000000 , 1.00000000 ]
  3  0.000000000000000      [-1.00000000 , 1.00000000 ]
  4  0.000000000000000      [-1.00000000 , 1.00000000 ]
               REMAINDER BOUND INTERVAL
  R     [-.2157121190249145E-012,0.2178948979195422E-012]
*********************************************************
```

Table 5.7: Taylor model describing final x positions as a function of the initial conditions x_0, a_0, y_0, b_0 for the accelerator cell of Sec. 5.2.4.

z in the domain of the generating function, we can write Eqn. (5.19) as

$$\hat{z} - M \cdot z = M \cdot J \cdot (\nabla F)^T \left(C \cdot (\hat{z} - M \cdot z) + z \right), \tag{5.38}$$

where $\hat{z} = \mathcal{M}_{[S]}(z)$ is the image of z under the formally symplectified map $\mathcal{M}_{[S]}$, and

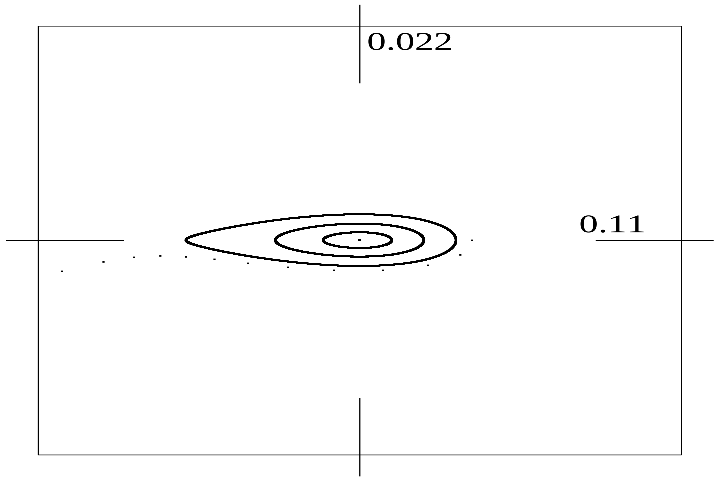
$$C = \frac{1}{2} (I + J \cdot S) \cdot M^{-1}. \tag{5.39}$$

After defining $\hat{z} - M \cdot z$, we write this as

$$w = M \cdot J \cdot (\nabla F)^T (C \cdot w + z). \tag{5.40}$$

The last equation is in fact an implicit relation for $w = w(z)$. Once Eqn. 5.40 has been solved for w, the final result is then computed via

$$\hat{z} = w + M \cdot z. \tag{5.41}$$



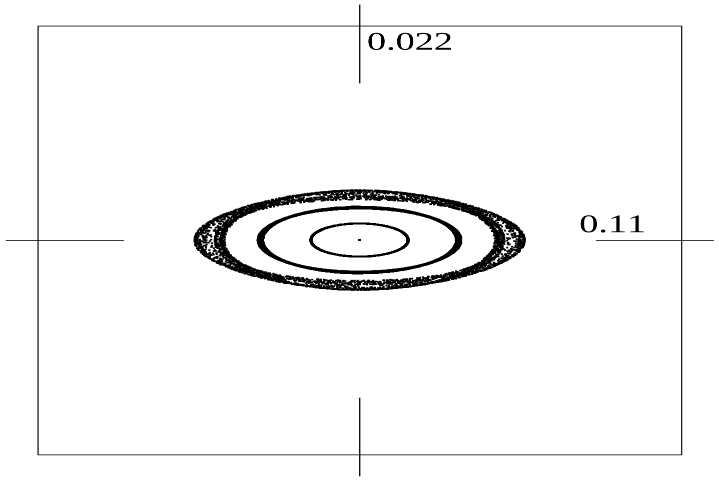


Figure 5.18: (x, a) and (y, b) tracking pictures of the one-turn map of the accelerator cell for particles launched along the spatial x and y axes, respectively. The box of guaranteed invertibility of the generator associated with $S = 0$ extends to at least $[-0.1, 0.1]$ in the spatial variables.

This result is then, up to machine precision, the image of z under the exactly symplectic map $\mathcal{M}_{[S]}$, which is close to the actual map $\mathcal{M}$ of interest. By repeating this procedure, it can be used for the tracking of particles with an exactly symplectic map that is close to the original map of interest.

While this algorithm has been shown to give outstanding results [40], it does not provide any rigorous measures on how close the actual map $\mathcal{M}$ and the symplectified map $\mathcal{M}_{[S]}$ are, since the method does not provide any bounds on the error of the image $\hat{z}$ with respect to the mathematically correct result. However, according to the presented analysis, all quantities determining the implicit relation (5.40) are readily available in the form of Taylor models. By using the Taylor model approach to solving implicit equations presented in Sec. 3.5, we can therefore compute a Taylor model for $w = w(z)$ from (5.40). Using this Taylor model, Eqn. (5.41) describes in fact a Taylor model for $\hat{z}$ in terms of the initial condition z. If we denote this Taylor model by T_2 and the Taylor model for $\mathcal{M}$ computed by integration of the ODEs by T_1, then the Taylor model $\Delta = T_1 - T_2$ gives a rigorous enclosure for the deviation of the symplectified map $\mathcal{M}_{[S]}$ from the mathematically correct map $\mathcal{M}$. The Taylor model Δ can then be used to compute a rigorous upper bound for the error introduced by the symplectification.

Since the Taylor model approach allows the computation of tight enclosures even after extended arithmetic operations, the computed bound on the symplectification error is generally very small in comparison to the estimated dynamic aperture of the Hamiltonian system. Hence, the error analysis can provide a rigorous a posteriori justification for the generating function based symplectic tracking.

5.2.6 Summary

We have shown that the Taylor model approach can indeed be used to rigorously prove that the assumptions of theorems are satisfied. This is of great interest in a variety of settings of theoretical physics and numerical analysis, where sophisticated theorems are often used without any means for an automated analysis of the applicability. Moreover, while a priori or a posteriori analysis can sometimes be used to justify the applicability of methods and theorems, this approach often fails if the input data are obtained as results of previous computations. Here we have seen how the Taylor model approach can be used to simultaneously compute the results and provide a rigorous justification for the applicability of the theory.

In the particular examples studied here, we have proved the existence of generating functions for Hamiltonian systems. Since these generating functions can be used for the symplectic tracking [14, 40, 138] of Hamiltonian systems, the method is of great practical importance in the study of general dynamical systems and especially of particle accelerators.

Chapter 6

Implementational Details

In this final chapter we focus on some of the practical aspects of developing verified numerical methods for real world computer environments. In particular, we discuss the ideas and principles that govern the implementation of the portable interval library in COSY Infinity [26, 20]. The design of this library has been guided by demands for efficiency, portability, and reliability. To that end, the package has been implemented in standard Fortran 77 and does not rely on hardware support for the directed rounding. However, great care has been taken to reduce the unavoidable performance overhead imposed by the additional measures to ensure rigorous enclosures.

Finally, we close with a summary of the least common denominator approach towards language independent software development, and we discuss the C++ and Fortran 90/95 interfaces to COSY Infinity that have been developed based on this paradigm. With the availability of these interfaces, it is now possible to use Taylor model methods within the frameworks of these modern object oriented languages and to interface Taylor model algorithms with existing codes and applications.

6.1 Implementation of Portable Interval Libraries

In the eyes of most computational scientists, floating point numbers are somewhat of a mystery. While floating point support is clearly required for most modern hardware platforms, it was not until 1987 that the IEEE has formally agreed on a standard for the implementation of floating point support [64, 48]

However, to properly implement interval arithmetic on computers, some knowledge about the implementation and the basic layout of floating point numbers is essential. Moreover, to implement *portable* interval arithmetic, a common standard for floating point numbers is required. Still, most of the complications in implementing interval libraries stem from the desire to design packages that are portable between a wide range of hardware and software systems.

6.1.1 Floating Point Numbers

In the following we will assume that floating point numbers are stored in a T-digit, base-B format that complies with the IEEE Standard 754 for floating point numbers [64]:

$$(s) \cdot \left(\sum_{i=1}^{T} x_i \cdot B^{-i} \right) \cdot \left(B^E \right) . \tag{6.1}$$

The number s is an internal representation for the sign of the floating point number, the number E satisfies the condition $E_{\min} \leq E \leq E_{\max}$ with machine dependent integers $E_{\min}$, $E_{\max}$, and T. It should be noted that $E_{\min} < 0$, while $E_{\max} > 0$. With the exception of the number 0, all numbers are normalized such that the leading digit x_1 is non-zero. While the IEEE standard 754 requires B to be two and allows $T = 24$ and $T = 53$, the newer standard IEEE 854 also permits $B = 10$. However, in the remainder of this section we will always assume that all floating point numbers are represented in a fixed binary format; i.e., we assume that $B = 2$ and that the integers

T, $E_{\min}$, and $E_{\max}$ are fixed.

In the following small letters x, y, and z denote real numbers, and their corresponding representations on the computer are denoted by $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$. The elementary operations $+$, $-$, $\times$, and $/$ on the set of floating point numbers are denoted by $\oplus$, $\ominus$, $\otimes$, and $\oslash$, respectively.

The most fundamental and most limiting property of floating point numbers is that only a finite subset of rational numbers has an exact floating point representation. We denote this set of floating point numbers by

$$\mathcal{F} = \left\{ x \in \mathbb{Q} \pm \left(\sum_{i=1}^{T} x_i 2^{-i} \right) \cdot 2^E \, , E_{\min} \le E \le E_{\max} \, , x_1 = 1 \right\} \tag{6.2}$$

It should be noted that within any given floating point system $\mathcal{F}$, some important real numbers have exact floating point representations: $-2, -1, 0, 1, 2 \in \mathcal{F}$. The next proposition highlights another two important floating point numbers which will be used throughout this section.

Proposition 6.1. *For a given binary representation of floating point numbers, define the numbers ε and μ by*

$$\varepsilon = 2^{1-T} \tag{6.3a}$$

$$\mu = 2^{E_{\min}+1}. \tag{6.3b}$$

Then $\varepsilon, \mu \in \mathcal{F}$ and the two numbers can be characterized by

$$\varepsilon \;=\; \min \{ x \,|\, x \in \mathcal{F} \text{ and } \mathbf{1} \oplus \boldsymbol{x} > \mathbf{1} \} \, , \tag{6.4a}$$

$$\mu \;=\; \min \{ x \,|\, x \in \mathcal{F} \text{ and } \boldsymbol{x} > \mathbf{0} \} \, . \tag{6.4b}$$

Proof. To prove that the two numbers have exact floating point representations, we write

$$\varepsilon = 2^{1-T} = 2^1 \cdot 2^{-T}. \tag{6.5}$$

Thus, according to Eqn. (6.2), it is indeed $\varepsilon = \boldsymbol{\varepsilon}$. Similarly, it is

$$\mu = 2^{E_{\min}+1} = 2^1 \cdot 2^{E_{\min}} \tag{6.6}$$

Hence, the number μ also has an exact floating point representation. Next we show that ε is the smallest floating point satisfying the condition $\mathbf{1} \oplus \boldsymbol{x} > \mathbf{1}$:

$$\begin{aligned} \mathbf{1} \oplus \boldsymbol{\varepsilon} &= \left(\mathbf{2^1 \cdot 2^{-1}}\right) \oplus \left(\mathbf{2^1 \cdot 2}^{-\boldsymbol{T}}\right) = \mathbf{2^1} \otimes \left(\mathbf{2^{-1}} + \mathbf{2}^{-\boldsymbol{T}}\right) && \text{(6.7a)} \\ &= 2^1 \cdot \left(2^{-1} + 2^{-T}\right) = 1 + \varepsilon > 1 = \mathbf{1}. && \text{(6.7b)} \end{aligned}$$

On the other hand if we assume that there is $\boldsymbol{y} \in \mathcal{F}$ such that $\boldsymbol{y} < \varepsilon$ and $\boldsymbol{y} \oplus \mathbf{1} > \mathbf{1}$, then $\boldsymbol{y}$ is of the form

$$\boldsymbol{y} = 2^{1-T} \cdot 2^E \cdot \sum_{i=1}^{T} x_i 2^{-i} \tag{6.8}$$

with $E_{\min} + T - 1 \leq E \leq 0$ and $x_1 = 1$. Consequently it would be

$$\begin{aligned} 1 + y &= \left(\mathbf{2^1 \cdot 2^{-1}}\right) \oplus \left(\mathbf{2^1 \cdot 2}^{\boldsymbol{E}-\boldsymbol{T}} \cdot \sum_{i=1}^{T} \boldsymbol{x_i} \mathbf{2}^{-\boldsymbol{i}}\right) && \text{(6.9a)} \\ &= \mathbf{2^1} \cdot \left(\mathbf{2^{-1}} \oplus \sum_{i=1}^{T} \mathbf{2}^{-\boldsymbol{i}-\boldsymbol{T}+\boldsymbol{E}}\right) && \text{(6.9b)} \\ &= 2^1 \cdot 2^{-1} = 1 = \mathbf{1}, && \text{(6.9c)} \end{aligned}$$

in contradiction to the assumption that $\boldsymbol{y} \oplus \mathbf{1} > \mathbf{1}$. Thus, the floating point number ε is indeed characterized by (6.4a).

The last part of the assertion follows from the definition of μ and the basic requirements of floating point representations: clearly $\mu > 0$, and any positive floating point number smaller than μ would have either $E < E_{\min}$ or $x_1 = 0$, both violating the definition of the set $\mathcal{F}$ of floating point numbers. Thus, μ can indeed be characterized by (6.4b), making it the smallest positive floating point number. $\square$

6.1.2 Directed Rounding

In Sec. 2.2 we showed that the sum of two intervals is given by

$$[x_1, x_2] + [y_1, y_2] = [x_1 + x_2, y_1 + y_2]. \tag{6.10}$$

However, if we naively implement this interval addition on a computer with floating point numbers satisfying $T = 53$, the result of the following addition would violate the basic inclusion property of intervals:

$$[-1, 1] + \left[2^{64}, 2^{64}\right]. \tag{6.11}$$

While all interval endpoints have exact floating point representations, the floating point sums needed for the interval addition would lead to truncation:

$$\begin{aligned} \mathbf{-1} \oplus \mathbf{2^{64}} &= \mathbf{2^{64}}, && (6.12a) \\ \mathbf{1} \oplus \mathbf{2^{64}} &= \mathbf{2^{64}}. && (6.12b) \end{aligned}$$

Clearly, this situation is not acceptable in an interval library used for rigorous computations and the necessity of finding a solution to this problem has already been recognized in [92]. The easiest way of alleviating this problem is by using directed rounding for the result of interval computations: at the end of each elementary interval computation, the lower bound of the resulting interval is replaced by a number that is, by an appropriate amount, smaller than the originally computed floating point number. Similarly, the computed result for the upper bound is shifted by an appropriate amount towards $+\infty$. Since floating point numbers are not uniformly spaced, the results cannot be computed by simply adding fixed numbers to the endpoints. Instead, *directed rounding* towards $-\infty$ and $+\infty$ is used.

The most accurate and most efficient way of implementing directed rounding in an interval library would be to rely on intrinsic routines built into IEEE 754 compliant floating point hardware: the standard requires the hardware to provide at least the following three rounding modes:

1. rounding towards $-\infty$,

2. rounding towards $+\infty$,

3. rounding to the nearest floating point number.

With the ability to set the rounding mode, Alg. 6.1 could be used for the addition of two intervals $[x_1, x_2]$.

Algorithm 6.1 Interval addition with hardware support to set the rounding mode.

