RIGOROUS HIGH-PRECISION ENCLOSURES OF FIXED POINTS AND THEIR INVARIANT MANIFOLDS

By

Alexander N. Wittig

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Physics
Mathematics

2011

Abstract

RIGOROUS HIGH-PRECISION ENCLOSURES OF FIXED POINTS AND THEIR INVARIANT MANIFOLDS

By

Alexander N. Wittig

The well established concept of Taylor Models is introduced, which offer highly accurate C^0 enclosures of functional dependencies, combining high-order polynomial approximation of functions and rigorous estimates of the truncation error, performed using verified arithmetic. The focus of this work is on the application of Taylor Models in algorithms for strongly non-linear dynamical systems.

A method is proposed to extend the existing implementation of Taylor Models in COSY INFINITY from double precision coefficients to arbitrary precision coefficients. Great care is taken to maintain the highest efficiency possible by adaptively adjusting the precision of higher order coefficients in the polynomial expansion. High precision operations are based on clever combinations of elementary floating point operations yielding exact values for round-off errors. An experimental high precision interval data type is developed and implemented. Algorithms for the verified computation of intrinsic functions based on the High Precision Interval datatype are developed and described in detail. The application of these operations in the implementation of High Precision Taylor Models is discussed.

An application of Taylor Model methods to the verification of fixed points is presented by verifying the existence of a period 15 fixed point in a near standard Hénon map. Verification is performed using different verified methods such as double precision Taylor Models, High Precision intervals and High Precision Taylor Models. Results and performance of each method are compared.

An automated rigorous fixed point finder is implemented, allowing the fully automated search for all fixed points of a function within a given domain. It returns a list of verified enclosures of each fixed point, optionally verifying uniqueness within these enclosures. An application of the fixed point finder to the rigorous analysis of beam transfer maps in accelerator physics is presented.

Previous work done by Johannes Grote is extended to compute very accurate polynomial approximations to invariant manifolds of discrete maps of arbitrary dimension around hyperbolic fixed points. The algorithm presented allows for automatic removal of resonances occurring during construction.

A method for the rigorous enclosure of invariant manifolds of continuous systems is introduced. Using methods developed for discrete maps, polynomial approximations of invariant manifolds of hyperbolic fixed points of ODEs are obtained. These approximations are outfit with a sharp error bound which is verified to rigorously contain the manifolds. While we focus on the three dimensional case, verification in higher dimensions is possible using similar techniques.

Integrating the resulting enclosures using the verified COSY VI integrator, the initial manifold enclosures are expanded to yield sharp enclosures of large parts of the stable and unstable manifolds. To demonstrate the effectiveness of this method, we construct enclosures of the invariant manifolds of the Lorenz system and show pictures of the resulting manifold enclosures. To the best of our knowledge, these enclosures are the largest verified enclosures of manifolds in the Lorenz system in existence.

To my parents

ACKNOWLEDGMENT

During the past five years I had the pleasure of collaborating with many exceptional people here at Michigan State University, both in the Department of Physics and Astronomy and the Department of Mathematics.

First and foremost, I would like to thank my academic advisor Professor Martin Berz for his continued support, patience, and determined guidance when needed during my graduate studies. At the same time, I shall not forget Professors Kyoko Makino and Sheldon Newhouse without whom my research would not have been possible.

I also would like to acknowledge Professors Pratt, Mahanti and Schenker for kindly serving on my guidance committee.

A large part of my graduate research has been supported by a generous scholarship by the Studienstiftung des Deutschen Volkes as well as further financial consideration by the U.S. Department of Energy and the Thesis Completion Fellowship by the Michigan State University Graduate School.

This acknowledgement would not be complete without thanking my former and current fellow students Shashikant Manikonda, Youn-Kyung Kim, Johannes Grote, Pavel Snopok, Alexey Poklonskiy, Ravi Jagasia, He Zhang, Roberto Armellin, Pierluigi DiLizia, and Benjamin Loseth. The numerous discussions with them, as well as the many shared laughs, were of great value to me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis is the product of my work as a dual degree student in both the Department of Physics and Astronomy as well as the Department of Mathematics at Michigan State University. As such, parts of this thesis may be of more interest to the mathematician, while other parts may be more interesting to the physicist.

But before jumping into a detailed discussion of the various research topics making up this work, I want to make clear why I felt attracted in bridging the gap of two disciplines with this interdisciplinary research. Instead of attempting to put this into my own words, I will quote the words of Norbert Wiener in what will be the only block quote in this dissertation. The following is taken from the preface of Wiener's famous book *Cybernetics: or Control and Communication in the Animal and the Machine*[43]:

> For many years Dr. Rosenblueth and I had shared the conviction that the most fruitful areas for the growth of the sciences were those which had been neglected as a no-man's land between the various established fields. Since Leibniz there has perhaps been no man who has had a full command of all the intellectual activity of his day. Since that time, science has been increasingly the task of specialists, in fields which show a tendency to grow progressively narrower. A

century ago there may have been no Leibniz, but there was a Gauss, a Faraday, and a Darwin. Today there are few scholars who can call themselves mathematicians or physicists or biologists without restriction. A man may be a topologist or an acoustician or a cleopterist. He will be filled with the jargon of his field, and will know all its literature and all its ramifications, but, more frequently than not, he will regard the next subject as something belonging to his colleague three doors down the corridor, and will consider any interest in it on his own part as an unwarrantable breach of privacy.

These specialized fields are continually growing and invading new territory. The result is like what occurred when the Oregon country was being invaded simultaneously by the United States settlers, the British, the Mexicans, and the Russians – an inextricable tangle of exploration, nomenclature, and laws. There are fields of scientific work, as we shall see in the body of this book, which have been explored from the different sides of pure mathematics, statistics, electrical engineering, and neurophysiology; in which every single notion receives a separate name from each group, and in which important work has been triplicated or quadruplicated, while still other important work is delayed by the unavailability in one field of results that may have already become classical in the next field.

It is these boundary regions of science which offer the richest opportunities to the qualified investigator. They are at the same time the most refractory to the accepted techniques of mass attack and the division of labor. If the difficulty of a physiological problem is mathematical in essence, the physiologists ignorant of mathematics will get precisely as far as one physiologist ignorant of mathematics, and no further. If a physiologist who knows no mathematics works together with a mathematician who knows no physiology, the one will be unable to state his problem in terms that the other can manipulate, and the second will be unable to put the answers in any form that the first can understand. Dr. Rosenblueth has always insisted that a proper exploration of these blank spaces on the map of science could only be made by a team of scientists, each a specialist in his own field but each possessing a thoroughly sound and trained acquaintance with the fields of his neighbors; all in the habit of working together, of knowing one another's intellectual customs, and of recognizing the significance of a colleague's new suggestion before it has taken on a full formal expression. The mathematician need not have the skill to conduct a physiological experiment, but he must have the skill to understand one, to criticize one, and to suggest one. The physiologist need not be able to prove a certain mathematical theorem, but he must be able to grasp its physiological significance and to tell the mathematician for what he should look. We had dreamed for years of an institution of independent scientists, working together in one of these backwoods of science, not as subordinates of some great executive officer, but joined by the desire, indeed by the spiritual necessity, to understand the region as a whole, and to lend one another the strength of that understanding.

1.1 Computers and Numerical Algorithms

Ever since early computers became available, both physicists and mathematicians have used them to examine and simulate problems in their fields. As the speed and computational power of computers grew, so did the the size and scope of the problems they were applied to.

However, surprisingly, the underlying algorithms used to solve various problems have not developed nearly as fast as the technology. The most widely used numerical integrators, for example, are still Runge-Kutta type integrators named after the German mathematicians Carl Runge and Martin Kutta who developed these methods around 1900.

Similarly, the fundamental techniques behind most numerical algorithms in use today are still linear algebraic methods based on mathematical techniques that have been known for over a century. One of the most commonly used operations in modern numerical mathematics is still the inversion of matrices, which is used in approximating solutions to PDEs, fitting of parameters and many other problems in numerical analysis.

While the implementations of these algorithms have become better and faster, and new types of algorithms to solve these problems have been developed, the underlying approach to the actual numerical problem to be solved has not changed very much since the early days of the computer age. In part this is due to the fact that many of the currently used methods provide quick, sufficiently good results to be used in physics and engineering applications.

Many new discoveries would have been impossible had it not been for the application of these numerical techniques to large real world systems. In physics, nowadays the field of computational physics has become a fully accepted third branch, equally well established besides theoretical and experimental physics. Engineers use numerical tools every day in simulations during the design process for new and exciting products.

Also in mathematics, computer simulations have been widely used, especially in the field of dynamical systems. Everybody has seen pictures of the Lorenz “butterfly” attractor, or the strange attractor in the Hénon map, or the fractal pictures of the Mandelbrot sets. However, while having spurred lots of interest and research into chaos theory, except for the very simplest one dimensional system very little is actually rigorously known about them. The numerical simulations have spurred lots of conjectures, many of which are widely believed to be true. But almost none of these have been rigorously proven to this day.

1.2 Problem

Why is it that, while having a huge impact in Physics and Engineering, computers have not yet had much impact in the mathematical field in terms of rigorous understanding?

The following simple example demonstrates the problem of using computers in dynamical systems research. Consider the logistic map given by

$$f(x) = r \cdot x \cdot (1 - x) \tag{1.1}$$

where r is a parameter. For $r = 4$, it is easy to see that the map maps the interval $[0, 1]$ onto itself.

A typical question in dynamical systems is the behavior of a system under iteration. Starting with the value 15/16, this very simple map is iterated repeatedly. To do so on a computer typically means that the actual numbers are approximated by so called double precision floating point numbers (see Section 2.1). Fortunately, the number 15/16 can be represented exactly in double precision, as can be 4 and 1, so in the initial step and the representation of $f(x)$ are not a problem.

Reaching ahead a little, the logistic map is simultaneously evaluated in rigorous interval arithmetic (see Section 2.2) using the rigorous, high precision intervals introduced in Chapter 3. While delving into the details in the following chapters, suffice it to say at this point that these intervals provide a mathematically correct enclosure of the actual result.

Figures 1.1 through 1.11 show the evolution of both the floating point value and the mathematically correct high precision interval after a growing number of iterations. The graph shown is that of $f(x)$ over the interval $[-1, 1]$. The red mark on the x-axis of each graph shows the value of the floating point evaluation, while the two blue bounds are enclosures of the mathematically correct value.

In the beginning, the red mark and the blue bounds are identical, and the blue bounds cover up the red mark. However, as the iteration continues, the two values grow apart. After 55 iterations (Figure 1.6), it is clearly visible that the floating point value on the right and the blue sharp interval enclosure of the mathematically correct value do not coincide any more.

The situation does not improve with further iteration, and in the end it happens that not a single decimal digit of the result computed using floating point arithmetic is correct any more, as the correct result is found near 0 after 60 iterations, while the floating point value is close to 1 at the other end of the interval.

So why not just use these High Precision Intervals instead of double precision floating point numbers? Continuing the process to 130 iterations (Figure 1.8) shows that just as the double precision value grew increasingly inaccurate, the High Precision interval also loses sharpness. This trend has been going on all along, it was just not visible so far due to its scale: the High Precision interval started with a width of about 10^{-60}, so that the gradual change in the width was not visible in the picture up until 130 iterations.

After about 130 iterations, however, it is clear that the width of the interval grows quickly, and after 133 iterations, it already covers 1/3 of the entire interval, rendering any further computation useless.

This very simple example demonstrates the issues involved in rigorous computation. Even after a very modest number of iterations of certain rather trivial maps the errors accumulated can be staggering. Considering how many evaluations of a typically much more complicated function are needed to just perform one single step in a Runge-Kutta integration scheme, and how many of such steps are typically performed, it becomes quite apparent that while a powerful tool in the study of mathematical and physical systems, computers can very quickly reach their limitations when applied to certain problems.

While the example in Equation 1.1 is particularly chosen to exhibit chaotic behavior, this is not an untypical case. Most interesting real world systems studied by physicists, such as for example in celestial mechanics or accelerator physics, exhibit similar chaotic motion and are just as susceptible to very fast accumulation of errors. This is not even a question of making computer implementations better, but this is “hard coded” into the DNA of the problem itself. While posing a great problem, at the same time this effect is what makes the study of such systems so interesting and rewarding.

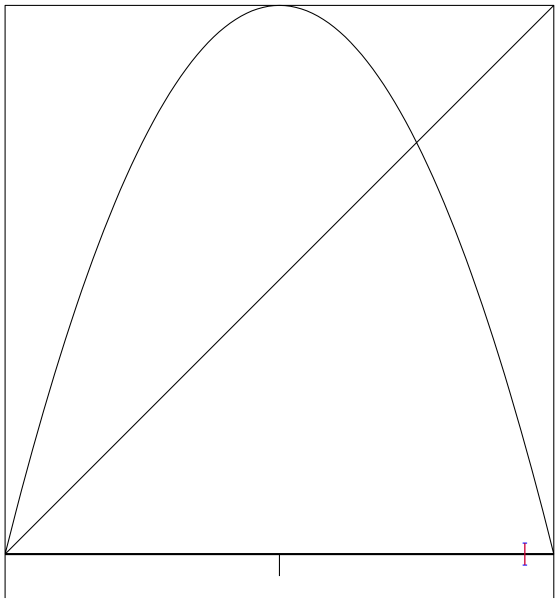


Figure 1.1: The logistic map for $r = 4$ after 5 iterations with floating point numbers (red) and High Precision intervals (blue). For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation.

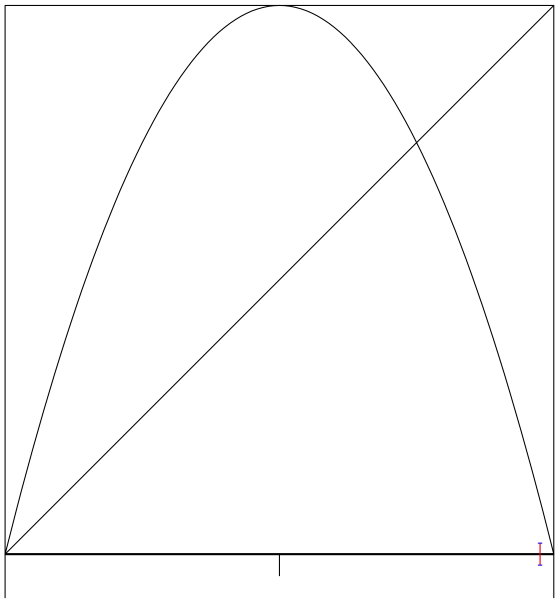


Figure 1.2: The logistic map for $r = 4$ after 15 iterations with floating point numbers (red) and High Precision intervals (blue).