```
save current rounding mode
set rounding towards $-\infty$
$z_1 \Leftarrow x_1 \oplus y_1$
set rounding towards $+\infty$
$z_2 \Leftarrow x_2 + y_2$
restore saved rounding mode
```

However, this strategy defies the goal of writing a portable interval library, since it would require hardware-specific code to change the rounding modes. Moreover, even on the same platform, different compilers use different conventions of embedding hardware instructions, leading to an unacceptable platform dependence in the resulting code. Finally, it should also be noted that on many current hardware platforms a change in the rounding mode results in a severe performance penalty, since it leads to pipeline flushes that render this approach unacceptably slow [65, 135]. Only recently have interval researchers been able to convince hardware manufacturers of the need for more efficient rounding mode switches, and the resulting interval implementations have shown substantial increases in performance [136, 137].

In order to implement directed rounding in software, it is necessary to know by how many *Units in the Last Place* (ULPs) the computed result of any of the floating point operations differs from the mathematically correct result. We will denote this number by N and note that most of the current computer manufacturers guarantee N to be at most one. Based on the number N, the following theorem allows the use of directed rounding for verified interval results:

Theorem 6.1. *Let $\boldsymbol{x} \in \mathcal{F}$ be given and assume that it is sufficiently large (c.f. Cor. 6.1 for a definition of how large $\boldsymbol{x}$ has to be). Provided that none of the following operations leads to a floating point overflow, the number*

$$\left(\mathbf{1} \oplus \mathbf{sgn}\,(\boldsymbol{x}) \otimes \boldsymbol{N} \otimes \boldsymbol{\varepsilon}\right) \otimes \boldsymbol{x} \tag{6.13}$$

is by at least N ULPs bigger than $\boldsymbol{x}$. Similarly the number

$$\left(\mathbf{1} \ominus \mathbf{sgn}\,(\boldsymbol{x}) \otimes \boldsymbol{N} \otimes \boldsymbol{\varepsilon}\right) \otimes \boldsymbol{x} \tag{6.14}$$

is by at least N ULPs smaller than $\boldsymbol{x}$.

Proof. The assertions will be proved for $N = 1$ and for positive numbers $\boldsymbol{x}$. Similar arguments hold for higher values of N and negative numbers $\boldsymbol{x}$. It is

$$(\mathbf{1} + \boldsymbol{\varepsilon})\boldsymbol{x} = 2^1\left(2^{-1} + 2^{-T}\right) \cdot 2^E \cdot \left(\sum_{i=1}^{T} x_i 2^{-i}\right) \tag{6.15a}$$

$$= 2^{1+E} \cdot \left(2^{-1} + 2^{-T}\right) \cdot \left(\sum_{i=1}^{T} x_i 2^{-i}\right) \tag{6.15b}$$

$$= 2^E \cdot \left(\sum_{i=1}^{T} x_i 2^{-i} + \sum_{i=1}^{T} x_i 2^{-(i+T-1)}\right) \tag{6.15c}$$

$$= \left\lfloor 2^E \cdot \left(\left(\sum_{i=1}^{T-1} x_i 2^{-i} + (x_T + x_1)2^{-T} + \sum_{i=2}^{T} x_i 2^{-(i+T-1)}\right)\right)\right\rfloor \tag{6.15d}$$

$$= 2^E \cdot \left(\sum_{i=1}^{T-1} + (x_T + x_1)2^{-T}\right) \tag{6.15e}$$

Since $x_1 = 1$, the result is indeed by at least one ULP bigger than the original number $\boldsymbol{x}$. A similar computation shows that the second multiplication does indeed round towards $-\infty$:

$$\begin{aligned}
(\mathbf{1} - \boldsymbol{\varepsilon})\boldsymbol{x} &= \boldsymbol{x} \ominus \varepsilon \boldsymbol{x} && \text{(6.16a)}\\
&= 2^E \cdot \left(\sum_{i=1}^{T} x_i 2^{-i}\right) - 2^{1-T} \cdot 2^E \cdot \left(\sum_{i=1}^{T} x_i 2^{-i}\right) && \text{(6.16b)}\\
&= 2^E \cdot \left(\sum_{i=1}^{T} x_i 2^{-i} - \sum_{i=1}^{T} x_i 2^{-(i+T-1)}\right) && \text{(6.16c)}\\
&= \left\lfloor 2^E \cdot \left(\sum_{i=1}^{T-1} x_i 2^{-i} + (x_T - x_1)2^{-T} + \sum_{i=2}^{T} x_i 2^{-(i+T-3)}\right)\right\rfloor && \text{(6.16d)}
\end{aligned}$$

Since $x_1 = 1$, the result is indeed by at least one ULP smaller than the original number $\boldsymbol{x}$, which finishes the proof. □

A Portable Algorithm for Directed Rounding

Based on the results derived in the previous section, we can now present an algorithm that, under mild assumptions on the accuracy of the hardware and system software, allows the computations of verified interval operations with directed rounding. The next two corollaries are immediate consequences of Thm. 6.1 and they provide the mathematical foundation for software implemented directed rounding:

Corollary 6.1. *The number $\delta = \mu / (1 - N\varepsilon)$ is the smallest positive floating point number that can be rounded towards zero without remaining fixed or resulting in an underflow result of* 0.

Corollary 6.2. *Let $x, y \in \mathbb{R}$ be such that* $\min(|x|, |y|) \geq \delta$ *and assume that the following multiplications do not result in a floating point overflows Then the following* *inclusion holds:*

$$[x, y] \subset [(\mathbf{1} - \mathbf{sgn}(\boldsymbol{x})\boldsymbol{N\varepsilon}) \cdot \boldsymbol{x}, (\mathbf{1} + \mathbf{sgn}(\boldsymbol{x})\boldsymbol{N\varepsilon}) \cdot \boldsymbol{x}]\,. \tag{6.17}$$

Corollary 6.2 excludes two special cases that require exception handling in an implementation of an interval library that uses software implemented directed rounding:

- The multiplications might result in overflow errors and since this cannot be handled in a general and portable way, the exception handling is left to the runtime environment.
- Rounding of floating point numbers $\boldsymbol{x}$ with $0 \leq |\boldsymbol{x}| < \delta$ need special attention because a multiplication may result in an underflow error or a fixed point.

From now on we will assume that $N = 1$, and we define the following two numbers:

$$M_u = 1 + \varepsilon, \tag{6.18a}$$

$$M_d = 1 - \varepsilon. \tag{6.18b}$$

With these numbers and the previously defined ε, Alg. 6.2 describes portable software simulated directed rounding: if $[\boldsymbol{x_1}, \boldsymbol{x_2}]$ is a floating point interval, then Alg. 6.2 performs the necessary rounding of that interval.

On average, Alg. 6.2 is too pessimistic since it assumes that all floating point operations have systematic errors, even the ones with exact floating point operands and results. Moreover, both endpoints of the intervals are always rounded outwards, although on average half the computed endpoints are already properly rounded. However, considering the result of Thm. 6.1, the presented algorithm is optimal for software simulated directed rounding. In order to implement Alg. 6.2, it is necessary to know the machine dependent numbers μ and ε, which can be calculated with the algorithms 6.3 and 6.4.

Algorithm 6.2 Directed rounding of the two interval endpoints of $[x_1, x_2]$.

```
**if** $x_1 \geq \delta$ **then**
  $x_1 \Leftarrow x_1 \otimes M_d$
**else if** $x_1 \leq -\delta$ **then**
  $x_1 \Leftarrow x_1 \otimes M_u$
**else if** $x_1 > 0$ **then**
  $x_1 \Leftarrow 0$
**else**
  $x_1 \Leftarrow -\delta$
**end if**

**if** $x_2 \geq \delta$ **then**
  $x_2 \Leftarrow x_2 \otimes M_u$
**else if** $x_2 \leq -\delta$ **then**
  $x_2 \Leftarrow x_2 \otimes M_d$
**else if** $x_2 \geq 0$ **then**
  $x_2 \Leftarrow \delta$
**else**
  $x_2 \Leftarrow 0$
**end if**
```

Algorithm 6.3 Portable determination of the machine constant ε at run time.

```
$\varepsilon \Leftarrow 1$
**while** $1 + \varepsilon/2 > 1$ **do**
  $\varepsilon \Leftarrow \varepsilon/2$
**end while**
```

Algorithm 6.4 Portable determination of the machine constant μ at run time.

```
$\mu \Leftarrow 1$
**while** $\mu/2 > 0$ **do**
  $\mu \Leftarrow \mu/2$
**end while**
```

Thus, by combining Alg. 6.2 with the two algorithms 6.3 and 6.4 we can indeed implement a truly portable interval library with fully rigorous directed rounding. The performance of the combined algorithms will be discussed in Sec. 6.1.4.

6.1.3 Implementation of Interval Operations

In Sec. 2.2 we noted that the intrinsic functions can be extended to intervals in such a way that the basic inclusion property is always maintained:

$$f([a,b]) = \{f(x) \mid a \leq x \leq b\}\,. \tag{6.19}$$

Here we discuss for some of the intrinsic functions how the extension to intervals can be defined such that inclusions are indeed propagated. The interval extensions of the standard mathematical functions are not based on their respective Taylor expansions, but rely on domain decompositions such that the functions are monotonic over the subdomains. The following two proposition show how the monotonicity of certain functions can be used to compute simple and accurate interval extensions for monotone functions:

Proposition 6.2. *For any interval $[a,b] \subset \mathbb{R}$, the interval extension of the exponential function is defined as*

$$\exp\left([a,b]\right) = [\exp(a), \exp(b)] \tag{6.20}$$

and satisfies the basic requirement (6.19).

Proposition 6.3. *If we assume that $a > 0$, then the interval extension of the logarithm is defined as*

$$\log\left([a,b]\right) = [\log(a), \log(b)] \tag{6.21}$$

and satisfies the basic requirement (6.19).

Alg. 6.5 shows an algorithm for extending the sine function to interval arguments $[a_1, a_2]$. Special attention is given to arguments that are too large for an accurate determination of the sine function. However, since the interval $[-1, 1]$ is always a correct, albeit overly pessimistic enclosure for the correct result, even this case can be handled rigorously.

Algorithm 6.5 Reliable interval extension of the sine function.

```
$\pi_2 \Leftarrow 2\pi$
**if** $\max(|a_1|, |a_2|) < 10^6$ **then**
    perform outward rounding on the interval $[a_1, a_2]$ and store the result in $[b_1, b_2]$
    $r_1 \Leftarrow$ nearest integer to $((b_1 \oslash \pi_2) \ominus 1/4) \otimes \pi_2$
    **if** $b_1 > r_1$ **then**
        $r_1 \Leftarrow r_1 \oplus \pi_2$
    **end if**
    $r_2 \Leftarrow$ nearest integer to $((b_2 \oslash \pi_2) \oplus 1/4) \otimes \pi_2$
    **if** $b_2 < r_2$ **then**
        $r_2 \Leftarrow r_2 \ominus \pi_2$
    **end if**
    **if** $b_1 \leq r_2$ and $b_2 \geq r_1$ **then**
        $c_1 \Leftarrow -1$
        $c_2 \Leftarrow +1$
    **else if** $b_1 \leq r_2$ **then**
        $c_1 \Leftarrow \min(\sin(b_1), \sin(b_2))$
        $c_2 \Leftarrow +1$
        round $c_1$ towards $-\infty$
    **else if** $b_2 \geq r_1$ **then**
        $c_1 \Leftarrow -1$
        $c_2 \Leftarrow \max(\sin(b_1), \sin(b_2))$
        round $c_2$ towards $+\infty$
    **else**
        $c_1 \Leftarrow \min(\sin(b_1), \sin(b_2))$
        $c_2 \Leftarrow \max(\sin(b_1), \sin(b_2))$
        perform outward rounding on the interval $[c_1, c_2]$
    **end if**
**else**
    $c_1 \Leftarrow -1$
    $c_2 \Leftarrow +1$
**end if**
return the interval $[c_1, c_2]$
```

6.1.4 Benchmarks and Results

Here we present several benchmark results of the interval implementation in the COSY Infinity [26] language environment, which is based on the previously presented algorithms. The results show that the implementation is very efficient and introduces only a small overhead in the interval operations of COSY Infinity.

We chose the multiplication of intervals as the benchmark operation, since its computational cost is similar to the one of the presented software-based rounding algorithm. For the actual implementation, we have measured the execution times of $9 \cdot 10^6$ interval multiplications that test all nine cases of the interval multiplication. Thus, the results allow a very good comparison between the execution times and the imposed overheads. For additional comparison, we also compared the new code to the previous implementation of the interval routines in COSY Infinity, which did not provide any rounding and were therefore not fully verified. Altogether, we tested the following three cases:

A: The code has been executed with an unmodified version of the original interval routines of COSY Infinity.

B: The code has been executed with the new interval routines, but rounding was disabled.

C: The code has been executed with the new interval routines and outward rounding was enabled.

The results of A and B allow a comparison between the new and the old interval libraries in COSY Infinity. A comparison between B and C on the other hand, puts a measure on the performance overhead imposed by the software rounding.

First we tested the new routines in a pure Fortran 77 program, to eliminate the overhead of the COSY Infinity runtime environment. The program had to be compiled without any optimization, thus the execution times should be viewed relative to each other, rather than in absolute numbers.

Platform, Compiler	*Test A*	*Test B*	*Test C*	*A/B*	*C/B*
VMS/Alpha, Digital F77	9.16 s	4.63 s	13.39 s	1.98	2.89
Linux/Intel, G77	9.75 s	8.44 s	14.77 s	1.16	1.75
Solaris/Sparc, Forte F77	8.29 s	6.22 s	11.03 s	1.33	1.77
Digital Unix/Alpha, Digital F77	2.80 s	1.40 s	3.10 s	2.00	2.21

Table 6.1: Average execution times for the three interval test cases in pure Fortran 77 environments.

Next we computed the interval products in the COSY Infinity language environment and the corresponding execution times are given in the next table. This time we used the highest optimization levels available. Thus the numbers should not be compared to the previous benchmarks.

Platform, Compiler	*Test A*	*Test B*	*Test C*	*A/B*	*C/B*
VMS/Alpha, Digital F77	28.97 s	25.55 s	58.28 s	1.13	2.28
Linux/Intel, G77	24.71 s	26.24 s	31.72 s	0.94	1.21
Solaris/Sparc, Forte F77	17.27 s	18.32 s	19.81 s	0.94	1.08
Digital Unix/Alpha, Digital F77	7.24 s	7.63 s	8.77 s	0.95	1.15

Table 6.2: Average execution times for the three interval test cases in COSY Infinity language environments.

It should be noted that the new interval routines have a different calling scheme in the COSY Infinity language environment than the old interval routines: compared to the case A, B requires one additional subroutine call, while C requires two additional calls. The plain Fortran programs on the other hand have only one additional call in the case of C and the same number of subroutine calls in the cases of A and B. Since the new call graph leads to an improved maintainability of the code, the additional overhead is however acceptable. It should also be noted that the actually computed interval results of A and B are always equal, but unverified. The case C on the other hand produces correctly rounded enclosures for the mathematically correct results in all situations.

6.1.5 Summary

In this section we have presented the basic ideas behind rigorous interval arithmetic implementations. In particular, we introduced the concept of directed rounding, which can be used to eliminate truncation and underflow errors from conventional floating point operations. We also outlined how hardware support can in principle be utilized for outward rounding. However, the main focus of the discussion has been the presentation of a portable algorithm for the rigorous implementation of directed rounding.

To illustrate how the use of directed rounding can avoid the problem illustrated in (6.11), consider the following similar example with directed rounding:

$$\begin{aligned}\left(\left[2^{64},2^{64}\right]+[-1,1]\right)-\left[2^{64},2^{64}\right] &= \left[2^{64}\downarrow,2^{64}\uparrow\right]-\left[2^{64},2^{64}\right] && \text{(6.22a)}\\ &= \left[2^{64}-2^{11},2^{64}+2^{11}\right]-\left[2^{64},2^{64}\right] && \text{(6.22b)}\\ &= \left[-2^{11}-2^{-42},2^{11}+2^{-42}\right]. && \text{(6.22c)}\end{aligned}$$

Although the last interval is a rather pessimistic enclosure of the mathematically correct result of $[-1,1]$, it is nevertheless a guaranteed enclosure. And it is easy to see that the result is in fact the *best* enclosure one can obtain for this problem on a computer with $T = 53$. Generall, overly large final interval enclosures often indicate that the program, although mathematically correct, is stated in a numerically unstable form and should be reformulated.

6.2 Language Independent Programming

Scientific software is commonly developed in high-level programming languages like C [69], C++ [132], and Fortran 90/95 [89]. More recently the use of Java has been discussed for the use in scientific computing. Despite its performance problems, the main advantage of Java lies in its combination of the object oriented programming paradigm with the promise of portability across a wide range of platforms; while almost all other languages have undergone standardization efforts, none has ever fully achieved true portability between systems.

Although all compiled programming languages with support for floating point numbers are more or less equivalent, for the purposes of computational mathematics, the portability of software is an issue of great concern and often guides important design decisions: compilers are not necessarily available all platforms and legacy applications need to be extended. Thus, it is natural to ask for a *language independent* programming model. The obvious approach to this problem is to write software in a *meta-language* and use code generators to derive native language programs from this high-level description. While we are not aware of any large scale system developed in this model, we have been able to adopt a similar approach for the development of the C++ and Fortran 90/95 interfaces to COSY Infinity. The basic ideas, results, and experiences with this approach will be discussed in this section.

6.2.1 The Least Common Denominator Approach

The COSY Infinity language environment offers an object oriented approach to advanced numerical data types. However, access to these data types has traditionally required using the COSY Infinity programming and run time environment. This restriction has often made it difficult to interface COSY Infinity's data types and algorithms with existing software packages, which are likely to be written in compiled languages like C++ and Fortran 90/95. The C++ and Fortran 90/95 interfaces to COSY Infinity offer a solution to this problem: the flexibility of a modern object-oriented language combined with the power of the high performance data types and algorithms of COSY Infinity.

While it can be argued that Fortran 77 this is not the most appropriate language for the development of a large-scale software package – version 8.1 of COSY Infinity consists of more than 35,000 lines of highly optimized source code – the use of Fortran 77 also has undeniable advantages over the deployment of more recent languages like C++, Java, or Fortran 90/95:

Availability Together with C, Fortran 77 is one of the most widely available programming languages. Virtually all computer platforms have a Fortran 77 compiler.

Performance Fortran 77 is a very simple language with low runtime overhead, resulting in fast execution times and outstanding resource utilizations.

Legacy Many existing and well tested software packages are written in Fortran 77, providing programmers with a large pool of reusable software components.

Fortran 77 is in fact the least common denominator of most modern programming languages, since it has no support for operator overloading, dynamic memory allocation, and low level hardware operations like bit manipulation and pointers. Thus, any program written in Fortran 77 can, at least in principle, be mapped one-to-one to languages like C, C++, and Fortran 90/95. Since the Fortran family of programming languages is backwards compatible, this mapping is trivial for the Fortran 90/95 environments. Moreover, with the availability of the F2C converter [43], we also found the appropriate tool for a conversion from Fortran 77 to C and C++ source code.

COSY Infinity is a programming environment that uses a byte-compiler approach similar to Java: source code is translated into an internal byte code, which is then executed by an interpreter. Thus, the COSY Infinity system consists of three main parts: the compiler, the interpreter, and the support libraries implementing all the available language features. Not much would be gained by simply converting Fortran 77 code to C++ source or renaming the Fortran 77 source files into Fortran 90/95 source files. However, both C++ and Fortran 90/95 offer object oriented language features, which allow the definition of new data types, and have been used for the implementations of the interfaces to COSY Infinity. This approach offers:

- native language support via classes (C++) and Modules (Fortran 90/95),
- thin and lightweight object oriented wrappers with low overhead,
- embedding of the COSY interpreter into Fortran 90/95 and C++ programs,
- operator overloading for transparent transition from built-in numerical data types to COSY objects.

The interfaces disregard the byte-compiler part of COSY Infinity and embed the interpreter to gain access to the support libraries of the system. Since all advanced data types of COSY Infinity are implemented as part of the support libraries, they are immediately available to C++ and Fortran 90/95 programmers.

We also mention that the system has been implemented in a way that completely removes the need for any manual code manipulations in any language other than Fortran 77. Thus, at the core of the *least common denominator* (LCD) approach lies a set of Fortran 77 source files that implement all algorithms and data structures and all code for the interfaces is generated from this uniform code base in an automated way. Therefore, the interfaces immediately benefit from any enhancements in the COSY Infinity system. In the remainder of this section we discuss the C++ and the Fortran 90/95 in further details and present their application programming interfaces (APIs).

6.2.2 The C++ Interface to COSY Infinity

The C++ interface is implemented through the Cosy class, which offers access from within C++ to the core of COSY Infinity. This interfacing is achieved by embedding the COSY Infinity execution engine into a C++ class. Since the glue that holds the two systems together is a very thin wrapper of C++ code, the performance of the resulting class is comparable with the performance of COSY Infinity itself and exceeds that of other approaches (c.f. Sec. 6.2.4 for further details).

The COSY Infinity programming language [20] uses an object-oriented approach to programming which centers around the idea of dynamic typing: all Cosy objects have an internal type, which may be real, string, logical, etc., and the exact meaning of operations on Cosy objects is determined at runtime and not at compile time.

The Cosy class attempts to be compatible with the C++ double precision data type. In most cases, it should be possible to convert an existing numerical program to a Cosy-based one by simply replacing the string “double” with the string “Cosy” in the source. However, using this approach would underutilize the Cosy class, which shows its real strengths if the advanced data types like intervals, DA vectors, or Taylor models are used. For example, replacing the double precision numbers in an existing program with Cosy objects that are initialized to DA vectors would allow high-order sensitivity analysis of the original program. Other benefits lie in the automatic verification of existing programs by using intervals or Taylor models.

Memory Management

The Cosy class manages its own internal memory and does not use dynamic allocation of memory by either malloc or new. To a large extent, this is the reason for the performance advantage that COSY Infinity has over languages like C and C++.

As a consequence of this, every Cosy object requires a small portion of space in some non-dynamic memory region. While this is never an issue with global and local variables, this becomes an issue when Cosy objects are created dynamically by using `new` or `new[]`. Consequently, *dynamic allocation of Cosy objects should be avoided* whenever possible. If Cosy objects really have to be created dynamically, care should be taken to delete the objects as soon as possible, or the COSY system will exhaust its internal memory.

Constructors

To allow an easy conversion of existing code from the double data type to the Cosy data type, several constructors have been defined that should accommodate this through a variety of implicit constructions. Together with the built-in type conversions of C++, this mechanism should be able to handle almost any situation correctly.

`Cosy( );`

The default constructor creates a Cosy object with enough internal space to store one number or character. The object's type is initialized to RE (real) and its value is set to zero.

`Cosy( const double val, int len = 1 );`

Create a Cosy object with enough internal space to hold `len` numbers or characters. The parameter `len` is optional and defaults to 1. The object's type is initialized to

RE and its value is set to `val`.

`Cosy( const int val, int len = 1 );`

Create a Cosy object with enough internal space to store `len` numbers or characters. The parameter `len` is optional and defaults to 1. The type of the object is initialized to RE, since COSY Infinity does not have a dedicated data type for integers; its value is set to `val`.

`Cosy( const bool f );`

Create a Cosy object with enough internal space to store one number or character. The object's type is initialized to LO (logical or boolean) and its value is set to the boolean value `f`.

`Cosy( const char *str );`

Create a Cosy object from a C string `str`. The object's type is set to ST (string) and enough internal memory locations are allocated to hold the string (without the terminating NULL character, which is not needed in COSY). The object is initialized with the string `str`.

`Cosy( const Cosy& src );`

Create a new Cosy object from an existing one. The new object is initialized with a deep copy of `src`.

`Cosy( integer len, const int n, const int dim[] );`

This special constructor creates a Cosy object that represents a Cosy array of dimensionality `n`. The length of each of the dimensions is given in the array `dim`. And each entry of the array has internal space for `len` numbers and is initialized to zero with type RE. For further details on Cosy arrays, refer to Sec. 6.2.2.

Assignment Operators

The Cosy class supports all assignment operations available in C++. Moreover, all the assignment operations that are commonly used with floating point numbers are implemented in a way compatible with the standard C++ definitions for floating point data types.

`Cosy& operator =(const Cosy& rhs)`

Assign a deep copy of `rhs` to the object and return a reference to it.

`Cosy& operator+=(const Cosy& rhs)`

Add `rhs` to the object and return a reference to it; equivalent to $x = x + rhs$.

`Cosy& operator-=(const Cosy& rhs)`

Subtract `rhs` from the object and return a reference to it; equivalent to $x = x - rhs$.

`Cosy& operator*=(const Cosy& rhs)`

Multiply the object with `rhs` and return a reference to it; equivalent to $x = x * rhs$.

`Cosy& operator/=(const Cosy& rhs)`

Divide the object by `rhs` and return a reference to it; equivalent to $x = x/rhs$.

`Cosy& operator&=(const Cosy& rhs)`

Unite the object with `rhs` and return a reference to it. For numerical Cosy objects, the result of a union is usually a vector. It should be noted that this implementation of this operator is not compatible with the default behavior of this operator in C++.

Unary Mathematical Operators

The Cosy class supports all unary operators available in C++. The operators are compatible with the default implementations for floating point variables.

`Cosy operator+()`

Return the positive of the object. This is in fact an identity operation and is included only for completeness.

`Cosy operator-()`

Return the negative of the object without modifying it.

`Cosy operator++()`

Add one to the object and return the result.

`Cosy operator--()`

Subtract one from the object and return the result.

`Cosy operator++(int)`

Add one to the object and return a copy of the object before the operation.

`Cosy operator--(int)`

Subtract one from the object and return a copy of the object before the operation.

Array Access

`Cosy get(const int coeff[], const int n)`

Obtain a copy of an array element. The element is described by the n-dimensional array coeff. More details on Cosy arrays are provided in Sec. 6.2.2.

`void set(const Cosy& arg, const int coeff[],const int n)`

Copy the Cosy object arg into an array. The target element is described by the n-dimensional array coeff. More details on Cosy arrays are provided in Sec. 6.2.2.

Printing, IO, and Streams

As indicated earlier, the code for the Cosy class is automatically derived from Fortran 77 code by using the F2C [43] converter. Consequently, the IO handling of the underlying C code is conceptually closer to the printf-type ideas of C than it is to the streams of C++.

However, by using temporary files, the Cosy class has partial support for the stream based IO of C++. This mechanism uses the file COSY.TMP in the current working directory as a translation buffer. This allows the Cosy class to be compatible with output streams.