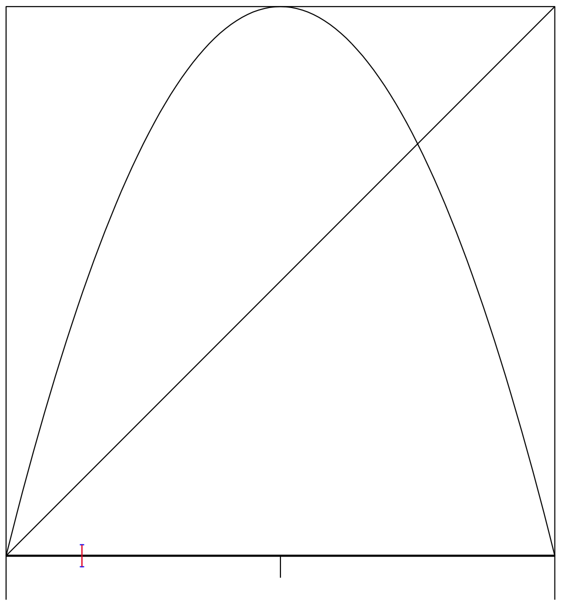


Figure 1.3: The logistic map for $r = 4$ after 25 iterations with floating point numbers (red) and High Precision intervals (blue).

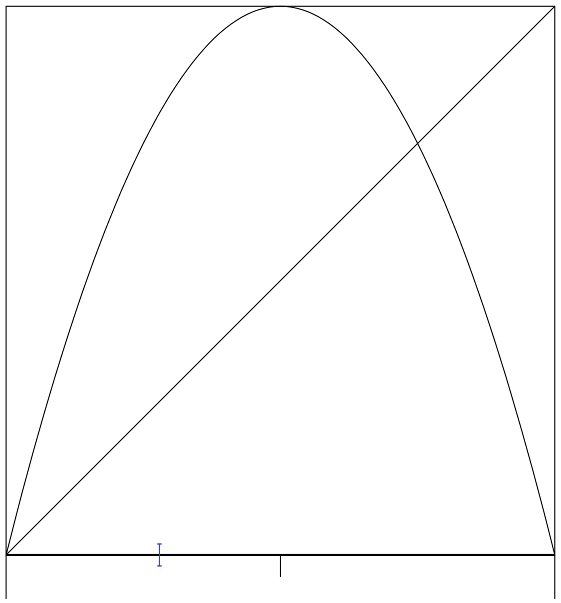


Figure 1.4: The logistic map for $r = 4$ after 35 iterations with floating point numbers (red) and High Precision intervals (blue).

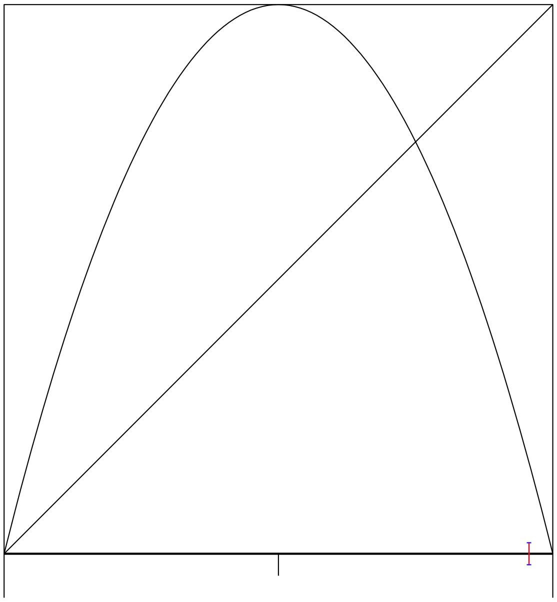


Figure 1.5: The logistic map for $r = 4$ after 45 iterations with floating point numbers (red) and High Precision intervals (blue).

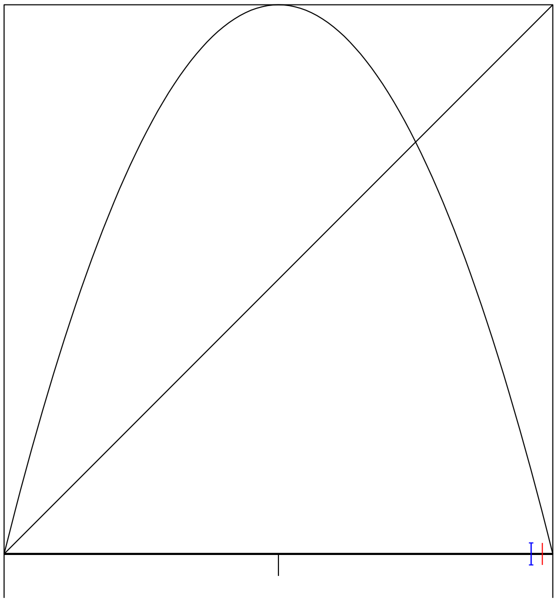


Figure 1.6: The logistic map for $r = 4$ after 55 iterations with floating point numbers (red) and High Precision intervals (blue).

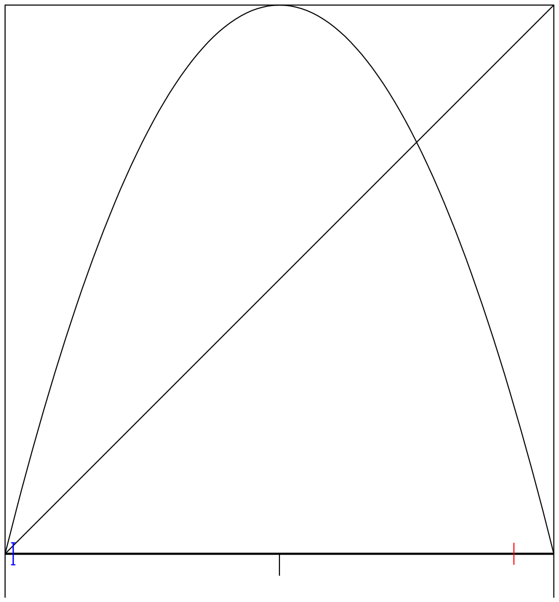


Figure 1.7: The logistic map for $r = 4$ after 60 iterations with floating point numbers (red) and High Precision intervals (blue).

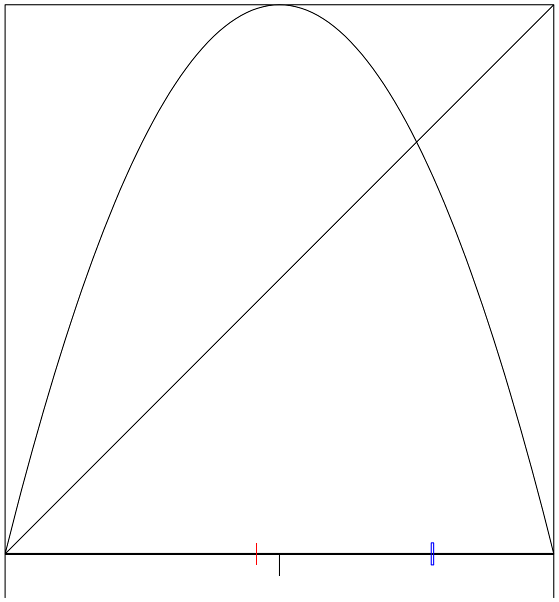


Figure 1.8: The logistic map for $r = 4$ after 130 iterations with floating point numbers (red) and High Precision intervals (blue).

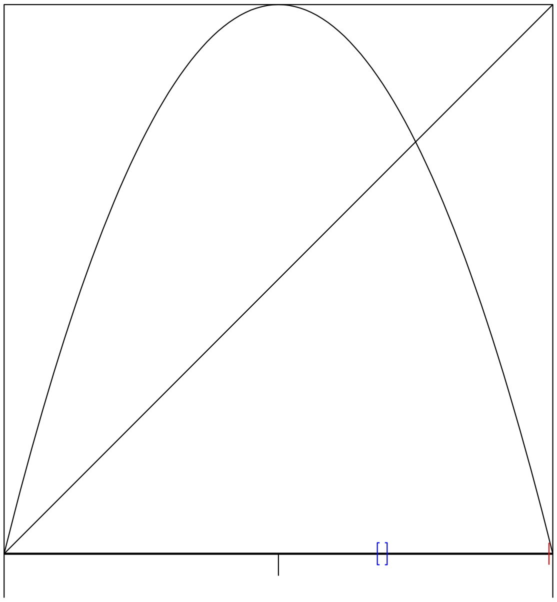


Figure 1.9: The logistic map for $r = 4$ after 131 iterations with floating point numbers (red) and High Precision intervals (blue).

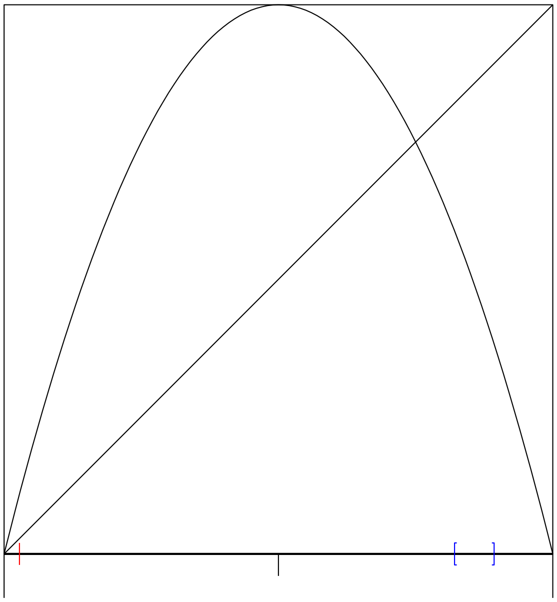


Figure 1.10: The logistic map for $r = 4$ after 132 iterations with floating point numbers (red) and High Precision intervals (blue).

1.3 Outline

This dissertation certainly will not solve all open problems in rigorous numerics. But it will present various relatively new high order, verified numerical techniques which will then be applied to various problems in dynamical systems. These techniques, developed mostly over the last decade, will be improved to meet the demands of working with even more complicated dynamical systems. It is the belief of the author of this dissertation that these tools and techniques will prove to be useful in the future study of dynamical systems via computer assisted proofs.

In Chapter two, some general concepts of numerical computation will be introduced, including floating point numbers, and methods for verified rigorous computations such as interval arithmetic and high order Taylor Model methods. These allow the implementation of numerical computations which result in a mathematically rigorous result that can be used to prove various statements.

Chapter three describes in detail the design and implementation of a major improvement of the Taylor Model methods introduced in Chapter two. The introduction of rigorous high precision Taylor Models with high precision coefficients presented in this chapter will form the basis for many exciting new computations that are not possible with the currently available rigorous numerical techniques.

Chapter four will deal with the rigorous identification and verification of fixed points of maps. Again using Taylor Model methods we will prove the existence of a period 15 fixed point in the Hénon map with near-standard parameters. We then extend this work to describe an automated rigorous global fixed point finder, and apply this tool to a rigorous analysis of a beam transfer map from accelerator physics.

In Chapter five, various methods are presented to compute rigorous enclosures of invariant manifolds using high order Taylor Model methods. Unlike many of the existing non-verified methods, the techniques detailed here allow the rigorous enclosure of large pieces of invariant manifold with a very small error.

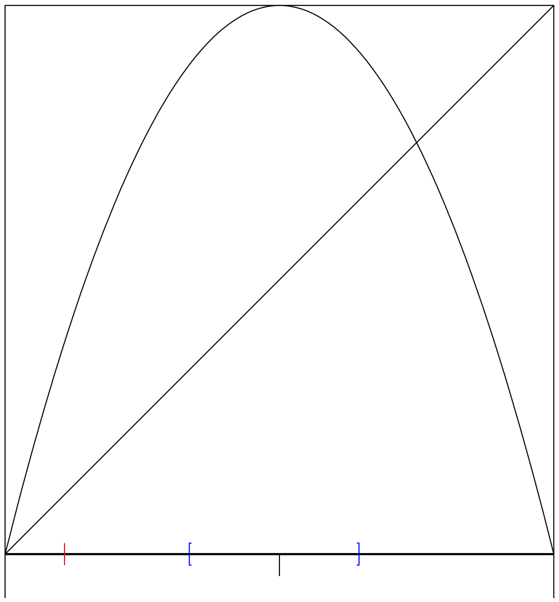


Figure 1.11: The logistic map for $r = 4$ after 133 iterations with floating point numbers (red) and High Precision intervals (blue).

Chapter 2

Fundamental Concepts

In this chapter we introduce certain well-known and fundamental concepts used throughout this thesis.

2.1 Floating Point Numbers

To represent calculations on the real numbers on a computer, most modern processors use floating point numbers. The concept behind the representation of a floating point number is essentially the same as the "scientific notation" in terms of relevant digits and a power of ten to represent the order of magnitude. The same is done with floating point numbers. The only difference is that, due to the binary number system, a power of two is used to signify the magnitude.

Definition 2.1. We define the set of all floating point numbers R to be given by

$$R = \{m_z \cdot 2^{e_z} |\ 2^{t-1} \leqslant |m_z| < 2^t;\ \underline{M} < e_z < \overline{M}\}$$

The constants $t, \underline{M}, \overline{M}$ are fixed and define the floating point number system. $\underline{M}$ and $\overline{M}$ define the exponent range and thus the largest and smallest representable numbers. To make the following proofs easier to understand, we will assume that the exponent range is unlimited, i.e. $\underline{M} = -\infty$ and $\overline{M} = \infty$. This is of course not true for computer systems, where overflows and underflows of the exponent may happen. In our practical implementation we have to deal with those cases separately. The parameter t is the mantissa length in binary digits and thus defines the relative precision of the floating point system (see below).