```
friend ostream& operator<<(ostream& s, const Cosy& src)
```

Print a representation of the object src onto the ostream s. The printing uses the formats specified in [20].

Type Conversion

While the implicit type conversion mechanisms of C++ allow a transparent transition from the default C++ data types to Cosy objects. The conversion of Cosy objects into standard C++ data types on the other hand requires use of the dedicated conversion functions listed below.

```
friend double toDouble(const Cosy& arg)
```

Return a double precision variable that represents the result of calling the function CONS on the Cosy object arg.

```
friend bool   toBool  (const Cosy& arg)
```

Return a boolean variable that contains the boolean value of the Cosy object arg. If arg is not of type LO, the return value is undefined.

`friend string toString(const Cosy& arg)`

Return a C++ string object that contains the string contained in the Cosy object arg. If arg is not of type ST, the result is undefined.

Elementary Operations and Functions

The COSY Infinity environment has a large number of operators and functions built into its language. The C++ interface to COSY Infinity aims to give transparent access to these functions by trying to be compatible with both the notations of C++ and of COSY Infinity. To that end, the operators are compatible with the C++ notations, and the elementary functions are compatible with the standard C++ naming conventions, and almost all functions defined in *math.h* for double precision floating point numbers are supported for Cosy objects.

As a general rule, all functions in C++ are named with the lower case version of their corresponding COSY Infinity identifier. However, whenever COSY Infinity uses a name for a function that does not exist in C++, e.g.,the absolute value function is called *abs* in COSY Infinity, while it should be called *fabs* in C++, both names are made available. Whenever the name of a COSY function clashes with reserved words of C++, the first letter of that function's name is capitalized: e.g., the COSY Infinity function REAL is called *Real* in the C++ interface.

For the operators defined in COSY Infinity, the following deviations from these general rules exist:

- While the exponentiation is an operation in COSY Infinity, C++ uses the function `pow(...)` for this.
- The operator `#` of COSY Infinity is not defined in C++ and has been replaced with the standard C++ operator `!=`.

- The operator `&` does not follow the standard C++ conventions and computes the union of two Cosy objects. However, since the Cosy class is meant to be used for the development of new programs, or as a replacement for double variables, overloading this operator is unlikely to cause any problems.

The signatures of all operators and functions derived from the COSY operators are:

```
Cosy operator+
Cosy operator-
Cosy operator*
Cosy operator/
Cosy pow
bool operator<
bool operator>
bool operator==
bool operator!=
Cosy operator&
bool operator<=
bool operator>=
```

The standard functions defined for the Cosy class are listed in Appendix B.1. These functions are also referred to as *intrinsic functions* for Cosy objects.

COSY Procedures

The COSY Infinity language environment has several procedures built into its language. These procedures range from diagnostic tools (e.g., MEMFRE) over file handling to complex tasks (e.g., POLVAL). For a complete interface from C++ to COSY Infinity it was necessary to make these procedures available as *void functions.* The C++ interfaces to the procedures all have a standardized signature:

```
void <name> (...);
```

All procedures take at least one argument, and all arguments are either of type `Cosy &` or `const Cosy &`. The list in Appendix B.2 shows the names of the COSY Infinity procedures in the first column, and the declaration of the corresponding C++

functions. Further details on the corresponding COSY Infinity procedures are given in [20].

Cosy Arrays vs. Arrays of Cosy Objects

In the COSY Infinity language environment, arrays are collections of objects that may or may not have the same internal type. Thus, within COSY Infinity, it is conceivable to have an array with entries representing strings, intervals, and real numbers. In that sense, the notion of arrays in COSY Infinity is quite similar to the notion of arrays of Cosy objects in C++.

However, there is a fundamental difference between the two concepts: a C++ array of Cosy objects is not a Cosy object. Due to this difference, the C++ interface does not use C++ arrays of Cosy objects (although the user obviously has the freedom to declare and use them). As a consequence, the interface provides two different (and slightly incompatible) notions of arrays. *Arrays of Cosy Objects* are C++ arrays and they can be used wherever C++ permits the use of arrays. *Cosy Arrays*, on the other hand, are individual Cosy objects which themselves contain Cosy objects. Since several important procedures of COSY Infinity assume their arguments to be Cosy arrays, Cosy arrays are quite important in the context of COSY Infinity and its C++ interface.

Since the C++ interface to Cosy does not use the `[]` operator for the access to elements, users should use the utility functions

```
Cosy get(const int coeff[], const int n)
```

and

```
void set(const Cosy& arg, const int coeff[], const int n)
```

described in Sec. 6.2.2 to access the elements of a Cosy array. To simplify the access to individual array elements, we suggest that users use inheritance or external utility functions for convenient access to the elements of Cosy arrays. For two-dimensional arrays, such a function could be imlemented as:

```
Cosy get(Cosy &a, int i, int j) {
    int c[2] = {i+1, j+1};
    return a.get(c, 2);
}
```

Since Cosy arrays start at one, (as opposed to C++ arrays that start at 0), these utility functions could also be used to mask this implementational detail from the user. However, since the user's requirements on the dimensionality of Cosy arrays vary widely, the distribution of the C++ interface does not provide any of these convenience functions.

Finally, we point out that the two different concepts of arrays lead to the possibility of having C++ arrays of Cosy arrays — although it might be challenging to maintain a clear distinction between the various indices needed to access the individual elements.

6.2.3 The Fortran 90/95 Interface to COSY Infinity

The Fortran 90/95 interface to COSY Infinity gives Fortran 90/95 programmers easy access to the sophisticated data types of COSY Infinity. The interface has been implemented in the form of a Fortran 90/95 module, which has recently been used as part of the GlobSol [46] system for constrained global optimization.

Special Utility Routines

The Fortran 90/95 interface to COSY Infinity uses a small number of utility routines for low-level access to the internals. In this section we describe these routines in detail.

`SUBROUTINE COSY_INIT [<NTEMP>] [<NSCR>] [<MEMDBG>]`

Initialize the COSY system. This subroutine has to be called before any COSY objects are used.

`NTEMP` sets the size of the pool of temporary objects and defaults to 20. This pool of variables is used for the allocation of temporary COSY objects. Since Fortran 90/95 does not support automatic destruction of objects, it is necessary to allocate all temporary objects beforehand and never deallocate them during the execution of the program. The pool is organized as a circular list; and in the absence of automatic destruction of objects, if the number of actually used temporary variables ever exceeds `NTEMP`, memory corruption will occur. It is the responsibility of the user to set the size appropriately.

`NSCR` defaults to 5000 and sets the size of the variables in the pool. Additionally, the subroutine SCRLEN is called to set the size of COSY's internal temp variables. `MEMDBG` may be either 0 (no debug output) or 1 (print debug information on memory usage). It should never be necessary for users of the Fortran 90/95 module to set `MEMDBG`.

Neither the size of the pool, nor the size of the variables in the pool can be changed after this call. More details on the pool of temporary objects are provided in Sec. 6.2.3.

`SUBROUTINE COSY_CREATE <SELF> [<LEN>] [<VAL>] [<NDIMS>] [<DIMS>]`

Create a variable in the cosy core. All COSY objects have to be created before they can be used! This routine allocates space for the variable and registers it with the COSY system. `SELF` is the COSY variable to be created.

`LEN` is the desired size of the variable `SELF`, which determines how many DOUBLE PRECISION values can be stored in `SELF`, and defaults to 1. If `VAL` is given, the variable is initialized to it; otherwise `VAL` defaults to zero. Independent of the parameters `LEN` and `VAL`, the type of the variable is set to RE.

This routine can also be used for the creation of COSY arrays, which are discussed in Sec. 6.2.3. If `NDIMS` and `DIMS` are specified, the variable `SELF` is initialized to be an `NDIMS`-dimensional COSY array with length `DIMS(I)` in the i-th direction. Each entry of the array has length `LEN` and is initialized to `VAL` with type RE.

`SUBROUTINE COSY_DESTROY <SELF>`

Destruct the COSY object `SELF` and free the associated memory. If `SELF` hasn't been initialized with `COSY_CREATE`, the results are undefined.

`SUBROUTINE COSY_ARRAYGET <SELF> <NDIMS> <IDXS>`

Return a copy of an element of the array `SELF`. `NDIMS` specifies the dimensionality of the array and `IDXS` is an array containing the index of the desired element. More details on COSY arrays are provided in Sec. 6.2.3.

`SUBROUTINE COSY_ARRAYSET <SELF> <NDIMS> <IDXS> <ARG>`

Copy the COSY object `ARG` into an element of the `NDIMS`-dimensional array `SELF`. The target is specified by the `NDIMS`-dimensional array `IDXS` which contains the index of the target. More details on COSY arrays are provided in Sec. 6.2.3.

`SUBROUTINE COSY_GETTEMP <SELF>`

Return the address of the next available temporary object from the circular buffer of such objects. While the value of the returned variable is undefined, the type is guaranteed to be RE. Refer to Sec. 6.2.3 for more details.

`SUBROUTINE COSY_DOUBLE <SELF>`

Extracts the DOUBLE PRECISION value from the variable `SELF` by calling the function COSY function CONS.

SUBROUTINE COSY_LOGICAL <SELF>

Extracts the logical value from the variable SELF. If the type of SELF is not LO, the result is undefined.

SUBROUTINE COSY_WRITE <SELF> [<IUNIT>]

Writes the COSY variable SELF to the unit IUNIT, which defaults to 6. This function uses the same algorithms employed by the COSY procedure WRITE, which is discussed in [20].

SUBROUTINE COSY_TMP <ARG>

Return a temporary COSY object initialized with the value ARG, which may be either of type DOUBLE PRECISION or INTEGER. The main purpose of this function is for the temporary conversion of parameters to COSY procedures. As an example, consider the following two equivalent code fragments. They illustrate that the use of the function COSY_TMP leads to simpler and less error prone code:

```
TYPE(COSY) :: A,B,X
CALL COSY_CREATE(A)
CALL COSY_CREATE(B)
CALL COSY_CREATE(X,2)
A = 2
B = 5
CALL INTERV(A,B,X)
CALL COSY_DESTROY(A)
CALL COSY_DESTROY(B)

TYPE(COSY) :: X
CALL COSY_CREATE(X,2)
CALL INTERV(COSY_TMP(2),COSY_TMP(5),X)
```

Operators

The Fortran 90/95 interface to COSY Infinity offers all operators that the standard COSY system offers. For the convenience of of the user, additional support functions are provided that allow mixed operations between built-in data types and the COSY objects and all operations behave as expected. A complete list of all the defined operations between COSY objects and built-in types is given in Appendix C. It should be noted that all operations involving COSY objects return COSY objects.

Assignment

The Fortran 90/95 interface to COSY Infinity provides several assignment operations that allow an easy transition between built-in data types and COSY objects. This section lists all the defined assignment operators involving COSY objects.

`COSY LHS = COSY RHS`

Copies the COSY object `RHS` to `LHS`. If `LHS` hasn't been created yet, it will be created automatically.

`DOUBLE PRECISION LHS = COSY RHS`

Converts the COSY object `RHS` to the DOUBLE PRECISION number `LHS` by calling the function `COSY_DOUBLE`.

`LOGICAL LHS = COSY RHS`

Converts the COSY object `RHS` to the LOGICAL variable `LHS` by calling the function `COSY_LOGICAL`.

`COSY LHS = DOUBLE PRECISION RHS`

Copies the DOUBLE PRECISION variable `RHS` to the COSY object `LHS`. If `LHS` hasn't been created yet, it will be created automatically. The type of `LHS` will be set to RE.

`COSY LHS = LOGICAL RHS`

Copies the LOGICAL variable `RHS` to the COSY object `LHS`. If `LHS` hasn't been created yet, it will be created automatically. The type of `LHS` will be set to LO.

`COSY LHS = INTEGER RHS`

Copies the INTEGER variable `RHS` to the COSY object `LHS`. If `LHS` hasn't been created yet, it will be created automatically. The type of `LHS` will be set to RE.

Functions

The Fortran 90/95 interface to COSY Infinity supports most of the functions supported by the COSY language environment. The following list shows all the functions that are supported for COSY objects by the Fortran 90/95 interface to COSY Infinity. The left column shows the Fortran 90/95 name of the function and the right column shows the name of the function in the COSY Infinity environment. All functions have the exact same names as the corresponding COSY infinity functions discussed in [20].

Subroutines

All the standard procedures of the COSY Infinity language environment are available as subroutines from the Fortran 90/95 interface to COSY. The names and parameter lists of the subroutines match the names and parameter lists of the normal COSY Infinity procedures.

Automatic argument conversion is not available. That means that all arguments have to be either previously created COSY objects or temporary COSY objects obtained from calls to `COSY_TMP`.

Memory Management

The COSY Fortran 90/95 module is based on the standard core functions and algorithms of COSY Infinity. As such, it uses the fixed size memory buffers of COSY Infinity for storage of COSY objects. While this fact is mostly hidden from the user, understanding this concept helps in writing efficient code.

When a COSY object is created by using the routine `COSY_CREATE`, memory is allocate in the internal COSY memory. This memory is not freed until the routine `COSY_DESTROY` is called for this object. Moreover, since COSY's internal memory is stack based (and not garbage collected), memory occupied by one object will not be freed until all objects that have been created at a later time have also been destroyed.

Since Fortran 90/95 does not have automatic constructors and destructors, all objects have to be deleted manually. While this is generally acceptable for normal objects, this is impossible to guarantee for temporary objects. To allow temporary objects in the COSY module, a circular buffer of temp. objects is created when the COSY system is initialized with `COSY_INIT`.

As an example on how the pool of temporary objects should be used, consider the following fragment of code that implements a convenience interface to the COSY procedure RERAN. Internally, the function `CRAN` obtains one object from the pool for its return value. This avoids the obvious memory leak that would result if it was creating a new COSY object.

```
FUNCTION CRAN()
  USE COSY_MODULE
  IMPLICIT NONE
  TYPE(COSY) :: CRAN
  CALL COSY_GETTEMP(CRAN)
  CALL RERAN(CRAN)
END FUNCTION CRAN
```

However, it has to be stressed that the fixed size of the pool of temporaries bears a potential problem: there is no check in place for possible exhaustion of the pool. In other words, the pool has to be sized large enough to accommodate the maximum number of temp. objects at any given time during the execution of the program. Since this number is easily underestimated, especially for deeply nested expressions, the buffer should be sized rather generously.

COSY Arrays vs. Arrays of COSY objects

In the COSY Infinity language environment, arrays are collections of objects that may or may not have the same internal type. Thus, within COSY Infinity, it is conceivable to have an array with entries representing strings, intervals, and real numbers. In that sense, the notion of arrays in COSY Infinity is quite similar to the notion of arrays of COSY objects in Fortran 90/95.

However, there is a fundamental difference between the two concepts: a Fortran 90/95 array of COSY objects is not again a COSY object. Due to this difference, the Fortran 90/95 module does not use Fortran arrays of COSY objects (although the user obviously has the freedom to declare and use them). As a consequence, the interface provides two different (and slightly incompatible) notions of arrays. *Arrays of COSY Objects* are Fortran 90/95 arrays and they can be used wherever Fortran permits the use of arrays. *COSY Arrays* on the other hand, are individual COSY objects which themselves contain COSY objects. Since several important procedures of COSY Infinity assume their arguments to be COSY arrays, COSY arrays are quite important in the context of COSY Infinity and its Fortran 90/95 interface modules.

To access the elements of COSY arrays, users should use the utility routines