In the following we will use floating point systems with different mantissa lengths which we will denote by R_t. Over- and underflows notwithstanding, we clearly have that $R_t \subset R_{t'}$ if $t \leqslant t'$. The lower bound requirement on the mantissa is called the normalization. With this additional requirement, the values represented by floating point numbers become unique. Mantissae with absolute value less than 2^{t-1} can be multiplied by a power of two so that they lie within the allowed range for the mantissa, while the exponent is adjusted accordingly.

Given any real number $r \in \mathbb{R}$ within the range of the floating point representation, we will denote by $\tilde{r} \in R$ the closest floating point number in the given system of floating point numbers. Then it follows readily from Definition 2.1 that

$$\frac{|r - \tilde{r}|}{|r|} < \epsilon_m = 2^{-t}$$

The value ϵ_m is called the machine precision and is given by the length of the mantissa t.

Every floating point implementation has to provide at least the basic operations addition, subtraction, and multiplication. Clearly the mathematical result of any of those operations on two arbitrary floating point numbers $a, b \in R$ does not necessarily have to be in R. Thus, the floating point operations corresponding to $+, -, \times$ are not the same as their mathematical counterparts on the real numbers. Let $\oplus, \ominus, \otimes$ denote the floating point operations for $+, -, \times$.

Definition 2.2. Let $\odot$ denote one of the floating point operations $\oplus, \ominus, \otimes$ and $\bullet$ the same operation on the real numbers.

The operation $\odot$ is said to be *round-to-nearest* if $\forall a, b \in R$

$$|(a \odot b) - (a \bullet b)| = \min_{x \in R}(x - (a \bullet b))$$

Note that, if a floating point operation is round-to-nearest, the result is the floating point number closest to the mathematically correct result. In case of a toss-up, i.e. if the mathematically correct result is exactly between two floating point numbers, we will accept either one. Another immediate consequence is that if the result of an operation is representable exactly by a floating point number then we obtain the correct result without roundoff errors.

From this definition a bound for rounding errors and a useful condition for the mantissa of the result of a round-to-nearest operation $\odot$ easily follow. Let $z = m_z \cdot 2^{e_z} = a \odot b$. Then:

$$|z - (a \bullet b)| < \epsilon_m \cdot z \tag{2.1}$$

This is clear since if the error was more than $\epsilon_m \cdot z$ then either the floating point number $(m_z + 1) \cdot 2^{e_z}$ or $(m_z - 1) \cdot 2^{e_z}$ would be closer to the correct result. Furthermore for the mantissa m_z, the following equation holds:

$$m_z = \left[\frac{m_a \cdot 2^{e_a} \bullet m_b \cdot 2^{e_b}}{2^{e_z}} \right] \tag{2.2}$$

where $[x]$ denotes rounding to the nearest integer.

In most modern computers the constants $t, \underline{M}, \overline{M}$ are defined to follow the IEEE 754 standard[2]. The double precision numbers defined in that standard specify that $t = 53$, $\underline{M} = 1023$, $\overline{M} = -1024$. Thus, for double precision numbers $\epsilon_m = 2^{-53} \approx 10^{-16}$. Therefore in double precision we can represent about 16 valid decimal digits. The standard also defines that the elementary floating point operations $\oplus, \ominus, \otimes$ can be set to be round-to-nearest. Consistent with the notation introduced above, we will denote the set of double precision floating point numbers by R_{53}.

2.2 Interval Arithmetic

Traditionally, verified computations have been performed using interval arithmetic. The development of this technique began with the groundbreaking in the 1960s mostly by R. E. Moore[38] and since has formed a well known and broadly studied field. It is based on defining the result of an arithmetic operation on intervals as an interval containing all possible solutions, i.e. for compact intervals $A \subset \mathbb{R}$, $B \subset \mathbb{R}$ and an operation $\circ$, the result of $A \circ B$ is a compact interval satisfying the property

$$A \circ B \supset \{a \circ b \mid \forall a \in A, b \in B\}.$$

While easy to implement rigorously on a computer using the correct rounding of floating point numbers and well studied, interval arithmetic has various intrinsic drawbacks which inflate its error bounds tremendously and make it practically useless for many real world applications. While the final result of an interval computation is guaranteed to contain the

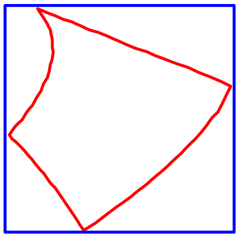


Figure 2.1: Enclosure of a relatively simple, non-convex set (red) by an interval box (blue) and the resulting overestimation.

correct result, the width of the resulting interval is typically much larger than mathematically necessary. This effect if called overestimation, and it is caused by various factors.

2.2.1 Dependency Problem

One source of over-estimation in interval arithmetic is the so called dependency problem. Evaluating a simple expression such as $x - x$ in interval arithmetic for $x = [-1, 1]$, one obtains

$$x - x = [-1, 1] - [-1, 1] = [-2, 2],$$

since the dependence of x in both intervals is lost. While correctly enclosing the mathematically rigorous result 0, vast overestimation is introduced.

There have been various techniques developed to try to minimize this effect through manual rewriting of equations before evaluating them in interval arithmetic, but they require manual rewriting for each equation and only ease the dependency problem, but do not solve it.

2.2.2 Wrapping Effect

Another problem of intervals that appears in higher dimensional systems is their limited shape. Even in the enclosure of very simple, convex sets large overestimation is introduced even if the best possible interval enclosure of the set were somehow computed. The enclosure of arbitrary, complicatedly shaped sets makes the problem even worse. This is known as the wrapping effect and is illustrated in Figure 2.1.

Since in complicated computations consisting of repeated iteration of similar steps this wrapping happens repeatedly for every intermediate result, these errors due to the wrapping effect can add up significantly during the computation.

2.2.3 Scaling

In interval arithmetic, the overestimation in general scales linearly with the width of an interval.

Consider again the example used before:

$$f(x) = x - x.$$

As described before, when evaluated over the domain $[-1, 1]$, large overestimation is introduced, and the width of the result is 4. Evaluating the same function instead over the interval $[m - k, m + k]$, where $|m|, |k| < 1$, yields the following interval result

$$f([m - k, m + k]) = [m - k, m + k] - [m - k, m + k] = [-2 \cdot k, 2 \cdot k].$$

Clearly, the width of the result, $4 \cdot k$, scales linearly with the width of the initial interval,

$2 \cdot k$.

Thus, to obtain a result of a prescribed width of less than ϵ, in general the width w of the initial interval to evaluate an expression with goes as

$$w = \epsilon / C \tag{2.3}$$

for some problem dependent constant C.

Particularly in higher dimensions, this quickly leads to an effect commonly referred to as the dimensionality curse. The number of boxes N required to cover an n dimensional box $[-1, 1]^n$ by boxes of side length less than w is given by the expression

$$N = \left(\frac{2}{w}\right)^n.$$

That is, the number of boxes scales exponentially with the dimensionality of the problem. It is therefore essential to make w as large as possible so that the number of boxes required is minimized.

2.3 Taylor Model Arithmetic

To overcome the problems inherent in Interval arithmetic, Martin Berz, Kyoko Makino et al. developed the concept of *Taylor Models.* Taylor Models use high order polynomial expansions based on automatic differentiation techniques combined with automatically verified error bounds over a given domain to represent very sharp enclosures of sets[28, 29, 9, 30].

In the following, we will give a brief overview over Taylor Models, drawing largely from [29], which is also a good reference for a more in depth description of Taylor Models.

A Taylor Model consists of a polynomial $P : \mathbb{R}^p \to \mathbb{R}$ of fixed order n in p variables as well as an remainder term $R \geqslant 0$. All Taylor Models are defined on a specific domain $D \subset \mathbb{R}^p$, which by convention and without loss of generality is typically assumed to be $[-1, 1]^p$. We write a Taylor Model T as $T = P(\vec{x}) \pm R$ or $T = (P, R)$.

Let now $f : D \to \mathbb{R}$ be any bounded function on D. We say the Taylor Model (P, R) is a *representation* of f, if

$$|f(x) - P(x)| \leqslant R \; \forall \, x \in D.$$

Note in particular that this definition is valid for any function f independent of its smoothness.

Consider now the size of R in such a representation. If f is in C^{n+1}, Taylor's theorem guarantees that R can be chosen in such a way that

$$R = \mathcal{O}\left(|D|^{n+1}\right) \text{ as } |D| \to 0.$$

2.3.1 Taylor Model Operations

Elementary operations on two Taylor Models (P_1, R_1) and (P_2, R_2) are defined as

$$(P_1, R_1) + (P_2, R_2) = (P_1 + P_2, R_1 + R_2)$$

$$c \cdot (P, R) = (c \cdot P, |c| \cdot R) \text{ for } c \in \mathbb{R}$$

$$(P_1, R_1) \cdot (P_2, R_2) = (P_{1 \cdot 2}, R_{1 \cdot 2})$$

Here $P_{1\cdot 2}$ represents $P_1 \cdot P_2$ truncated to order n. Letting $P_e = P_1 \cdot P_2 - P_{1\cdot 2}$ be the remaining polynomial terms of order $(n+1)$ through $2n$, $R_{1\cdot 2}$ is then given by

$$R_{1\cdot 2} = R_1 \cdot R_2 + B(P_e) + B(P_1) \cdot R_2 + B(P_2) \cdot R_1.$$

$B(P)$ in this context represents a bound on the maximum absolute value of the range of P over the domain D, i.e.

$$|P(x)| < B(P) \; \forall x \in D.$$

Let $f_1, f_2 : D \to \mathbb{R}$ be two bounded functions, and $(P_1, R_1), (P_2, R_2)$ two Taylor Models representing f_1 and f_2 respectively. It can be shown[31, 29, 28] that the Taylor Model operations as defined above satisfy the two properties

$$(P_1, R_1) + (P_2, R_2) \text{ represents } f_1 + f_2,$$

$$(P_1, R_1) \cdot (P_2, R_2) \text{ represents } f_1 \cdot f_2.$$

Furthermore, for any real c we have that the Taylor Model $c \cdot (P_1, R_1)$ represents the function $c \cdot f_1$.

2.3.2 Intrinsic Functions on Taylor Models

In the references given above, further discussions of intrinsic functions such as trigonometric functions for Taylor Models can be found. For all of these operations the representation property is retained, i.e. if (P, R) represents f then $\mathrm{op}((P, R))$ represents $\mathrm{op}(f)$ on the chosen domain.

For example, consider the exponential function exp for Taylor models. Let (P, R) be a Taylor Model, and denote by P_0 the constant part of P, and by $P_> = P - P_0$ the non-constant part of P. Then

$$\begin{aligned}
\exp((P,R)) &= \exp(P_0 + (P_>, R)) \\
&= \exp(P_0) \cdot \exp((P_>,R)) \\
&= \exp(P_0) \cdot \left(\sum_{i=0}^{n} \frac{(P_>, R)^i}{i!} + \frac{(P_>, R)^{n+1}}{(n+1)!} \exp(\theta \cdot (P_>, R)) \right)
\end{aligned}$$

where the expression in parenthesis is the truncated Taylor series of the exponential function and the Taylor remainder term which is valid for some $\theta \in [0, 1]$.

Note that since P_0 is only a number, a sharp enclosure of $\exp(P_0)$ with a rigorous error bound can be computed, e.g. using the algorithm described in Section 3.4.5. Furthermore, the sum in parenthesis can be evaluated directly in Taylor Model arithmetic, yielding both a polynomial part as well as an remainder term.

For the last Taylor remainder part, note that since $P_>$ is a polynomial of order at least 1, in Taylor Model arithmetic $(P_>, R)^{n+1}$ does not have any polynomial part since the resulting polynomial is of order greater than n. Thus $(P_>, R)^{n+1}$ is a Taylor Model consisting only of a Taylor Model remainder term. The expression $\exp(\theta \cdot P_>)$ can be bounded by first computing a constant part only Taylor Model (B, R_B) representing a bound of $P_>$ over the domain D. Then a rigorous upper bound of the maximum absolute value of $\exp([0, 1] \cdot (B, R_B))$ can be computed using again constant part only Taylor Model arithmetic.

2.3.3 Scaling

Let $f, g \in C^{n+1}$ and $(P_f, R_f), (P_g, R_g)$ are n-th order Taylor Model enclosures of f and g over some domain of width w respectively. Furthermore let $(P_{f+g}, R_{f+g}) = (P_f, R_f) + (P_g, R_g)$, $(P_{f \cdot g}, R_{f \cdot g}) = (P_f, R_f) \cdot (P_g, R_g)$, and $(P_{op(f)}, R_{op(f)}) = op((P_f, R_f))$, where op represents any intrinsic operation.

Assume the error terms R_f and R_g scale with order $n+1$ in the width of the initial domain, i.e.

$$\begin{aligned} R_f &= \mathcal{O}(w^{n+1}) \\ R_g &= \mathcal{O}(w^{n+1}), \end{aligned}$$

for sufficiently small w.

It can then be shown that in the absence of round-off errors the remainder terms $R_{f+g}, R_{f \cdot g}, R_{op(f)}$ for Taylor Model addition, multiplication and intrinsic operation op all also satisfy the same scaling property as the initial intervals R_f, R_g, i.e.

$$\begin{aligned} R_{f+g} &= \mathcal{O}(w^{n+1}) \\ R_{f \cdot g} &= \mathcal{O}(w^{n+1}) \\ R_{op(f)} &= \mathcal{O}(w^{n+1}). \end{aligned}$$

This fact is known as the Taylor Model scaling theorem.

So instead of a linear scaling as is the case with pure interval arithmetic, the error bound of Taylor Models scales with the order of the polynomial used. This is crucial especially in higher dimensions to obtain results with a specified accuracy. In Equation2.3, the width w of the initial interval required to obtain a prescribed accuracy ϵ for interval evaluation of a function f scaled linearly in ϵ. For Taylor Models, the same equation reads

$$w = \sqrt[n+1]{\epsilon}/C \tag{2.4}$$

for some problem dependent constant $C > 0$, since the error term scales as $C \cdot w^{n+1}$. That is, the minimum width required for Taylor Models to obtain a prescribed accuracy is much larger than in the interval case, so fewer Taylor Models are required.