```
SUBROUTINE COSY_ARRAYGET <SELF> <NDIMS> <IDXS>
```

and

```
SUBROUTINE COSY_ARRAYSET <SELF> <NDIMS> <IDXS> <ARG>
```

Finally, we point out that the two different concepts of arrays lead to the possibility of having Fortran 90/95 arrays of COSY arrays – although it would be quite challenging to maintain a clear distinction between the various indices needed to access the individual elements.

6.2.4 Performance Analysis

To demonstrate the success of the LCD approach in the creation of C++ and Fortran 90/95 interfaces to COSY Infinity, Tab. 6.3 lists the execution times of a moderately complicated DA algorithm in six variables and order ten on various hardware/compiler combinations. We have timed the exact same algorithm with the different interfaces and it is important to note that:

a) the Fortran 90/95 interface is not significantly slower than the standard COSY Infinity system

b) the C++ interface is only about two to three times slower than the corresponding Fortran code

While the C++ interface seems slow, most of the computational overhead can be attributed to the implementation of the Cosy class that is optimized for maintainability and not performance. Moreover, to the best of our knowledge all other DA package implemented in C++ are at least ten times slower than COSY Infinity. Thus, the C++ interface to COSY Infinity is in fact the fastest general purpose DA package available; and gives users from all areas of the computational sciences access to advanced Taylor model algorithms and high-order verification.

	COSY, G77	*G++*	*COSY, Fort77*	*Fort90*
Alpha/666, Linux	1:26	4:15	1:11	1:09
	1:17	4:18	1:09	1:10
Intel PII/333, Linux	5:36	10:31		
	5:51	10:31		
	5:54	10:21		
Alpha, True64			1:53	1:54
			1:53	1:54

Table 6.3: Execution times in minutes and seconds for a typical mix of DA operations on different platforms with different interfaces to COSY Infinity.

6.2.5 Summary

In this section we have presented the least common denominator (LCD) approach to language independent software development. We have demonstrated its applicability by summarizing its utilization in the development of C++ and Fortran 90/95 interfaces to COSY Infinity. While we refrain from recommending the LCD approach with Fortran 77 code in new software projects, our experiences show that the method can be used for the maintenance of large existing legacy codes that cannot easily be rewritten in new programming languages. However, it has proven invaluable in the design and development of the C++ and Fortran 90/95 interfaces to COSY Infinity, broadening the applicability and availability of advanced Taylor model methods.

APPENDIX

Appendix A

Orbital Elements of Major Planets

For the computation of asteroid orbits presented in Sec. 5.1, the orbital elements of the nine major planets play an important role: they are the basis of the computation of the planets' positions and velocities. For reference purposes, the following tables list the orbital elements and their linear time dependences. It should however be noted that approximating the time dependence of the orbital elements by linear relations is not sufficient to reproduce the accuracy of the ephemeris DE405 [130].

Tab. A.1 lists the orbital elements of the nine major planets; Tab. A.2 and A.3 list the daily rates of change of the orbital elements. The numerical values are listed in [125] and are accurate at the epoch J2000. While the number of significant digits is reduced in the following tables, the computations have been performed with the full accuracy available.

Planet	Ω	i	ω	a	ε	M
Mercury	48.33167	7.00487	29.12478	0.38710	0.20563	174.79439
Venus	76.68069	3.39471	54.85229	0.72333	0.00677	50.44675
Earth	−11.26064	0.00005	114.20783	1.00000	0.01671	357.51716
Mars	49.57854	1.85061	286.46230	1.52366	0.09341	19.41248
Jupiter	100.55615	1.30530	−85.80230	5.20336	0.04839	19.65053
Saturn	113.71504	2.48446	−21.28310	9.53707	0.05415	317.51238
Uranus	74.22988	0.76986	96.73436	19.19126	0.04717	142.26794
Neptune	131.72169	1.76917	−86.75034	30.06896	0.00859	259.90868
Pluto	110.30347	17.14175	113.76329	39.48169	0.24881	14.86205

Table A.1: Orbital Elements of the major planets (angles in degree, distances in astronomical units) at the epoch J2000.

Planet	Ω'	i'	ω'
Mercury	$-0.3394174461 \cdot 10^{-5}$	$-0.1787968669 \cdot 10^{-6}$	$0.7756255234 \cdot 10^{-5}$
Venus	$-0.7581489085 \cdot 10^{-5}$	$-0.2175070345 \cdot 10^{-7}$	$0.6754049719 \cdot 10^{-5}$
Earth	$-0.1386284128 \cdot 10^{-3}$	$-0.3569853221 \cdot 10^{-6}$	$0.1477415013 \cdot 10^{-3}$
Mars	$-0.7758688874 \cdot 10^{-5}$	$-0.1937029431 \cdot 10^{-6}$	$0.1962864091 \cdot 10^{-4}$
Jupiter	$0.9256749562 \cdot 10^{-5}$	$-0.3156133568 \cdot 10^{-7}$	$-0.2868963411 \cdot 10^{-5}$
Saturn	$-0.1210015971 \cdot 10^{-4}$	$0.4646741210 \cdot 10^{-7}$	$-0.2721423688 \cdot 10^{-5}$
Uranus	$0.1278728420 \cdot 10^{-4}$	$-0.1589474479 \cdot 10^{-7}$	$-0.2805080227 \cdot 10^{-5}$
Neptune	$-0.1150277600 \cdot 10^{-5}$	$-0.2768271345 \cdot 10^{-7}$	$-0.5271731676 \cdot 10^{-5}$
Pluto	$-0.2838999240 \cdot 10^{-6}$	$0.8418891347 \cdot 10^{-7}$	$-0.7218799621 \cdot 10^{-6}$

Table A.2: Daily rates of change of the orbital elements Ω, i, and ω of the nine major planets .

Planet	a'	ε'	M'
Mercury	$0.1806982342 \cdot 10^{-10}$	$0.6918549067 \cdot 10^{-9}$	4.092334433949377
Venus	$0.2518818487 \cdot 10^{-10}$	$-0.1351950719 \cdot 10^{-8}$	1.602131301695948
Earth	$-0.1368904989 \cdot 10^{-11}$	$-0.1041478438 \cdot 10^{-8}$	0.985599987451460
Mars	$-0.1977002118 \cdot 10^{-08}$	$0.3258590009 \cdot 10^{-8}$	0.524021165107646
Jupiter	$0.1662888405 \cdot 10^{-07}$	$-0.3526351815 \cdot 10^{-8}$	0.083080374325026
Saturn	$-0.8255441486 \cdot 10^{-07}$	$-0.1006488706 \cdot 10^{-7}$	0.033485450148293
Uranus	$0.4162217593 \cdot 10^{-07}$	$-0.5242984255 \cdot 10^{-8}$	0.011721311354533
Neptune	$-0.3427679829 \cdot 10^{-07}$	$0.6882956878 \cdot 10^{-9}$	0.005987479199909
Pluto	$-0.2105736030 \cdot 10^{-07}$	$0.1770020547 \cdot 10^{-8}$	0.003976577306242

Table A.3: Daily rates of change of the orbital elements a, ε, and M of the nine major planets.

Appendix B

The COSY C++ Interface

In this section we describe the intrinsic functions and procedures of COSY Infinity that have been exported to the C++ interface. The definitive reference for these information is given by [20].

B.1 Available COSY Functions

The following table lists all COSY Infinity functions that are available from the C++ interface. The first column lists the name of the COSY function as described in [20] and the second column shows the complete signature of the corresponding C++ function.

TYPE	`Cosy type  ( const Cosy& x );`
LENGTH	`Cosy length( const Cosy& x );`
VARMEM	`Cosy varmem( const Cosy& x );`
VARPOI	`Cosy varpoi( const Cosy& x );`
NOT	`Cosy not   ( const Cosy& x );`
EXP	`Cosy exp   ( const Cosy& x );`
LOG	`Cosy log   ( const Cosy& x );`
SIN	`Cosy sin   ( const Cosy& x );`
COS	`Cosy cos   ( const Cosy& x );`
TAN	`Cosy tan   ( const Cosy& x );`
ASIN	`Cosy asin  ( const Cosy& x );`
ACOS	`Cosy acos  ( const Cosy& x );`
ATAN	`Cosy atan  ( const Cosy& x );`
SINH	`Cosy sinh  ( const Cosy& x );`
COSH	`Cosy cosh  ( const Cosy& x );`
TANH	`Cosy tanh  ( const Cosy& x );`
SQRT	`Cosy sqrt  ( const Cosy& x );`
ISRT	`Cosy isrt  ( const Cosy& x );`
SQR	`Cosy sqr   ( const Cosy& x );`

ERF	`Cosy erf   ( const Cosy& x );`
WERF	`Cosy werf  ( const Cosy& x );`
ABS	`Cosy fabs  ( const Cosy& x );`
ABS	`Cosy abs   ( const Cosy& x );`
NORM	`Cosy norm  ( const Cosy& x );`
CONS	`Cosy cons  ( const Cosy& x );`
RE	`Cosy re    ( const Cosy& x );`
IN	`Cosy in    ( const Cosy& x );`
WIDTH	`Cosy width ( const Cosy& x );`
REAL	`Cosy Real  ( const Cosy& x );`
IMAG	`Cosy Imag  ( const Cosy& x );`
INT	`Cosy Int   ( const Cosy& x );`
NINT	`Cosy rint  ( const Cosy& x );`
NINT	`Cosy nint  ( const Cosy& x );`
DA	`Cosy da    ( const Cosy& x );`
CMPLX	`Cosy cmplx ( const Cosy& x );`
CONJ	`Cosy conj  ( const Cosy& x );`

B.2 Available COSY Procedures

The following table lists all COSY Infinity procedures that are available from the C++ interface. The first column lists the name of the COSY procedure as described in [20], while the second column shows the complete signature of the coresponding C++ function (for readability, `c` denotes a `const Cosy &` argument, while `v` stands for `Cosy &` arguments).

MEMALL	`void memall ( v );`
MEMFRE	`void memfre ( v );`
OPENF	`void openf  ( c, c, c );`
CLOSEF	`void closef ( c );`
REWF	`void rewf   ( c );`
BACKF	`void backf  ( c );`
CPUSEC	`void cpusec ( v );`
QUIT	`void quit   ( c );`
SCRLEN	`void scrlen ( c );`
DAINI	`void daini  ( c, c, c, v );`
DANOT	`void danot  ( c );`
DAEPS	`void daeps  ( c );`
DAPEW	`void dapew  ( c, c, c, c );`
DAREA	`void darea  ( c, v, c );`
DAPRV	`void daprv  ( c, c, c, c, c );`
DAREV	`void darev  ( v, c, c, c, c );`
DAFLO	`void daflo  ( c, c, v, c );`

CDFLO	`void cdflo  ( c, c, v, c );`
RERAN	`void reran  ( v );`
DARAN	`void daran  ( v, c );`
DADIU	`void dadiu  ( c, v, v );`
DADER	`void dader  ( c, v, v );`
DAINT	`void daint  ( c, v, v );`
DAPLU	`void daplu  ( v, c, c, v );`
DAPEE	`void dapee  ( c, c, v );`
DAPEA	`void dapea  ( c, c, c, v );`
DAPEP	`void dapep  ( v, v, v, v );`
DANOW	`void danow  ( c, c, v );`
CDF2	`void cdf2   ( v, v, v, v, v );`
CDNF	`void cdnf   ( v, v, v, v, v, v, v, v );`
CDNFDA	`void cdnfda ( v, v, v, v, v, v, v );`
CDNFDS	`void cdnfds ( v, v, v, v, v, v, v );`
MTREE	`void mtree  ( v, v, v, v, v, v, v );`
LINV	`void linv   ( c, v, c, c, v );`
LDET	`void ldet   ( c, c, c, v );`
MBLOCK	`void mblock ( c, v, v, c, c );`
SUBSTR	`void substr ( c, c, c, v );`
STCRE	`void stcre  ( c, v );`
RECST	`void recst  ( c, c, v );`
VELSET	`void velset ( v, c, c );`
VELGET	`void velget ( c, c, v );`
VEZERO	`void vezero ( v, v, v );`
VELMAX	`void velmax ( c, v );`
VEFILL	`void vefill ( v, c, c, c, c );`
IMUNIT	`void imunit ( v );`
LTRUE	`void ltrue  ( v );`
LFALSE	`void lfalse ( v );`
INTERV	`void interv ( c, c, v );`
INSRND	`void insrnd ( c );`
INLO	`void inlo   ( c, v );`
INUP	`void inup   ( c, v );`
IVVELO	`void ivvelo ( c, v );`
IVVEUP	`void ivveup ( c, v );`
IVSET	`void ivset  ( v, c, c );`
IVGET	`void ivget  ( c, c, v );`
OIELEM	`void oielem ( c, v );`
OIIN	`void oiin   ( c, c, v );`
OVELEM	`void ovelem ( c, v );`
OVIV	`void oviv   ( c, c, v );`
INTPOL	`void intpol ( v, c );`
RDAVAR	`void rdavar ( c, c, c, c, c, c, v );`
RDVAR	`void rdvar  ( c, c, c, v );`

RDANOT	`void rdanot ( c, c, v );`
RDREA	`void rdrea  ( c, v );`
DAEXT	`void daext  ( c, v );`
RDRBND	`void rdrbnd ( c, v );`
RDAREF	`void rdaref ( c, v );`
RDADOM	`void rdadom ( c, v );`
RDITIG	`void rditig ( c );`
RDNPNT	`void rdnpnt ( c );`
RDINT	`void rdint  ( c, c, v );`
RDPRIS	`void rdpris ( c );`
CLEAR	`void clear  ( v );`
GRMOVE	`void grmove ( c, c, c, v );`
GRDRAW	`void grdraw ( c, c, c, v );`
GRDOT	`void grdot  ( c, c, c, v );`
GRCURV	`void grcurv ( c, c, c, c, c, c, c, c, c, v );`
GRPROJ	`void grproj ( c, c, v );`
GRCHAR	`void grchar ( c, v );`