2.3.4 Order Loss

While always correct in theory, in practice this scaling law breaks down at some point when the width of the initial box becomes too small. This leads to somewhat counterintuitive situations, where Taylor Model methods perform very well over relatively large boxes, but suddenly fail with smaller boxes. The application of Taylor Models presented in Chapter 4 to compute very sharp enclosures of periodic and fixed points in particular will prove to suffer from this effect.

The cause of this effect is due to the limited precision with which the coefficients of the Taylor Model are stored on a computer. Effectively, this limited precision causes a loss of computation order when the coefficients of Taylor Models become too small.

In regular IEEE 754[2] compliant double precision floating point Taylor Models, where coefficients are stored as double precision numbers, each coefficient only carries about 16 significant decimal digits. This finite precision requires careful bookkeeping of the truncation errors during Taylor Model operations. Great care has been taken in implementing Taylor Models to bound all those errors in the remainder bound of the Taylor Model.

However, these round-off errors of each floating point operation do not scale according to the above Taylor Model scaling law. The round-off error for a given coefficient of the Taylor Model resulting from a Taylor Model operation only scales as the coefficient itself. So for linear coefficients, the round-off error only scales linearly, while the round-off error for the constant part does not scale at all.

Under most circumstances, this poses no problem as the absolute value of the round-off error is very much smaller than the error due to the truncation of the Taylor series. However, as the coefficients of a Taylor Model become smaller, the size of the Taylor series truncation error scales as $\mathcal{O}(w^{n+1})$, and the non-scaling round-off errors become a relatively larger contribution to the overall error.

Eventually, the higher order terms of the Taylor Model get smaller than the round-off errors. Effectively then those orders are lost, in the sense that keeping these terms in the Taylor Model does not increase the accuracy of the computation. The error term would not change if these orders were truncated during the computation. This effect is known as the broken scaling law, or *order loss.*

Consider as an example the simple function

$$f(x) = (1 + x + x^2 + x^3)/3.$$

Evaluate this function in Taylor Model arithmetic of order two with a Taylor Model box of width $2 \cdot \epsilon$ around 0 of the form $T = \epsilon \cdot x_1$. Since the operations are performed in double precision floating point arithmetic, the round-off error for the resulting constant part (1/3) is on the order of 10^{-16} as 1/3 is not a finite binary fraction, and has to be truncated after 16 digits. The round-off error of $\epsilon/3$ and $\epsilon^2/3$, respectively, is on the order of $10^{-16} \cdot \epsilon$ and $10^{-16} \cdot \epsilon^2$. Since computations are performed only to order 2, the entire $x^3/3$ term is truncated, yielding a truncation error contribution to the overall error bound on the order of $\epsilon^3/3$.

Let now $\epsilon = 2^{-10} \approx 10^{-3}$. That way, all powers of ϵ are exact floating point numbers, and the only round-off error occurs when dividing by 3. Floating point round-off errors for each of the resulting coefficient are on the order of 10^{-16}, 10^{-20}, and 10^{-24} respectively, while the series truncation error is of order $3 \cdot 10^{-10}$. Taken all together, the error bound of the resulting Taylor Model is on the order of 10^{-10}, clearly dominated by the truncation error, which scales as $\mathcal{O}(\epsilon^3)$ as expected. Note that the magnitude of the second order coefficient is $1/3 \cdot 2^{-20} \approx 3 \cdot 10^{-7}$.

Consider now the case where $\epsilon = 2^{-30} \approx 10^{-9}$. Again powers of ϵ are exact floating point numbers, and the only round-off error occurs when dividing by 3. The round-off errors for each coefficient are now on the order of 10^{-16}, 10^{-25}, and 10^{-34} respectively, while the truncation error has shrunk to the order of $3 \cdot 10^{-28}$. It can clearly be seen how the round-off error of the constant part does not scale with ϵ at all. The resulting error bound thus is on the order of 10^{-16}, easily surpassing the second order coefficient, $1/3 \cdot 2^{-60} \approx 3 \cdot 10^{-19}$, in magnitude.

Thus, carrying out the above computation using only first order Taylor Models would yield the same quality of result as the second order computation. Effectively the second order is lost, and the error scales no better than that of a first order Taylor Model evaluation would. As ϵ gets even smaller, the Taylor Model asymptotically behaves no better than a classical interval, as only the constant part dominates.

The only way to prevent this order loss is to perform the computation using a higher precision for the lower order coefficients. That way, the round-off error of the constant part is reduced in magnitude, and higher order computations remain viable for smaller values of ϵ. Of course for any finite precision, there exists an ϵ such that the Taylor Model still behaves no better than an interval. However as long as this ϵ is sufficiently small, full advantage can be taken of the scaling law. In Chapter 3 we introduce High Precision Taylor Models which will provide such features.

Note that this is a markedly high-order effect. The closest analogy to order loss in interval arithmetic is practically never observed because it only occurs with very narrow intervals. Consider an interval in which both the lower and upper bound are stored as double precision numbers. Since double precision numbers are discreet, there is a minimum width for an interval, given by the spacing of floating point numbers. Once that minimum width has been reached, attempts to reduce the width of the interval will not lead to better results.

Evaluate, for example, the above expression for $f(x)$ in interval arithmetic. Since double precision floating point numbers around $1/3$ are spaced approximately 10^{-16} apart, the resulting interval for $f([0, 10^{-16}])$ could be roughly of the form $[1/3, 1/3+10^{-16}]$. However, reducing the width of the argument further to e.g. $[0, 10^{-20}]$, does not bring any further improvement, as the thinnest possible double precision interval at $1/3$ is roughly $[1/3, 1/3+10^{-16}]$.

Clearly, for all practical purposes, this effect is negligible in interval arithmetic. In Taylor Models, however, it is much more pronounced due to the strong scaling of the higher order terms.

2.3.5 Implementation

A computer implementation of such Taylor Models is provided in the COSY INFINITY[32] package. With this implementation, many of the problems present in interval arithmetic are avoided and even complicated computations can be performed quickly and with minimal overestimation. Among the many applications are the R. E. Moore prize winning verified integrator COSY VI[34, 10, 35] and the global optimizer COSY GO[33], both developed by Kyoko Makino and Martin Berz.

This implementation stores the coefficients of the polynomial constituting each Taylor Model as double precision floating point numbers. In Chapter three, fundamental work will be presented to extend this implementation such that coefficients can be stored in high precision, allowing for much more accurate computations.

Chapter 3

High Precision Operations

In this chapter, we develop the theory and describe the implementation of High Precision intervals within the COSY INFINITY framework, which will then lead naturally to the implementation of High Precision Taylor Models.

To that end, we begin by reiterating some general concepts, as well as introducing well-known, fundamental operations that allow the precise computation of round-off errors for elementary operations on floating point numbers (2.1). Based on those *Dekker operations*[16], named after T.J. Dekker who first published them in 1971, I will then describe the general features of an implementation of a high precision interval data type.

Once the basic routines of this interval data type are available, we will briefly describe a generalization of those elementary arithmetic routines to full-fledged Taylor Models. Based on those elementary operations, we proceed to develop rigorous, verified algorithms to compute various intrinsic functions. These will then form the basis for a full implementation of High Precision Taylor Models.

Our main objectives for the design, as well as the implementation of all of the above algorithms are

Rigor The algorithms must be absolutely rigorous. That means we will take into account all possible sources of errors, including roundoff errors due to floating point operations and errors due to numerical methods used.

Speed Since the algorithms will be included in version 10 of the COSY INFINITY[32] software package, they have to perform well speed-wise. We will optimize our algorithms, using the fact that our calculations do not have to be precise to the last bit, as long as we do not violate the first objective, mathematical rigor.

Precision While focusing on the range of up to about 60 significant digits, the code is designed such that it is not restricted to any fixed precision. This separates it from other existing quadruple precision (about 30 digits) packages based on Dekker operations.

3.1 General Concepts

In this section, we will introduce various general concepts which are used throughout the remainder of this chapter. We will not provide proofs for these methods, but instead refer the reader to the given references or any introductory analysis textbook.

3.1.1 Newton Method

The Newton Method[12] is a well known method for iteratively finding the zeros of a function $f(x)$. For all sufficiently smooth functions f, there is a neighborhood of the solution, in which the convergence of this method is super-linear. This is of particular interest for our problems, since we can use the available non-rigorous floating point intrinsics to quickly obtain an approximation of the result, which we then can use as a starting value. From there, it typically takes very few iterations to converge to the final result in the desired precision.

The Newton iteration step is calculated according to the equation

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{3.1}$$

3.1.2 Interval Newton Method

Based on the traditional Newton method, Moore introduced the following, slightly modified, interval Newton method[37].

$$X_{n+1} = m(X_n) - \frac{f(m(X_n))}{f'(X_n)} \tag{3.2}$$

Here $m(X_n)$ stands for the midpoint of the interval X_n.

If all operations in the above equation are carried out rigorously, and the initial interval contained a zero of the function f, the resulting interval X_{n+1} is also guaranteed to contain a zero. Thus, by iterating 3.2, one can obtain an interval enclosure of a zero of f. For this method to work, it is required that the interval Newton step actually contracts the argument.

Note that it is possible to define X_{n+1} as the intersection of X_n and the interval Newton step on acting X_n. The above statements still hold in that case. This intersection, however, often is not necessary if the interval Newton step is contracting the interval strongly enough.

3.1.3 Taylor Expansion with Remainder

Using the Taylor expansion to evaluate certain intrinsic functions is a very powerful method to obtain good interval approximations[11, 12]. In particular, in many cases we can derive an analytic expression for the remainder term.

A Taylor expansion of a function $f(x)$ up to order n (around 0) is given by 3.3

$$f(x) = \sum_{i=0}^{n} \frac{f^i(0)}{i!} x^i + R_n \tag{3.3}$$

The remainder term of that expansion in the Lagrange form is given by:

$$R_n = \frac{f^{n+1}(\xi)}{(n+1)!} x^{n+1} \tag{3.4}$$

for some $\xi \in (0, x)$. While there are other forms for the remainder term, this one is the most useful for our purposes.

Given that $||f^{n+1}||_\infty$ exists, i.e. f^{n+1} is bounded over the whole domain of x, we can simplify the remainder term as follows:

$$|R_n| \leqslant \frac{||f^{n+1}||_\infty}{(n+1)!} |x|^{n+1} \tag{3.5}$$

3.1.4 Leibniz Criterion

Another method for bounding the remainder of a sum is the Leibniz criterion[12]. Given a monotonically decreasing sequence a_n such that $\lim_{n \to \infty} a_n = 0$. Then $a_n \geqslant 0$ for all n

and the series

$$s = \sum_{n=0}^{\infty} (-1)^n a_n$$

does converge. Furthermore, the remainder after the m-th partial sum is given by

$$|s_m - s| = \left| \sum_{n=0}^{m} (-1)^n a_n - \sum_{n=0}^{\infty} (-1)^n a_n \right| = \left| \sum_{n=m+1}^{\infty} (-1)^n a_n \right| \leqslant |a_{m+1}|$$

3.1.5 Kahan Summation

The Kahan summation algorithm[23] is an algorithm that allows the summation of a sequence of floating point numbers, producing a much smaller error than a naive floating point summation would yield. It follows the idea of the Dekker addition, by calculating the roundoff error in each addition and feeding it back into the calculation in the next step. While the result is, of course, still not exact, the roundoff error is greatly reduced compared to naive addition.

Let *sum* be the floating point sum of a sequence of floating point numbers a_i, and $c = 0$ initially. Then, for each a_i the following operations are preformed in floating point arithmetic.

$$\begin{aligned} y &= a_i - c \\ t &= sum + y \\ c &= (t - sum) - y \\ sum &= t \end{aligned}$$

We will not go into details of the algorithm here, for more information see [18, 23].

3.2 Precise Elementary Operations on Floating Point Numbers

In the following subsections we will state some well-known facts about obtaining exact results for the basic floating point operations. While this may sound surprising at first, it is indeed possible to obtain the roundoff errors of the basic floating point operations exactly from within the floating point arithmetic. The theorems and proofs given here are originally due to Dekker[16], who showed that the theorems also hold with slightly lesser requirements on the underlying floating point operations than prescribed by the IEEE 754 standard. But since our implementation will build on IEEE 754 double precision floating point numbers, we will restrict ourselves to those.

To give the reader an idea of how the proofs of those theorems work, we will prove some of the theorems while referring the reader to [16, 25] for others.

All throughout this section, I will use the notation introduced in Section 2.1.

3.2.1 Two-Sum

The first theorem will provide us with a way to calculate the exact round-off error occurring when adding two floating point numbers.

Theorem 3.1. *Let two double precision floating point numbers a and b such that $|a| > |b|$ be given. Let $z = a \oplus b$, $w = z \ominus a$ and $zz = b \ominus w$. Then, neglecting possible over- or underflows during the calculation, we have that $z + zz = a + b$ exactly.*

Proof. Let $a = m_a \cdot 2^{e_a}$ and $b = m_b \cdot 2^{e_b}$. Since $|a| > |b|$ and floating point numbers are normalized, we have that $e_a \geqslant e_b$. It is sufficient to show that $w \in R_{53}$ and $b - w \in R_{53}$, then the result follows readily from optimality of the floating point operations.

Let $z = a \oplus b = m_z \cdot 2^{e_z}$. From 2.2 we get that

$$m_z = \left[m_a \cdot 2^{e_a - e_z} + m_b \cdot 2^{e_b - e_z} \right]$$

Since $|a + b| < 2|a|$ we have that $e_z \leqslant e_a + 1$. Now we consider the two cases $e_z = e_a + 1$ and $e_z \leqslant e_a$.

Assume $e_z = e_a + 1$. Then $m_z = \left[m_a \cdot 2^{-1} - m_b \cdot 2^{e_b - e_a - 1} \right]$ and letting $w = m_w \cdot 2^{e_a}$ we find that

$$\begin{aligned}
|m_w| &= |m_z \cdot 2^{e_z - e_a} - m_a| \\
&= |m_z \cdot 2^{e_z - e_a} - m_a - m_b \cdot 2^{e_b - e_a} + m_b \cdot 2^{e_b - e_a}| \\
&\leqslant |2m_z - m_a - m_b \cdot 2^{e_b - e_a}| + |m_b \cdot 2^{e_b - e_a}| \\
&< 2|m_z - m_a \cdot 2^{-1} - m_b \cdot 2^{e_b - e_a - 1}| + 2^{53} \\
&< 2\frac{1}{2} + 2^{53}
\end{aligned}$$

Since m_w is an integer, we therefore have that $m_w \leqslant 2^{53}$ and thus $w \in R_{53}$, i.e. w is a double precision floating point number.