GRCOLR	`void grcolr ( c, v );`
GRWDTH	`void grwdth ( c, v );`
GRMIMA	`void grmima ( c, v, v, v, v, v, v );`
RKCO	`void rkco   ( v, v, v, v, v );`
POLVAL	`void polval ( c, c, c, c, c, v, c );`
POLSET	`void polset ( c );`
FOXDPR	`void foxdpr ( c, c );`
MEMWRT	`void memwrt ( c );`
VARMAX	`void varmax ( v );`

Appendix C

The COSY Fortran 90/95 Interface

In this section we describe the operations that are defined between COSY objects and the standard Fortran 90/95 data types. The definitive reference for these information is given by [20]. It should be noted that all operations involving COSY objects return COSY objects.

Addition +		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY

Table C.1: Defined combinations of COSY objects and standard data types for the addition.

Subtraction –		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY
	COSY	COSY

Table C.2: Defined combinations of COSY objects and standard data types for the subtraction.

Multiplication *		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY

Table C.3: Defined combinations of COSY objects and standard data types for the multiplication.

Division /		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY

Table C.4: Defined combinations of COSY objects and standard data types for the division.

Power **		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY

Table C.5: Defined combinations of COSY objects and standard data types for the power operation.

Comparison `.LT.`		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY

Table C.6: Defined combinations of COSY objects and standard data types for the comparison `.LT.`.

Comparison `.GT.`		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY

Table C.7: Defined combinations of COSY objects and standard data types for the comparison `.GT.`.

Comparison `.EQ.`		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY

Table C.8: Defined combinations of COSY objects and standard data types for the comparison `.EQ.`.

Comparison `.NE.`		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY

Table C.9: Defined combinations of COSY objects and standard data types for the comparison `.NE.`.

Concatenation `.UN.`		
COSY	COSY	COSY
DOUBLE PRECISION	COSY	COSY
COSY	DOUBLE PRECISION	COSY
INTEGER	COSY	COSY
COSY	INTEGER	COSY
DOUBLE PRECISION	DOUBLE PRECISION	COSY
DOUBLE PRECISION	INTEGER	COSY
INTEGER	DOUBLE PRECISION	COSY
INTEGER	INTEGER	COSY

Table C.10: Defined combinations of COSY objects and standard data types for the COSY concatenation `.UN.`.

BIBLIOGRAPHY

Bibliography

[1] D. Abell. *Analytic properties and approximation of transfer maps for Hamiltonian systems*. Ph.D. thesis, University of Maryland, 1995.

[2] R. Abraham, J. E. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*, volume 75 of *Applied Mathematical Sciences*. Springer Verlag, second edition, 1998.

[3] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, 1983. ISBN 0-12-049820-0.

[4] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.

[5] G. V. Balaji and J. D. Seader. Application of interval Newton methods to chemical engineering problems. *Reliable Computing*, 1(3):215–223, 1995.

[6] A. Banyaga. *The structure of classical diffeomorphism groups*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1997.

[7] R. Bellmann and R. E. Kalaba, editors. *Analytical and Numerical Methods of Celestial Mechanics*. Modern Analytic and Computational Methods in Science and Mathematics. American Elsevier Publishing Company, New York, 1967.

[8] B. Bertotti, editor. *Experimental Gravitation*. Academic Press, New York, 1974. Proceedings of the International School of Physics "Enrico Fermi" – Course LVI.

[9] Dimitri P. Bertsekas. *Linear Network Optimization*. MIT Press, Massachusetts and London, 1991.

[10] M. Berz. The method of power series tracking for the mathematical description of beam dynamics. *Nuclear Instruments and Methods*, A258:431, 1987.

[11] M. Berz. *The Description of Particle Accelerators using High Order Perturbation Theory on Maps, in: M. Month (Ed), Physics of Particle Accelerators*, volume 1, page 961. American Institute of Physics, 1989.

[12] M. Berz. Differential algebraic description of beam dynamics to very high orders. *Particle Accelerators*, 24:109, 1989.

[13] M. Berz. Forward algorithms for high orders and many variables. *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, SIAM, 1991.

[14] M. Berz. Symplectic tracking in circular accelerators with high order maps. In *Nonlinear Problems in Future Particle Accelerators*, page 288. World Scientific, 1991.

[15] M. Berz. Modern map methods for charged particle optics. *Nuclear Instruments and Methods*, 363:100, 1995.

[16] M. Berz. Differential algebras with remainder and rigorous proofs of long-term stability. In *Fourth Computational Accelerator Physics Conference*, volume 391, page 221. AIP Conference Proceedings, 1996.

[17] M. Berz. *Modern Map Methods in Particle Beam Physics*. Academic Press, San Diego, 1999. ISBN 0-12-014750-5.

[18] M. Berz. Constructive generation and verification of Lyapunov functions around fixed points of nonlinear dynamical systems. *International Journal of Computer Research*, 2001.

[19] M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors. *Computational Differentiation: Techniques, Applications, and Tools*. SIAM, Philadelphia, 1996.

[20] M. Berz and J. Hoefkens. COSY INFINITY Version 8.1 Programming Manual. Technical Report MSUCL-1196, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI 48824, 2001.

[21] M. Berz and G. Hoffstätter. Exact bounds of the long term stability of weakly nonlinear systems applied to the design of large storage rings. *Interval Computations*, 2:68–89, 1994.

[22] M. Berz and G. Hoffstätter. Computation and application of Taylor polynomials with interval remainder bounds. *Reliable Computing*, 4(1):83–97, 1998.

[23] M. Berz, G. Hoffstätter, W. Wan, K. Shamseddine, and K. Makino. COSY INFINITY and its applications to nonlinear dynamics. In Berz et al. [19], pages 363–365.

[24] M. Berz and K. Makino. Verified integration of ODEs and flows with differential algebraic methods on Taylor models. *Reliable Computing*, 4(4):361–369, 1998.

[25] M. Berz and K. Makino. New methods for high-dimensional verified quadrature. *Reliable Computing*, 5(1):13–22, 1999.

[26] M. Berz and K. Makino. COSY INFINITY Version 8.1 reference manual. Technical Report MSUCL-1195, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI 48824, 2001.

[27] M. Berz, K. Makino, and J. Hoefkens. Verified integration of dynamics in the solar system. *Nonlinear Analysis: Theory, Methods & Applications*, 47:179–190, 2001.

[28] Martin Berz. Practical elimination of the wrapping effect in validated solutions of ODEs by the Taylor model approach. *BIT*, 41, 2001. Supplement Issue.

[29] I. M. Bomze, T. Csendes, R. Horst, and P. M. Pardalos, editors. *Developments in Global Optimization*. Kluwer, 1997. ISBN 0-7923-4351-4.

[30] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. North-Holland, 1989.

[31] Y. F. Chang and G. F. Corliss. Solving ordinary differential equations using Taylor series. *ACM Trans. Math. Software*, 8:114–144, 1982.

[32] Y. F. Chang and G. F. Corliss. ATOMFT: Solving ODEs and DAEs using Taylor series. *Computers and Mathematics with Applications*, 28:209–233, 1994.

[33] P. J. Channell. Hamiltonian suspensions of symplectomorphisms: an alternative approach to design problems. *Physica D*, 127 (3-4):117–130, 1999.

[34] G. F. Corliss. Private communication.

[35] G. F. Corliss. Guaranteed error bounds for ordinary differential equations. In W. A. Light and M. Marletta, editors, *Theory of Numerics in Ordinary and Partial Differential Equations*, volume IV, pages 1–75. Oxford University Press, London, 1995.

[36] G. F. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors. *Automatic Differentiation: From Simulation to Optimization*. Springer Verlag, New York, 2001.

[37] G. F. Corliss and R. B. Kearfott. Rigorous global search: Industrial applications. In T. Csendes, editor, *Developments in Reliable Computing*, pages 1–16. Kluwer, 2000.

[38] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Otimization and Nonlinear Equations*. SIAM, Philadelphia, 1996.

[39] A. J. Dragt. Lectures on nonlinear orbit dynamics. In *1981 Fermilab Summer School*. AIP Conference Proceedings Vol. 87, 1982.

[40] B. Erdélyi. *Symplectic Approximation of Hamiltonian Flows and Accurate Simulation of Fringe Field Effects*. Ph.D. thesis, Michigan State University, East Lansing, Michigan, USA, 2001.

[41] J. Eriksson. *Parallel Global Optimization Using Interval Analysis, Licentiate Thesis*. Ph.D. thesis, University of Umeå, Sweden, 1991.

[42] Y. G. Evtushenko. Automatic differentiation viewed from optimal control theory. *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, SIAM:25–30, 1991.

[43] S. I. Feldman, David M. Gay, Mark W. Maimone, and N. L. Schreyer. A Fortran-to-C converter. Technical report, AT&T Bell Laboratories, Murray Hill, NJ 07974, 1995.

[44] V. Fock. *The Theory of Space Time and Gravitation*. Pergamon Press, 1959.

[45] J. Ford. The Fermi-Pasta-Ulam problem: Paradox turns discovery. *Physics Reports*, 213(5):271–310, May 1992.

[46] F. Fritz, P. Thalacker, G. F. Corliss, and R. B. Kearfott. Globsol user guide. Technical Report, Department of Mathematics, Statistics and Computer Science, Marquette University, Milwaukee, Wisc., 1998.

[47] I. Gjaja, A. J. Dragt, and D. T. Abell. A comparison of methods for long-term tracking using symplectic maps. *IOP Conf. Ser.*, 131:173–184, 1993.

[48] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, Mar 1991.

[49] H. Goldstein. *Classical Mechanics*. Addison-Wesley, Reading, MA, 1980.

[50] C. Grebogi, S. M. Hammel, J. A. Yorke, and T. Sauer. Shadowing of physical trajectories in chaotic dynamics. *Physical review Letters*, 65(13):1527–1530, 1990.

[51] E. Griepentrog and R. März. *Differential-algebraic equations and their numerical treatment*. Teubner, 1986.

[52] A. Griewank. On automatic differentiation. Technical Report MCS-P10-1088, Argonne National Laboratory, 1988.

[53] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer Verlag, New York, 1991.

[54] J. K. Hale. *Ordinary Differential Equations*. Krieger Publishing Company, Malabar, Florida, second edition, 1980. Originally published by Wiley-Interscience in 1969.

[55] E. R. Hansen. *An Overview of Global Optimization Using Interval Analysis*, pages 289–307. Academic Press, New York, 1988.

[56] E. R. Hansen. *Global Optimization using Interval Analysis*. Marcel Dekker, 1992.

[57] E. R. Hansen and R. Greenberg. An interval newton method. *Applied Mathematics and Computations*, 12:89–98, 1983.

[58] J. Hoefkens and M. Berz. Differential algebraic methods in feedforward control theory. In *Proceedings of Control Applications of Optimization – CAO'2000*. 2000.