If $e_z \leqslant e_a$ the exact same proof carries through, the only difference being that we define $w = m_w \cdot 2^{e_z}$.

To prove that $zz \in R_{53}$, we first note that we can write $w = i \cdot 2^{e_b}$ for some integer i since $e_a \geqslant e_b$. Secondly, we have that $|b - w| = |b - z + a| \leqslant |b|$ by optimality. To see this simply let $z = x$, and then apply Definition 2.2. We thus have

$$|zz| = |b - w| = |m_b - i| \cdot 2^{e_b} \leqslant |b| = |m_b| \cdot 2^{e_b} < 2^{53} \cdot 2^{e_b}$$

and thus $(m_b - i) \cdot 2^{e_b} = zz \in R_{53}$. □

Note that by Definition 2.1, floating point numbers are symmetric, i.e. if $a \in R$ then $-a \in R$. Thus the above theorem automatically provides exact subtraction as well.

It is worth mentioning that there are also other algorithms to calculate the same two values without the condition that $a > b$, but requiring some additional floating point operations. The following algorithm is due to Knuth [25]. The advantage of this method is that due to pipelining on modern processors it is often faster to perform the three additional floating point operations instead of having to evaluate a conditional statement on the absolute values of a and b.

Theorem 3.2. *Let two double precision floating point numbers a and b be given. Let $z = a \oplus b$, $b_v = z \ominus a$, $a_v = z \ominus b_v$ and $zz = (a \ominus a_v) \oplus (b \ominus b_v)$. Then, neglecting possible over- or underflows during the calculation, we have that $z + zz = a + b$ exactly.*

Proof. For a proof see, for example, [25]. □

3.2.2 Splitting

Before we can move on to the exact multiplication, we require the concept of the splitting of a double precision number.

Definition 3.3. Let $a \in R_{53}$ be given. We call $a_h, a_t \in R_{26}$ the head and the tail of the splitting of a if

$$\begin{aligned} a_h &= \left[m_a \cdot 2^{-26}\right] \cdot 2^{e_a+26} \\ a_t &= a - a_h \end{aligned}$$

This definition may sound surprising at first. After all a has 53 mantissa bits, but both a_h and a_t only have 26 bits each yielding a total of 52 bits. The solution to this riddle is the fact that the difference $|[a] - a| \leqslant 1/2$, but depending on x it can have either positive or negative sign. So the missing bit is the sign bit of the tail of the splitting.

The following theorem, also presented by Dekker, allows us to calculate such a splitting of a double precision number.

Theorem 3.4. *Let $a \in R_{53}$ be given and let $p = a \otimes (2^{27}+1)$. Then the head of the splitting of a is given by $a_h = p \oplus (a \ominus p)$.*

Proof. Since the proof of this theorem is somewhat technical and does not contribute much to the understanding of these operations, we refer the reader to the papers of Dekker [16] or Shewchuk [41]. □

3.2.3 Multiplication

With the notion of a splitting, we can formulate the following theorem for exact multiplication of two double precision numbers:

Theorem 3.5. *Given two double precision floating point numbers a and b let $a = a_h + a_t$, $b = b_h + b_t$ be a splitting as defined above. Also let $p = (a_h \otimes b_h)$, $q = (a_t \otimes b_h) \oplus (a_h \otimes b_t)$ and*

$r = (a_t \otimes b_t)$. *Then, neglecting possible over- or underflows during the calculation,* $z = p \oplus q$ *and* $zz = (p \ominus z) \oplus q \oplus r$ *satisfy* $z + zz = a \cdot b$ *exactly.*

Proof. First note that for any two numbers $x, y \in R_{26}$ their product $x \cdot y \in R_{52} \subset R_{53}$. This is clear since for $x = m_x \cdot 2^{ex}$ and $y = m_y \cdot 2^{ey}$ we have that $x \cdot y = m_x \cdot m_y \cdot 2^{ex+ey}$ and $|m_x \cdot m_y| < 2^{52}$ since $|m_x| < 2^{26}$ and $|m_y| < 2^{26}$.

We also have that

$$a \cdot b = (a_h + a_t) \cdot (b_h + b_t) = a_h \cdot b_h + a_h \cdot b_t + a_t \cdot b_h + a_t \cdot b_t.$$

Since $a_h, a_t, b_h, b_t \in R_{26}$, each single term in this sum is in R_{52}. Furthermore, the two cross terms $a_h \cdot b_t$ and $a_t \cdot b_h$ have the same exponent and therefore their sum is in R_{53}. Thus p, q, and r, as defined in the statement of the theorem, are exact, and we obtain that $a \cdot b = p + q + r$.

Now we perform an exact addition of p and q as described above, yielding the leading term $z = p \oplus q$ and a remainder term $z_1 = (p \ominus z) \oplus q$. We thus have $a \cdot b = z + z_1 + r$. Close examination of the proof of the exact addition shows that r and z_1 have the same exponent and both are in R_{52}, so their sum can be calculated exactly in R_{53}. This leaves us with the final equation $a \cdot b = z + (z_1 \oplus r) = z + ((p \ominus z) \oplus q \oplus r) = z + zz$, which completes the proof. □

3.3 High Precision Data Types

In this section, we present the detailed algorithms used in our High Precision Interval algebra[45]. We then briefly describe a generalization of these algorithms to High Precision Taylor Models, and proceed to give rigorous methods to compute all intrinsic functions supported by COSY INFINITY with rigorous remainder bounds.

3.3.1 High Precision Intervals

Definition 3.6. A high precision interval A is given by a finite sequence of double precision floating point numbers a_i, and a double precision error term a_{err}. The value of the interval is then given by:

$$A = \left[\sum_{i=1}^{n} a_i - a_{err}, \ \sum_{i=1}^{n} a_i + a_{err} \right]$$

For shorthand notation, we will also refer to the interval as $A = \sum_{i=1}^{n} a_i \pm a_{err}$.

We call a_i the i-th limb of the interval, the number of limbs n is its length. We call the exact sum of all limbs the value of the high precision interval. The rigorous remainder a_{err} is another floating point number that specifies the uncertainty (or width) of the high precision interval. The correct number represented by A is guaranteed to be in the interval $\left[\sum a_i - a_{err}, \sum a_i + a_{err}\right]$. We require a_{err} to always be a positive number. The sequence a_i is also called a floating point expansion of the midpoint of A.

A high precision interval A is called *well conditioned* if for all i we have $|a_i| > |a_{i+1}|$, i.e. the absolute values of the sequence are strictly decreasing.

A high precision interval A is called *normalized* or *canonical* if for all i we have $\varepsilon \cdot |a_i| > |a_{i+1}|$, i.e. the floating point numbers are non-overlapping where ε is the machine precision. Obviously normalized numbers are also well conditioned.

3.3.1.1 Limitations

This representation implies certain limitations to the range of our high precision intervals. Those limitations are a direct consequence of the limitations of double precision floating point numbers.

The range of representable numbers is obviously limited by the range of double precision numbers. The smallest possible non-zero number representable consists of one single limb representing the smallest possible double precision floating point number. That number is of order $2^{-1024} \approx 10^{-309}$.

The largest representable number, in theory, is only limited by available memory, since there is always an unbounded sequence of double precision floating point numbers. In practice, however, we would like our numbers to be normalized. Therefore, the largest representable number is of order $2^{1024} \approx 10^{309}$. Furthermore, our algorithms will encounter over- and underflow errors during the calculations when limbs get close to the double precision limits.

The relative precision of our high precision intervals is bound by the "quantization" in quanta of the size of the smallest possible representable double precision number. Thus, the last digit in a decimal representation of a number is of order 10^{-309}. A number of order one can, therefore, have a maximum of about 309 valid decimal digits. Note that the maximum number of valid digits depends on the size of the represented number itself.

Thus, we have the following limitations for a high precision number X:

$$10^{-309} \lesssim |X| \lesssim 10^{309} \tag{3.6}$$

where the relative precision depends in the order of X.

3.3.1.2 The Accumulator

The key operation in our algorithms is the accumulator. Every other operation will, directly or indirectly, call this function. The accumulator simply takes an arbitrary list of floating point numbers, and adds them up exactly using Dekker's addition. The result will be a high precision interval of a specified maximum length that represents the sum of the input rigorously.

If the length of the output interval is not sufficient to hold the exact result, the accumulator adds the leftover numbers to the error bound of the result. Consequently, the result is always a rigorous enclosure of the mathematically correct result. For performance reason, we also add terms to the error bound that are smaller than the error bound itself. This prevents computationally expensive calculations on numbers that are already below the remainder term. For this purpose, the initial error bound of the result is passed to the accumulator. This allows us to first calculate the error bound of the resulting interval of an operation, and then pass it to the accumulator.

Let a_i denote the input sequence with n elements, and S the resulting high precision interval of maximal length m. The accumulator then sums up the a_i successively using $n-1$ exact additions. This results in one double precision number representing an approximate result of the final sum. This number becomes the first limb s_1 of the result. In the exact additions we also obtained $n-1$ roundoff terms. To these, the same scheme is applied again to calculate the second limb of the result. This calculation is repeated until there are either no more roundoff errors left, or the maximum number of limbs in the result has been reached. If there are terms left, the absolute values of those terms are added to the result's error term

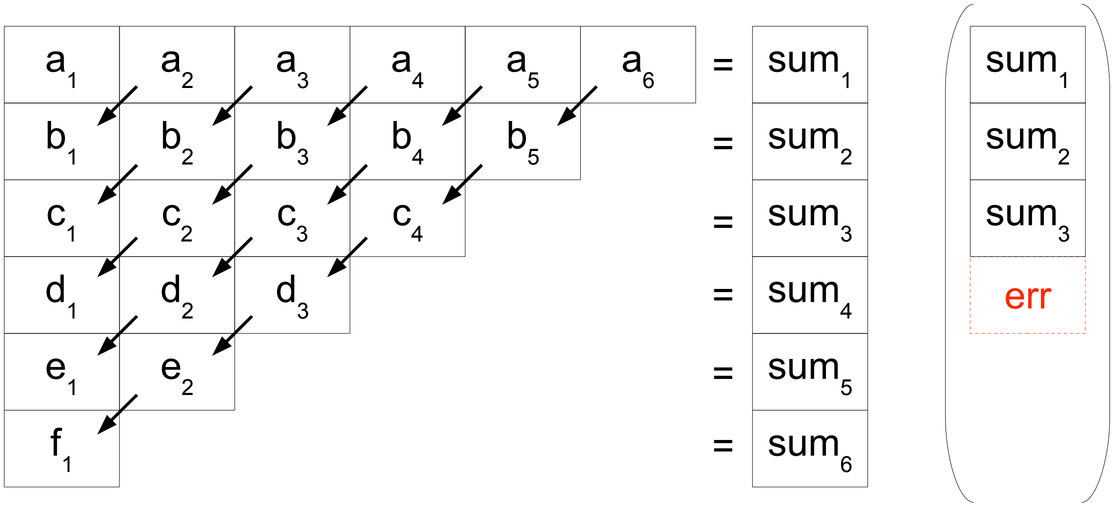


Figure 3.1: Graphical representation of the accumulator algorithm for $n = 6$. There are two cases shown: In the first the result holds six limbs. In the second case, shown in parentheses, the result holds only three limbs. In that case, the accumulation terminates after calculation three limbs. The absolute values of the remaining terms d_i are then added to the error bound.

s_{err}. For $n = 6$ and $m = 3$ this process is graphically shown in 3.1.

To increase performance, we add a slight adjustment to this algorithm. After performing each exact addition, we check if the roundoff error of the result actually is above s_{err}. Only in that is the case is it kept for the next round, otherwise the roundoff error is immediately added to s_{err}. The same test is applied to the resulting limb in each step. If its absolute value is less than the error bound, it is also added to s_{err} and the limb is discarded. The only exception is the very first limb, which is always kept, in order to prevent large intervals from always being centered at 0.

It is important to note that the order of the input numbers is important. It does influence the result and, more importantly, the performance of this algorithm. Best performance is obtained if the sequence a_i is ordered by increasing absolute value of the numbers. If two small numbers just above the error bound are added, the absolute value of the roundoff error for that operation will surely be below the error bound. As described above, it will therefore immediately be discarded from the further calculation. If, however, one such small number is added to an already significantly larger number, the roundoff will be of the order of the small number added. Thus the small number will just propagate as a roundoff error into the next round. So for best performance, it is important that, if possible, the numbers are at least roughly ordered by magnitude.

It is also interesting to note that the output of this algorithm is not necessarily normalized. In fact, if cancellation occurs in the addition, this can lead to poorly conditioned results. However, in most real world cases it is very likely that the numbers returned are well conditioned. This possibility for cancellation is the reason why none of our algorithms have any normalization requirements for the input numbers. However, we do optimize our algorithms to perform better with well conditioned or even normalized input.

3.3.1.3 Addition and Subtraction

Given the accumulator, addition of two high precision intervals A and B into C is quite trivial. All that has to be done is to concatenate the sequences of limbs making up the two numbers into a temporary array, and apply the accumulator to it. Before calling the accumulator, the two error terms of A and B are added up to form the new error term $c_{err} = a_{err} + b_{err}$. This new error term is then passed to the accumulator, along with the temporary array.

As mentioned above, passing roughly sorted input to the accumulator improves performance. So, instead of blindly concatenating the two sequences a_i and b_i, it is possible to merge them in order, assuming the high precision intervals A and B themselves are well conditioned. This is achieved by starting at the last limb of both sequences and successively copying the smaller of the two numbers into the temporary array. This yields, in linear time, a sequence of numbers ordered by increasing absolute value. If the input is not well conditioned, best performance of the accumulator would be obtained by a full sort of the input. In practice, however, the performance gain in the accumulator is too small to outweigh the cost of sorting the input.

For the subtraction $A - B$, the same algorithm as for addition is used, except all signs of the b_i are flipped. Since the set of floating point numbers is symmetric, i.e. if a is a floating point number so is $-a$, this operation is exact.

3.3.1.4 Multiplication

The multiplication is the core of our algorithms. Together with addition, this operation will be the most important and the most frequently performed operation. It is, therefore, vital to make this operation as efficient as possible.

Mathematically, it is clear what has to be done. To multiply two high precision numbers A and B one has to perform the following operation:

$$C = \left(\sum_n a_n\right) \cdot \left(\sum_m b_m\right) = \sum_{m,n} a_n b_m$$

It is obvious that, in general, there are $n \cdot m$ multiplications necessary to evaluate this expression. Each of these multiplications has to be done exactly, using a Dekker multiplication, and yields two resulting terms. Thus, in the end there are $2 \cdot n \cdot m$ terms to be summed up using the accumulator (3.3.1.2).

However, in practice it is possible to reduce the number of multiplications needed significantly. Once again this is due to the size of the error bound. The new error bound c_{err} is given by

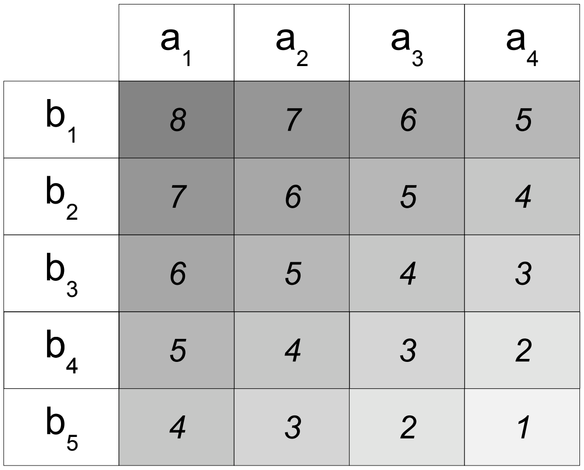
$$\begin{aligned} c_{err} &= \left(\sum_n a_n\right) \cdot b_{err} + \left(\sum_m b_m\right) \cdot a_{err} + a_{err} \cdot b_{err} \\ &\leqslant \left(\sum_n |a_n| + a_{err}\right) \cdot b_{err} + \left(\sum_m |b_m| + b_{err}\right) \cdot a_{err} \end{aligned}$$

As before, this new error bound is calculated first. Then, before performing a full Dekker multiplication, a simple floating point multiplication $x = a_i \cdot b_j$ of the two limbs a_i and b_j is performed. This product is then compared to c_{err}. There are three possible cases:

$|x| < c_{err}$ In this case, the result of the multiplication is smaller than the error term. Its absolute value can thus be rounded outwards and added to the error bound. No further processing is necessary.

$\varepsilon \cdot |x| < c_{err}$ In this case, the result of the multiplication does contribute to the final sum significantly. The roundoff error of the floating point multiplication, however, is below the error bound and thus can be estimated from above as $\varepsilon \cdot x$. The absolute value of this is then added to the error bound. Note that this is possible as long as we have round-to-nearest operations.

Else In all other cases, a full Dekker multiplication has to be performed. The result of this multiplication will consist of two floating point numbers, which have to be stored in a temporary array. This will eventually be handed off to the accumulator later. We also check wether the smaller of the two resulting numbers if it is below the error term. If so, we add the number to the error bound directly, thus reducing the number of elements that are passed to the accumulator.



	a_1	a_2	a_3	a_4
b_1	8	7	6	5
b_2	7	6	5	4
b_3	6	5	4	3
b_4	5	4	3	2
b_5	4	3	2	1

Figure 3.2: Graphical representation of the multiplication algorithm for two numbers of length 4 and 5 respectively. Every cell in the grid represents one product. The coloring of the cells shows products of the same order of magnitude, assuming that the input numbers where well conditioned. The numbering shows the order in which these products are evaluated, so that we take advantage of that ordering.

This way we can reduce the number of Dekker multiplications that are actually performed, significantly. It seems this results in an improvement in speed of about 5% – 10%. However, the full impact of these measures has to be determined by detailed performance analysis with real world examples at a later point.

Another way to reduce execution time is to “pre-split” the limbs of each number. In the worst case, every limb is multiplied by all limbs of the other number using a Dekker multiplication. If we split the numbers anew every time, we recalculate the same splitting over and over. Instead, the limbs are split once in the beginning and then stored in a local array. Whenever a multiplication takes place, it is performed using the pre-split limbs.

The last important issue with this algorithm is the order in which the limbs are multiplied. Clearly, the order of the multiplications is irrelevant, as long as it is ensured that every pairs is multiplied exactly once. The naive way would be to simply go straight through the first sequence a_i and multiply each limb with each b_j. However, if we want the input to the accumulator to be roughly ordered, it is better to multiply numbers that yield the same order of magnitude, starting with the lowest order (see 3.2). This is a little more involved, but not much more computationally expensive than the naive approach.

Note last that, if the input numbers were normalized, it would be sufficient to just multiply the first limbs and immediately add the products of higher limbs to the error term. Since we cannot assume that input is normalized, however, this method cannot be applied to our algorithm. But since we do compare each floating point product to the error term, our algorithm does not perform significantly slower.

3.3.1.5 Squaring

Analyzing the algorithm for multiplication given above, one finds that some steps in the process are redundant if both arguments are the same, i.e. if a number is squared. For one, the splitting only has to be performed for one number. Also, we can utilize the fact that a multiplication by 2 is always correct in floating point arithmetic[1].

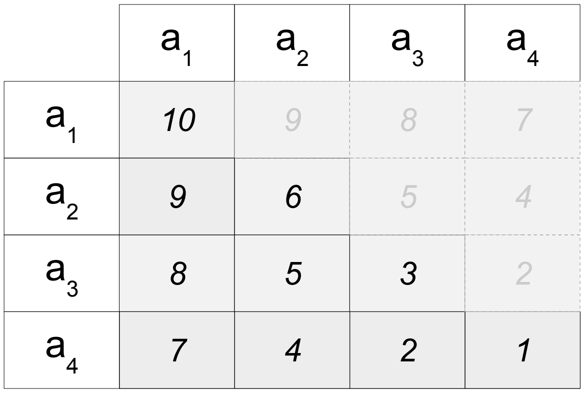
So, to calculate

$$\left(\sum_i a_i\right)^2 = \sum_{i,j} a_i a_j = \sum_i a_i^2 + \sum_{i \neq j} a_i a_j = \sum_i a_i^2 + 2 \sum_{j>i} a_i a_j$$

one simply goes through all indices i and, in an inner loop, lets j loop from $i+1$ to n. After the Dekker multiplication of the cross terms, one simply multiplies each part of the result by 2.

The algorithm is shown in 3.3. Note that, as in the regular multiplication, to keep the input to the accumulator at least roughly ordered, we evaluate products of the last limb first,

[1] Except for overflows, which we deal with separately.



	a_1	a_2	a_3	a_4
a_1	*10*	*9*	*8*	*7*
a_2	*9*	*6*	*5*	*4*
a_3	*8*	*5*	*3*	*2*
a_4	*7*	*4*	*2*	*1*

Figure 3.3: Graphical representation of the squaring algorithm for a number of length 4. Every cell in the grid represents one product. However, the off-diagonal products appear twice, so we can ignore half of them as shown in the illustration. The numbering shows the order in which these products are evaluated, so that we take advantage of well conditioned input.

and work our way up to the larger terms. While this only results in approximate ordering up to the 6th product, the speed gain by cutting the number of Dekker multiplications in half clearly outweighs the losses in the accumulator.

3.3.2 High Precision Taylor Models

Using similar techniques as those described in detail for High Precision intervals above, it is possible to implement the basic operations of addition and multiplication for Taylor Models. Here, too, each coefficient is stored as an unevaluated sum of double precision floating point numbers, which are operated on by the same type of Dekker based algorithms as in the case of High Precision intervals.

One such implementation was written by the author. It only operates on non-verified, high precision differential algebra vectors, storing each coefficient with a variable number of limbs as needed to achieve a pre-defined precision cut-off. To show the power of such a high precision implementation, we then used it to compute the transfer maps of various beam optical elements to about 60 accurate digits (see [44]).

Another fully verified implementation of these two operations for the High Precision Taylor Model datatype in the upcoming COSY INFINITY version 10 was done by Kyoko Makino[36]. In this implementation, each coefficient in the polynomial is itself stored as a polynomial with the limbs of the high precision representation of each coefficient as its coefficients. A detailed description of the algorithms used in this implementation is beyond the scope of this dissertation. Instead, we refer to the following publications describing the existing double precision Taylor Model data type[28, 4, 7, 31], as well as a forthcoming publication devoted entirely to the implementation of the new High Precision Taylor Models.

3.4 High Precision Intrinsic Functions

Intrinsic functions form an essential part of every scientific computing package. Arithmetic operations only form the very basis upon which a complex and diverse set of intrinsic functions is built. Almost all practical applications in physics and engineering require some form of trigonometric or exponential functions.

In this section, we will build upon the basic arithmetic operations addition, subtraction and multiplication described above to derive methods valid for both High Precision Intervals as well as the constant parts of High Precision Taylor Models.

With this information, it will be possible to implement a complete High Precision Interval datatype based on Dekker operations. To fully implement a working High Precision Taylor Model datatype, the algorithms for the constant part provided here need to be integrated into the algorithms already fully developed by Kyoko Makino for regular double precision Taylor Models in her thesis[28].

For simplicity, in the following we will collectively refer to both High Precision Taylor Models with only a constant part and Intervals as High Precision Numbers. If a distinction must be made between the two, this will be made clear in the text.

3.4.1 General Concepts

Intrinsic functions of a High Precision Number X are implemented using either an iterative Newton method or by the Taylor expansion of the intrinsic function. Newton methods are used mainly for the square root, as here the Newton iterations converge very quickly, and underlying floating point operations can be utilized to easily obtain a good starting values. To evaluate the other intrinsics, such as exponential, logarithm, or trigonometric functions and their inverse, we rely on the Taylor expansion of the respective functions, or we reduce the intrinsic to an evaluation of another intrinsic based on arithmetic identities.

3.4.1.1 Taylor Expansion with Remainder

With the Taylor expansion, we successively calculate each order, and add it to the result. By keeping the previous order term after adding it to the result, the cost of calculating the next order remains constant, so the computational cost of evaluating the series is linear in the order of the expansion.

To decide when to terminate the expansion, we generally use the current order term. Since in any useful Taylor formula the terms are monotonously decreasing, once the next order term becomes smaller than the error bound of the result, the expansion can be terminated. All further terms at that point would only contribute to the error term of the result.

This method can and is adapted depending on the requirements of the numerical environment. It is, for example, possible that due to certain external factors an intrinsic function does not need to be evaluated to full precision. This happens quite frequently when an argument is split (see below) and the boundary values are evaluated separately. In such cases, instead of using the remainder bound as a measure, a pre-set value can be used instead to terminate the series expansion in order to save computation time.

To rigorously estimate the error bound of the result, we use the Taylor remainder formula 3.5 where applicable. If that is not possible, we have to revert to other methods geared specifically towards the given expansion.

3.4.1.2 Argument Reduction and Splitting

To speed up convergence in the evaluation of the Taylor expansion, it is important to reduce the arguments so that they are close enough to the expansion point of the series to yield fast convergence. This is usually achieved by exploiting symmetries in the functions involved, as well as other analytical identities specific to each intrinsic function.

Another very important concept is argument splitting. In this context, argument splitting refers to the separate computation of the upper and lower bounds of wide arguments, which then are combined back into one single result using certain properties of the intrinsic in question, e.g. monotonicity. This approach helps with the fact that evaluating a Taylor expansion with wide intervals is subject to the usual interval overestimation due to repeated addition and subtraction in the Taylor expansion.

When determining wether to split an interval, various factors can be taken into account. For our purposes, we examine the size of the relative error and compare it to the relative computation precision multiplied by some value ξ. This ξ is a heuristically determined value for each intrinsic, which indicates how much wider than the computation precision an interval is allowed to be before it is split. Typical values for ξ lie in the range of 10^4 to 10^6.

For example, consider an intrinsic for which $\xi = 10^4$ being called with an argument 12 ± 10^{-30} with two limbs precision (i.e. relative computation precision ε of about 10^{-32}). The relative error of this argument is $\frac{10^{-30}}{12} \approx 10^{-31}$, which is less than $\varepsilon\xi = 10^{-28}$. Thus the argument would not be split. The argument 0.012 ± 10^{-29}, however, has a relative error of about 10^{-27} and thus would be split.

To speed up the computation in the case of splitting, it is furthermore possible to set a cutoff value for the intrinsic evaluation. Since error terms are only stored as a single double precision number, the maximum accuracy of the width of an interval is the precision of a double precision number, which is about 10^{-16}. Thus it is not useful to compute the intrinsic on the left and right boundary to higher precision than 10^{-16} times the expected width of the result.

For example, consider the exponential of the interval $[0, 1]$. The width of the resulting interval is roughly 1.7, so that the precision of the error variable is of order 10^{-16}. Computing the values of $\exp(0)$ and $\exp(1)$ to more than 16 significant digits is a waste of effort, as all this extra precision is lost in the computation of the width of the resulting interval.

The width of the final result can usually be estimated using a linear approximation of the function at the center point. To that end, the derivative is computed in non-verified double precision arithmetic, and then multiplied by the width of the interval. Even though this operation is not verified, the actual High Precision computation remains verified, but may not be optimally fast or sharp.

Depending on the requirements on the accuracy of intrinsics over large domains, it is even possible to set this cutoff higher than 10^{-16}, trading precision of the result versus computation time. In no case, however, is it useful to compute the result to more than 10^{-16} times the expected width significant digits.

3.4.2 Division

Division is somewhat different from the other algorithms presented so far. Instead of directly calculating the result using low level floating point operations of the processor, or using Newton Methods or Taylor expansions, we use a constructive approach that produces more and more valid limbs of the result in each step.

The algorithm is basically the same as the simple "black board" division (in the US also known as long division) that is taught in elementary school. It works by utilizing the division for floating point numbers to estimate the next limb. Then the rigorous multiplication and subtraction are used to calculate how far off this is from the correct result.