[59] J. Hoefkens, M. Berz, and K. Makino. Efficient high-order methods for ODEs and DAEs. In Corliss et al. [36], pages 343–350.

[60] J. Hoefkens, M. Berz, and K. Makino. Verified high-order integration of DAEs and higher-order ODEs. In Kraemer and v. Gudenberg [71], pages 281–292.

[61] J. Hohnerkamp and H. Römer. *Klassische Theoretische Physik*. Springer Verlag, Berlin, third edition, 1993.

[62] O. Holzmann, B. Lang, and H. Schütt. Newton's constant of gravitation and verified numerical quadrature. *Reliable Computing*, 2(3):229–240, 1996.

[63] K. Ichida and Y. Fujii. An interval arithmetic method for global optimization. *Computing*, 23:85–97, 1979.

[64] IEEE. IEEE standard 754-1985 for binary floating-point arithmetic. Technical report, IEEE, 1987. Reprinted in *SIGPLAN 22*, 2, 9–25.

[65] K. R. Jackson. Private communication.

[66] E. W. Kaucher and W. L. Miranker. *Self-Validating Numerics for Function Space Problems: Computation with guarantees for differential and integral equations*. Academic Press, New York, 1984. ISBN 0-12-402020-8.

[67] R. B. Kearfott. An interval branch and bound algorithm for bound constrained optimization problems. *Journal of Global Optimization*, 2:259–280, 1992.

[68] R. B. Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer, 1996. ISBN 0-7923-4238-0.

[69] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, Jun 1988. ISBN 0131103628.

[70] L. J. Kohout and I. Stabile. Interval-valued inference in medical knowledge-based system CLINAID. *Interval Computations*, 3:88–115, 1993.

[71] W. Kraemer and J. W. v. Gudenberg, editors. *Scientific Computing, Validated Numerics and Interval Methods*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2001.

[72] P. Kunkel. Augmented systems for generalized turning points. In Seydel et al. [126], pages 231–236.

[73] Jet Propulsion Laboratory. Solar system dynamics group, 2001. See also http://ssd.jpl.nasa.gov/.

[74] O. E. Lanford. Computer-assisted proof of the Feigenbaum conjecture. *Bulletin Americal Mathematical Society*, 6:427–434, 1982.

[75] O. E. Lanford. Computer-assisted proofs in analysis. *Physica A*, 124:465–470, 1984.

[76] T. Levi-Civita. *The n-Body Problem in General Relativity*. D. Reidel, Dordrecht, Netherlands, 1964.

[77] J. L. Lions. Ariane 5, flight 501 failure. Technical report, European Space Agency, 1996.

[78] R. Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. Ph.D. thesis, Universität Karlsruhe, 1988.

[79] R. J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. In Edgar W. Kaucher, Ulrich W. Kulisch, and Christian Ullrich, editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, pages 255–286. Wiley-Teubner Series in Computer Science, Stuttgart, 1987.

[80] R. J. Lohner. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In J. R. Cash and I. Gladwel, editors, *Computational Ordinary Differential Equations*, pages 425–435. Clarendon Press, Oxford, 1992.

[81] K. Makino. Rigorous integration of maps and long-term stability. In *1997 Particle Accelerator Conference*. APS, 1997.

[82] K. Makino. *Rigorous Analysis of Nonlinear Motion in Particle Accelerators*. Ph.D. thesis, Michigan State University, East Lansing, Michigan, USA, 1998. Also MSUCL-1093.

[83] K. Makino and M. Berz. Remainder differential algebras and their applications. In Berz et al. [19], pages 63–74.

[84] K. Makino and M. Berz. COSY INFINITY version 8. *Nuclear Instruments and Methods*, A427:338–343, 1999.

[85] K. Makino and M. Berz. Efficient control of the dependency problem based on Taylor model methods. *Reliable Computing*, 5(1):3–12, 1999.

[86] K. Makino and M. Berz. Advances in verified integration of ODEs. *SCAN2000*, 2000.

[87] K. Makino and M. Berz. Pertubative equations of motion and differential operators in nonplanar curvilinear coordinates. *International Journal of Applied Mathematics*, 3(4):421–440, 2000.

[88] K. Makino and M. Berz. Globabal optimzation with Taylor models. *International Journal of Computer Research*, 2001.

[89] M. Metcalf and J. K. Reid. *Fortran 90/95 Explained*. Oxford University Press, second edition, Aug 1999. ISBN 0198505582.

[90] R. E. Moore. Private communication.

[91] R. E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. Ph.D. thesis, Stanford University, Oct 1962.

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

[93] R. E. Moore. *Mathematical Elements of Scientific Computing*. Holt, Rinehart and Winston, 1975.

[94] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, PA, 1979. ISBN 0-89871-161-4.

[95] R. E. Moore and H. Ratschek. Inclusion functions and global optimization II. *Mathematical Programming*, 41:341–356, 1988.

[96] S. P. Mudur and P. A. Koparkar. Interval methods for processing geometric objects. *IEEE Comput. Graphics and Appl.*, 4(2):7–17, Feb 1984.

[97] N. S. Nedialkov. *Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation*. Ph.D. thesis, University of Toronto, 1999.

[98] N. S. Nedialkov and K. R. Jackson. A new perspective on the wrapping effect in interval methods for IVPs for ODEs. *SCAN2000*, 2000.

[99] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. & Comp.*, 105(1):21–68, 1999.

[100] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[101] A. Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. *Computing (Supplement)*, 9:175–190, 1993.

[102] K. Okumura and S. Higashino. A method for solving complex linear equations of AC networks by interval computation. In *Proceedings of IEEE International Symposium on Circuits and Systems – ISCAS '94*, pages 121–124. IEEE, New York, 1994.

[103] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Computer Science and Applied Mathematics. Academic Press, New York and London, 1970.

[104] I. Ozaki, F. Kimura, and M. Berz. A new approach to higher-order sensitivity analysis for optimal mechanical design. *Computational Mechanics*, 16:223, 1994.

[105] C. C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988.

[106] J. D. Pinter. *Global Optimization in Action: Continuous and Lipschitz Optimization*. Kluwer, Dordrecht, Netherlands, 1995.

[107] J. D. Pryce. Solving high-index DAEs by Taylor series. *Numerical Algorithms*, 19:195–211, 1998.

[108] J. D. Pryce. A simple structural analysis method for DAEs. *BIT*, 41(2):364–394, 2001.

[109] P. J. Rabier and W. Rheinboldt. A general existence and uniqueness theorem for implicit differential algebraic equations. *Differential Integral Equations*, 4:563–582, 1991.

[110] P. J. Rabier and W. Rheinboldt. A geometric treatment of differential-algebraic equations. *Journal of Differential Equations*, 109:109–146, 1994.

[111] L. B. Rall. *Automatic Differentiation: Techniques and Applications*. Springer-Verlag, Berlin; Heidelberg; New York, 1981.

[112] L. B. Rall and G. F. Corliss. An introduction to automatic differentiation. In Berz et al. [19], pages 1–18.

[113] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Ellis Horwood Limited, Chichester, England, 1984.

[114] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization*. Ellis Horwood Limited, Chichester, England, 1988.

[115] D. Ratz. *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*. Ph.D. thesis, Universität Karlsruhe, Institut für Angewandte Mathematik, 1992.

[116] Gunther Reißig, Wade S. Martinson, and Paul I. Barton. Differential-algebraic equations of index 1 may have an arbitrarily high structural index. *SIAM Journal on Scientific Computing*, 21(6):1987–1990 (electronic), 2000.

[117] P. J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, New Mexico, 1998.

[118] A. E. Roy. *The Foundation of Astrodynamics*. Crowell-Collier and MacMillan, New York, 1965.

[119] S. M. Rump. Private communication.

[120] S. M. Rump. Validated solution of large linear systems. In R. Albrecht, G. Alefeld, and H. J. Stetter, editors, *Computing Supplement 9*, pages 191–212. Springer Verlag, Wien, 1993.

[121] S. M. Rump. Verification methods for dense and sparse systems of equations. In J. Herzberger, editor, *Topics in Validated Computations*, pages 63–135. North-Holland, Amsterdam, 1994.

[122] S. M. Rump. Ill-conditioned matrices are componentwise near to singularity. *SIAM Review*, 41(1):102–112, 1999.

[123] J. M. Sanz-Serna and M. P. Calvo. *Numerical Hamiltonian Problems*. Chapman and Hall, 1994.

[124] T. Sauer and J. A. Yorke. Rigorous verification of trajectories for the computer simulation of dynamical systems. *Nonlinearity*, 4(3):961–979, 1991.

[125] P. K. Seidelmann. *Explanatory Supplement to the Astronomical Almanac*. University Science Books, Mill Valley, California, 1992.

[126] R. Seydel, F. W. Schneider, T. Küpper, and H. Troger, editors. *Proceedings of the Conference at Würzburg, Aug. 1990, Bifurcation and Chaos: Analysis, Algorithms, Applications*. Birkhäuser, Basel, 1991.

[127] SIMBAD. SIMBAD Astronomical Database, 2001.

[128] Solar System Dynamics Group. *The HORIZONS On-Line Ephemeris System.* Solar System Dynamics Group at JPL. NASA, ftp://ssd.jpl.nasa.gov/pub/ssd/Horizons_doc.ps, 2000. Version 2.80.

[129] V. Stahl. Interval methods for bounding the range of multivariate polynomials. Technical report, Research Institute for Symbolic Computation, Johannes Kepler University at Linz, Linz, Austria, 1995.

[130] E. M. Standish. JPL planetary and lunar ephemerides, DE405/LE405. Interoffice Memorandum IOM 312. F - 98 - 048, Jet Propulsion Laboratory, Aug 1998.

[131] U. Storck. Numerical integration in two dimensions with automatic result verification. In *Scientific Computing with Automatic Result Verification, E. Adams, U. Kulisch (Eds.)*, pages 187–224. Academic Press, 1993.

[132] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, third edition, Jul 1997. ISBN 0201889544.

[133] R. Tönjes. Private communication.

[134] G. L. Verschuur. *Impact! : The Threat of Comets and Asteroids*. Oxford University Press, Nov 1996. ISBN 0195101057.

[135] W. Walster. Private communication.

[136] W. Walster. Compiler support to compute sharp intervals without wasted splitting. In Corliss et al. [36].

[137] W. Walster. The future of intervals. In Kraemer and v. Gudenberg [71], pages 1–18.

[138] R. Warnock and J. Ellison. From symplectic integrator to Poincare map: spline expansion of a map generator in Cartesian coordinates. *Appl. Numer. Math.*, 29:89–98, 1999.

[139] C. M. Will. The theoretical tools of experimental gravitation. In Bertotti [8], pages 1 – 110. Proceedings of the International School of Physics “Enrico Fermi” – Course LVI.

[140] M. A. Wolfe. *Numerical Methods for Unconstrained Otimization – An Introduction*. Van Nostrand Reinhold, New York, 1978.

[141] H. Wolpe. Patriot missile defense – software problem led to system failure at Dhahran, Saudi Arabia. Technical Report B-247094, United States General Accounting Office, Information Management and Technology Division, Washington, D. C., 1992.

[142] D. K. Yeomans. Comet and asteroid ephemerides for spacecraft encounters. *Celestial Mechanics and Dynamical Astronomy*, pages 1–12, 1997.