The idea is to first approximate the high precision intervals by floating point numbers, and then divide those using floating point arithmetic. That number will be the first limb in the result. Then we use high precision operations to calculate the difference between our floating point result times the denominator, and the numerator, i.e. the error left over. Now we continue this process with this difference to get the next limb and so on. Once we have reach a high enough accuracy, or the maximal number of limbs, we can terminate the process and immediately have a rigorous error bound at hand.

In mathematical terms, we want to evaluate $C = A/B$. Then we can reformulate the problem like this:

$$C = A/B = c + D' \tag{3.7}$$

where $c = \sum a_i / \sum b_i$ evaluated in floating point arithmetic, and D' is a high precision

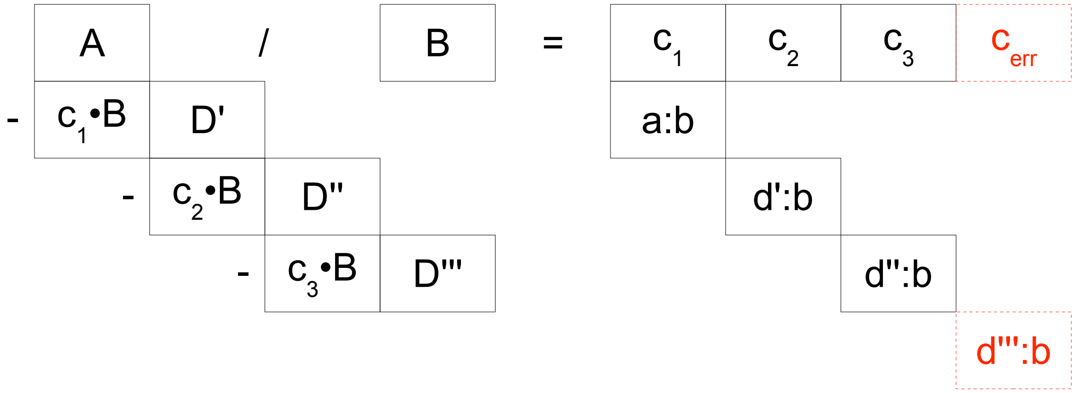


Figure 3.4: Graphical representation of the division algorithm for two numbers A and B. Capital letters represent high precision intervals, while the lower case letters are floating point numbers that represent the floating point sum of the limbs of the corresponding high precision interval. All arithmetic operations are Dekker operations, except for the floating point division, which is denoted by a colon (:).

interval or Taylor Model that rigorously represents the error. This error is, barring the possibility of cancellation during the summation of the high precision intervals, of order of $\varepsilon \cdot c$. Solving 3.7 for D' we obtain:

$$A/B = c + D' \;\Rightarrow\; A = Bc + D'B \;\Rightarrow\; \frac{A - Bc}{B} = D' \tag{3.8}$$

Let $D = A - Bc$. As noted before, D will usually be an order of ε smaller than A. To find the next limbs of the result, we have to determine D'. This, again, involves a division of the two high precision intervals D and B. The first limb of this can again be determined as described above, so one simply start over with the same algorithm, replacing A with D. This way one recursively obtains more and more limbs of the final result, while always keeping a rigorous error bound, namely D. This algorithm is graphically shown in 3.4.

The rigorous error D allows, of course, immediately the calculation of an error bound for the final result. When the algorithm has determined enough limbs of the result, one can obtain a floating point interval enclosure of the error D. The same can be done for the divisor B. Then, using straightforward floating point interval division, it is possible to obtain an upper bound on the error term for the result, since floating point division is guaranteed to be accurate up to $\epsilon/2$.

Of course, this algorithm needs to test the denominator for a division by zero. In our implementation, this is done by calculating a floating point interval approximation of the divisor B, which is then tested to not contain zero. If the divisor does contain zero, an error flag is set for the result, marking it invalid. For further remarks about checking for the inclusion of 0 in High Precision datatypes, see Section 3.5.

3.4.2.1 Argument Reduction and Splitting

The algorithm presented above does not require any splitting or argument reduction. This is due to the fact that neither addition nor multiplication require splitting, and that in each step, the value subtracted from the remaining error D is optimally sharp, resulting from a multiplication of High Precision Number B and the error-free floating point number c.

3.4.3 Square Root

To compute the square root X of a high precision number A, we use the interval Newton method[38] to obtain an interval enclosure of the solution of

$$X^2 - A = 0.$$

This yields the iteration step

$$X_{n+1} = M(X_n) - \frac{M(X_n)^2 - A}{2X_n} \tag{3.9}$$

where $M(X_n)$ stands for the midpoint of the interval X_n.

Provided that the initial interval enclosure X_0 contains the square root, each X_n contains the correct result. Furthermore, the sequence of intervals converges quickly, even without intersecting each X_n and X_{n+1}. The criterion for terminating this iteration is the width, i.e. the error bound, of X_n. Once the intervals do not contract any more, we have reached the best possible enclosure of the result.

The choice of the initial starting value X_0 is done in double precision interval arithmetic. First, an interval enclosure I of the high precision interval A is calculated (see 3.5.2). If this enclosure contains values less than or equal to 0, the resulting high precision interval is set to 0 and the error flag for that number is set. Otherwise, the interval square root of I is converted into a high precision number, which serves as the initial interval X_0.

Provided that the double precision interval operations are rigorous, the initial interval X_0 does contain the exact square root, and thus fulfills the requirements for the interval Newton algorithm. Note that the double precision floating point standard prescribes the same round to nearest accuracy requirements for the square root operation on double precision numbers as for the basic addition and multiplication,

On an implementational note, we remark that in our representation of High Precision Numbers it is particularly easy to obtain an the midpoint of an argument exactly. All that has to be done is to set the error bound to 0 temporarily. This way all further calculations are still performed in rigorous interval arithmetic, thus assuring that the resulting X_{n+1}

really contains the correct interval.

3.4.3.1 Argument Reduction and Splitting

Since the the square root is not differentiable at 0, the convergence of the Newton method very close to zero decreases. This will only really show once arguments get really close to 0, pushing the limits of our implementation. We therefore currently do not implement any argument reduction. It is, however, possible to do based on the identity

$$\sqrt{X} = \sqrt{2^{-2n}2^{2n}X} = \sqrt{2^{-2n}}\sqrt{2^{2n}X} = 2^{-n}\sqrt{2^{2n}X}$$

If n is chosen appropriately, the argument in the square root can be made large enough to be of order 1 and thus converging quickly using Newton's method. The multiplications and divisions by two are all easy to perform since they are exact in floating point arithmetic, thus producing no errors.

Argument splitting is also not necessary with interval Newton methods, as the method is contracting the entire interval. Overestimation is minimized by the Newton method's reliance on the mid point of intervals in most places.

3.4.4 Inverse Square Root

While it would be possible to calculate the inverse square root using another interval Newton method, we simply calculate the square root and then invert the result. This simplifies the code development and maintenance at a minimal cost compared to a full implementation of this operation based on another Newton method.

Note that all error handling, splitting and argument reduction is automatically inherited from the division and square root routines. Since neither one performs any of those operations, there is no performance penalty due to repeated argument reductions of splitting.

3.4.5 Exponential Function

To calculate the exponential of a High Precision number X, we simply use the probably best known Taylor expansion of all, the expansion of the exponential function around 0[12]:

$$\exp X = \sum_{n=0}^{\infty} \frac{X^n}{n!} = 1 + X + \frac{X^2}{2} + \frac{X^3}{6} + \dots \tag{3.10}$$

To estimate the remainder for that series, we use that

$$\left(\frac{d}{dy}\right)^n \exp(y) = \exp(y)$$

and

$$\sup(\exp((-\infty, 0.5]) = \exp(0.5) = \sqrt{e} \leqslant 2 \tag{3.11}$$

Thus, the Taylor remainder term of the exponential series computed up to order n for an argument $y \leqslant 0.5$ can be bounded above by twice the next term in the expansion:

$$|R_{n+1}| \leqslant 2 \cdot \left| \frac{X^{(n+1)}}{(n+1)!} \right| \tag{3.12}$$

This yields a rigorous enclosure of the value $\exp(y)$. The range of the bound in equation 3.11 is chosen such that it includes the argument after the following argument reduction is performed.

3.4.5.1 Argument Reduction

Since the exponential function for large arguments obviously grows very fast, it is necessary to reduce the argument to a small value to ensure convergence in reasonable number of terms and thus reasonable computation time. To achieve that, we make use of the identity

$$\exp(x \cdot 2^k) = \exp(x)^{(2^k)} \tag{3.13}$$

First notice that $\exp(700) > 10^{304}$, which is near the largest representable double precision floating point number. Similarly, of course, $\exp(-700) < 10^{-304}$ which again is near the smallest representable double precision floating point number. We thus somewhat arbitrarily limit the domain on which the exponential function is computed to $D = [-650, +650]$. If a value is larger than 650, it will produce an error in the form of a defective result, while a value smaller than 650 will always return a predetermined interval $[0, d]$ where $d > 10^{-280}$. This value is determined by considering that

$$\exp(-650) = \exp(-65)^{10} < (10^{-28})^{10} = 10^{-280},$$

where $\exp(-65)$ is bounded using our very own algorithm described here, and checked using a different verified interval package (such as MPFI[40]).

Now, given some value $x \in D$, we define $y = x \cdot 2^{-11}$. This ensures that $|y| \leqslant 650/2048 < \frac{1}{2}$. By 3.13 we then have

$$\exp(x) = \exp(y \cdot 2^{11}) = \exp(y)^{(2^{11})}.$$

Because of the choice of the factor 2^{-11}, y is of sufficiently small size such that the Taylor expansion for $\exp(y)$ converges relatively quickly. As a criterion for terminating the expansion, we use the size of the n-th computed coefficient of the expansion. If that coefficient is less than the error bound of the result, all further coefficients will merely increase the error bound, and thus the expansion can be truncated at that point. Other conditions for terminating the expansion earlier can be considered, based on the external circumstances.

In order to obtain the final result for $\exp(x)$, the result of the series evaluation has to be raised to the power of $2^{11} = 2048$. Naively, this can be done very fast since for any x

$$x^{(2^n)} = \left(x^{\left(2^{n-1}\right)}\right)^2 = \left(\left(x^{\left(2^{n-2}\right)}\right)^2\right)^2 = \cdots = \left(\left(\ldots\left(x^2\right)^2\ldots\right)^2\right)^2$$

To evaluate $\exp(y)^{(2^{11})}$ one simply squares $\exp(y)$ repeatedly 11 times. Since squaring is the second cheapest operation in our arithmetic, these operations incur no performance penalty.

However, such a naive evaluation produces large relative overestimation. This is because for a typical exponential argument of order 1, the exponential of the reduced argument is of order

$$\exp(2^{-11}) = \exp(\frac{1}{2048}) \approx 1.00048\ldots.$$

When such a value is squared, the result is still of order 1, but the error has doubled in size because

$$(1.00048 \pm err)^2 = 1.00048^2 \pm 2 \cdot 1.00048 \cdot err + err^2 \approx 1.00048^2 \pm 2 \cdot err$$

for small err. Thus the relative error has doubled in size.

Squaring 11 times, the relative error in each step grows by a factor of 2, leading to a relative error of the final result of $2^{11} = 2048$ times the size of the initial relative error. In practice the Taylor expansion evaluation of the reduced argument yields an error roughly the size of the selected precision. For a two limb computation of exp(1), for example, that error would be on the order of 10^{-30}. After the squaring, however, the resulting error bound will be on the order of $2048 \cdot 10^{-30} = 2 \cdot 10^{-27}$.

In order to avoid this effect, we compute $z_1 = \exp(y) - 1$ instead of $\exp(y)$ by simply omitting the constant term in the Taylor expansion of the exponential function. We then observe that

$$\exp(y)^2 = (1 + z_1)^2 = 1 + 2 \cdot z_1 + z_1^2. \tag{3.14}$$

Letting $z_2 = 2 \cdot z_1 + z_1^2$, note that the relative error of z_2 remains about the same as that of z_1. This is because the multiplication of z_1 by a factor of 2 of course doubles both the error term but also the value of the center point. Thus the ratio remains the same. For small values of z_1 the term z_1^2 is so small that it does not contribute significantly to the value or the error of z_2.

Repeating the squaring operation as described in equation 3.14 eleven times, keeping the 1 separate from the result in each step and only actually adding it in the very end, finally yields the desired result for $\exp(x)$ with a relative error of the same order of magnitude as the initial evaluation of the Taylor expansion.

3.4.5.2 Argument Splitting

In order to suppress the effects of the interval dependency problem in the evaluation of the Taylor expansion for the exponential function, we split large intervals into two separate High Precision numbers X_u and X_l for the upper and lower bound respectively. Then the exponential function is evaluated as described above separately for both X_u and X_l, yielding $Y_u = \exp(X_u)$ as well as $Y_l = \exp(X_l)$.

Due to the monotonic nature of the exponential function, the final interval is then computed by merging Y_u and Y_l into one interval covering the range $[Y_l, Y_u]$.

3.4.6 Natural Logarithm

To evaluate the natural logarithm of a high precision number X, we utilize the following expansion which can be derived by integrating the Taylor series of the derivatives of $\log(x+1)$ and $\log(x-1)$ and subtracting them from each other[12]:

$$\ln X = \sum_{n=0}^{\infty} \frac{2}{2n+1} \left(\frac{X-1}{X+1}\right)^{2n+1} = 2\frac{X-1}{X+1} + \frac{2}{3}\left(\frac{X-1}{X+1}\right)^3 + \frac{2}{5}\left(\frac{X-1}{X+1}\right)^5 + \ldots \quad (3.15)$$

Since X has to be positive for the logarithm to be defined, the argument for the series is always less than or equal to 1, and therefore the sequence is monotonously decreasing. That allows us to use the current order as a termination criterion for the evaluation of the series.

To estimate the remainder of the natural logarithm, we use the following expression for the remainder bound to the first n terms in the series given in 3.15:

$$|R_{n+1}| \leqslant \left| \frac{(X-1)^2}{2X} \left(\frac{X-1}{X+1} \right)^{2n} \right| \tag{3.16}$$

For purely implementational purposes, we slightly rearrange this remainder term so that we can use as many of our already computed quantities as possible:

$$|R_{n+1}| \leqslant \frac{(X-1)^2}{2|X|} \left(\frac{X-1}{X+1} \right)^{2n} = \frac{1}{4|X|} \cdot 2 \left(\frac{X-1}{X+1} \right)^{2n+1} (X^2-1) \tag{3.17}$$

At the end of the series evaluation, we already calculated $2\left(\frac{X-1}{X+1}\right)^{2n+1}$. All that remains to do is to divide by 4 and X (which is positive) and multiply the result by (X^2-1).

3.4.6.1 Argument reduction

Obviously, this series converges the faster the closer X is to 1. In particular, it converges very slowly for X close to 0, or large values of X. We hence reduce the argument to be close to 1 making use of the identity:

$$\ln(x) = m\ln(2) + \ln(2^{-m}x) \tag{3.18}$$

Choosing m to be the nearest integer to the 2 based logarithm, i.e.

$$m = \lfloor \ln(x)/\ln(2) + 0.5 \rfloor,$$

the new argument satisfies the inequality $|2^{-m}x - 1| \leqslant \frac{1}{3}$. For those arguments, the Taylor expansion converges relatively quickly. Note that even if m is not chosen exactly as specified above, all operations remain verified, they are just not optimally fast any more. Therefore it is sufficient to compute m using rather crude floating point approximations without any verification.

For this, it is necessary to once compute an accurate enclosure of the value of $\ln(2)$. This is done by simply evaluating the series given above while skipping the argument reduction step. This value is then stored internally for later use so it does not need to be recomputed every time the logarithm is called.

3.4.6.2 Argument Splitting

As with the exponential function before, we also split wide arguments to the logarithm in order to suppress the effects of the interval dependency problem in the evaluation of the Taylor expansion. We split wide arguments into two separate High Precision numbers X_u and X_l for the upper and lower bound respectively. Then the logarithm is evaluated as described above separately for both X_u and X_l, yielding $Y_u = \log(X_u)$ as well as $Y_l = \log(X_l)$.

Again, due to the monotonic nature of the logarithm, the final interval is then computed by merging Y_u and Y_l into one interval covering the range $[Y_l, Y_u]$.

3.4.7 Sine and Cosine

We use the well known expansions of the sine and cosine functions[12]:

$$\sin X = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} X^{2n+1} = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \dots \tag{3.19}$$

$$\cos X = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} X^{2n} = 1 - \frac{X^2}{2!} + \frac{X^4}{4!} - \dots \tag{3.20}$$

$$R_n = \frac{X^{n+1}}{(n+1)!} \tag{3.21}$$

In the remainder term we used that $\left(\frac{d}{dx}\right)^{n+1} \sin(x) \leqslant 1$ as well as $\left(\frac{d}{dx}\right)^{n+1} \cos(x) \leqslant 1$.

Computationally, it is advantageous to compute both the sine and the cosine simultaneously. The overhead needed is rather small, and in the implementation of the Taylor Model sine and cosine, it is necessary to compute both values of the same argument anyway. Another advantage is that the argument reduction, which is rather complicated for these functions, only has to be performed once.

3.4.7.1 Argument reduction

The argument to the sine and cosine is reduced in several steps. First, we exploit the periodicity of the sine and cosine functions, $\sin(x) = \sin(x + 2\pi)$ and $\cos(x) = \cos(x + 2\pi)$. In a first step, this allows us to reduce the argument into the interval $[-\pi, \pi]$ using the equation

$$k = \left\lfloor \frac{x}{2\pi} + \frac{1}{2} \right\rfloor$$

This reduction is performed using a floating point approximation x of the value of the argument X as well as a floating point approximation of π to estimate the number k of integer multiples of 2π to subtract from the argument. While this approach does not guarantee that the argument rigorously is reduced to the interval $[-\pi, \pi]$, the result is still verified since a wrong estimate here will only increase the computation time, but not affect the correctness of the result.

The value of 2π itself, which is subtracted from the argument, is of course computed as a rigorous enclosure. This computation only happens once using the equality $2\pi = 8 \cdot \arctan(1)$. Similarly, we precompute values for π and $\pi/2$. All of these values are stored after their first use for later use.

In the second argument reduction step, we make use of the symmetry that $\sin(x) = -\sin(x+\pi)$ and $\cos(x) = -\cos(x+\pi)$. If after the first reduction step the result is negative, i.e. in the interval $[-\pi, 0]$, a rigorous enclosure of π is added to the argument to bring it into the interval $[0, \pi]$. If this step is performed, the final results are negated once they are computed.

The third step reduces the argument to the interval $[0, \pi/2]$ by using the identities $\sin(x) = \sin(\pi - x)$ and $\cos(x) = -\cos(\pi - x)$ respectively. If the argument is larger than $\pi/2$, i.e. in the interval $[\pi/2, \pi]$, it is subtracted from π, and the sign of the cosine result is flipped.

In the very last step, the reduced argument is divided by 3 and result is called t. Then the actual Taylor expansions of the sine and cosine are evaluated. To undo the last division by 3, the following identities are used:

$$\begin{aligned} \sin(3t) &= 3\sin(t) - 4\sin^3(t) \\ \cos(3t) &= 4\cos^3(t) - 3\cos(t) \end{aligned} \tag{3.22}$$

Note that using similar equations for higher multiples it would be possible to reduce the argument even further. However, these equations become increasingly complicated and the additional operations introduce further computational errors. In our experience, using the triple angle formula given above provides good accuracy and reasonably fast evaluations.

To obtain the sharpest possible results for the cosine, we replace the multiple angle equation given above by a slightly modified equation. The problem here is the same as with the squaring operations in the exponential function (see Section 3.4.5.1): While the sine of t is a number relatively close to 0, the cosine of t is a number of the form $1+\varepsilon$, where ε is a small number compared to 1. Computing the cube of such a number increases the relative error of the result significantly.

Instead of computing the expression $\cos(t)$, we proceed as before with the exponential function and compute $y = \cos(t) - 1$ instead by simply omitting the first term of the Taylor expansion. We then rewrite equation 3.22 in terms of y:

$$\cos(3t) = 4\,(y+1)^3 - 3(y+1) = 4y^3 + 12y^2 + 9y + 1.$$

Evaluating this expression instead, the relative error of the result does not increase significantly.

3.4.7.2 Argument splitting

Proper argument splitting for the sine and cosine is more complicated than for other intrinsic routines. The periodicity of these functions requires more complicated tests when assembling the final interval from the results of the left and right boundary.

First off, a plausibility test is performed to test if the width of the interval is greater than a floating point approximation of π. If that is the case, the interval $[-1, 1]$ is returned immediately without further computation. Since $[-1, 1]$ is always a rigorous enclosure of the sine or cosine, it does not matter that the comparison is done in non-verified floating point arithmetic.

If the argument passes this first test, the sine and cosine splitting for wide arguments begins with computing two separate High Precision numbers X_u and X_l for the upper and lower bound respectively. These can be easily computed by rigorously adding and subtracting the remainder bound from the High Precision number, resulting in rigorous enclosures of X_u and X_l.

Then the sine and cosine are evaluated as described above separately for both X_u and X_l, yielding $Y_u = \sin(X_u)$ and $Z_u = \cos(X_u)$ as well as $Y_l = \sin(X_l)$ and $Z_l = \cos(X_l)$.

The sine function attains its maximum at $\frac{\pi}{2} + k \cdot 2\pi$ and its minimum at $\frac{3\pi}{2} + k \cdot 2\pi$ where in both cases $k \in \mathbb{Z}$. Inbetween these values, the sine function is monotonic, since the derivative is non-zero there. To determine the final result from the results at the two endpoints Y_u and Y_l it is therefore necessary to test if the argument contains a value of the form $\frac{\pi}{2} + k \cdot 2\pi$ (or $\frac{3\pi}{2} + k \cdot 2\pi$). If so, the upper bound Y_u (or lower bound Y_l) need to be replaced by the value 1 (or -1). Then the resulting High Precision number can be computed by merging Y_u and Y_l into one interval covering the range $[Y_l, Y_u]$.

Similarly for the cosine the maximum is attained at $k \cdot 2\pi$, while the minimum is attained at $\pi + k \cdot 2\pi$. By the same method as for the sine applied to these values and the bounds Z_u and Z_l, the final result of the cosine is readily computed.

It remains to actually implement the test wether a High Precision number contains a value of one of the above forms. We will describe here how to perform the test for the maxima of the sine function at $\frac{\pi}{2} + k \cdot 2\pi$. The test for the other three cases is absolutely analogous. First, the un-split argument X is linearly transformed as follows

$$J = \frac{X}{2\pi} - \frac{1}{4} \tag{3.23}$$

where a rigorous High Precision enclosure of 2π is used in the denominator. By inserting the expression for the location of the maxima into equation 3.23, it is clear that whenever J contains an integer, then X contains a location of a maximum of the sine function. Note that the choice of the value $-\frac{1}{4}$ is of course not unique, any value of the form $-\frac{1}{4} + k$ for $k \in \mathbb{Z}$ will work.

Next, a floating point approximation of J is computed, and truncated to the nearest integer, the representation of which is precise in double precision floating point arithmetic. If non-zero, this value is then subtracted from J. Clearly, if J contains an integer, so does $J - k$ for any $k \in \mathbb{Z}$. If the approximation of J was sufficiently accurate, this step reduces J to lie in the interval $\left[-\frac{1}{2}, \frac{1}{2}\right]$, because in the beginning we already eliminated intervals wider than 2π. Note that even if the approximation was not accurate, the result is still mathematically rigorous, but J may lie outside this interval.

In the last step, J is bounded using double precision intervals. The resulting upper and lower bounds are then tested for inclusion of zero using simple floating point comparisons. In case the approximation in the previous step was not accurate, we also test for inclusion of any other integer. If any of these tests are true, then J may contain an integer. If they are all false, J certainly does not contain an integer.

As described in Section 3.5.2, the test for zero is very accurate despite the use of double precision intervals in the last step. This is the reason why we perform the previous reduction of J to the interval $\left[-\frac{1}{2}, \frac{1}{2}\right]$.

The remaining maxima and minima tests are performed analogously, adjusting the value $-\frac{1}{4}$ in equation 3.23 accordingly to e.g. $\frac{1}{4}$ for the minima of the sine function, 0 for the maxima of the cosine function and $-\frac{1}{2}$ for the minima of the cosine function. Since all these tests require the computation of $\frac{X}{2\pi}$, this value is only computed once and then reused in each test.

3.4.8 Tangent

The Taylor expansion of the tangent is not as trivial as that of the sine and cosine. Furthermore, the argument checking and splitting required for the periodic trigonometric functions is the most complicated of our available intrinsic functions. Thus instead of reproducing the effort and implementation for the tangent Taylor expansion, we rely on the definition of the tangent:

$$\tan(X) = \frac{\sin(X)}{\cos(X)} \tag{3.24}$$

This requires one evaluation of each the sine and the cosine, but the performance impact is greatly reduced by the combined evaluation of the sine and the cosine as described above. Through this, all argument reductions are only performed once, by those routines. Similarly, the argument splitting is also left to the sine/cosine routine.

There is no argument splitting needed for the final division, as the division is fully functional for large arguments as well.

3.4.9 Arc Sine and Arc Cosine

Both the arcsine and arccosine functions rely on the arctangent, as given by the identity[12]:

$$\begin{aligned} \arcsin X &= 2 \arctan \frac{X}{1+\sqrt{1-X^2}} \\ \arccos X &= \frac{\pi}{2} - \arcsin X \end{aligned} \tag{3.25}$$

In the computation of the argument of the arctangent, neither argument splitting nor argument reduction is necessary, as all involved operations, the square root, division, addition and multiplication, are fully functional on their own without argument splitting.

Furthermore, no testing of the argument to lie within the domain $D = [-1, 1]$ of the arcsine or arccosine is necessary. This test is automatically performed by the square root routine when computing the value of $\sqrt{1-X^2}$ in the argument for the arctangent. If this square root produces an arithmetic error, the argument is outside of the allowed domain, and an error is returned immediately without further computations.

Note that for the arccosine, a pre-calculated, rigorous interval enclosure is used for the value of π. This constant is computed once using the identity $\pi = 4 \cdot \arctan(1)$ and then stored for future use. All verification and argument splitting and reduction is left to the arctangent function.

3.4.10 Arc Tangent

For the arctangent we use the following expansion[12]:

$$\arctan X = \sum_{n=0}^{\infty} (-1)^n \frac{X^{2n+1}}{2n+1} = X - \frac{1}{3}X^3 + \frac{1}{5}X^5 + \dots \tag{3.26}$$

This series converges for $|X| \leqslant 1$.

To obtain the error bound, it is unfortunately not very useful to use the Taylor remainder term. Careful examination of the expression $\sup \left(\frac{d}{dx}\right)^n \arctan x$ shows that it grows quickly, so that the Lagrange remainder term only converges as $\frac{1}{n}$. This overestimates the error by several orders of magnitude.

Instead, we use the fact that within the radius of convergence, this series has monotonously decreasing elements and alternating signs. Thus the Leibniz criterion (Section 3.1.4) applies, and an estimate of the remainder of the partial series after n terms is given by:

$$R_n = \left| \frac{X^{2n+3}}{2n+3} \right| \tag{3.27}$$

3.4.10.1 Argument Reduction

To reduce the argument to a range where the above expansion is valid, we make use of the following identity[12]:

$$\arctan(X) = 2 \arctan \frac{X}{1 + \sqrt{1 + X^2}} \tag{3.28}$$

Applying this equation once, reduces all arguments from $(-\infty, +\infty)$ to $(-1, 1)$. This is clear since we always have that

$$\left| \frac{X}{1 + \sqrt{1 + X^2}} \right| < 1$$

and

$$\lim_{x \to \pm\infty} \frac{X}{1 + \sqrt{1 + X^2}} = \pm 1.$$

Convergence of the series on $(-1, 1)$ is still rather slow, thus applying the same reduction repeatedly n times ensures that the result lies in a range that converges quickly. After evaluating the series for an argument that has been reduced n times, only a multiplication by 2^n is necessary. This is, even in floating pint arithmetic, an exact operation and can be easily done.

It is, however, not desirable to perform too many reduction steps, as each reduction causes overestimation and thus increases the remainder bound of the result. We found that $n = 4$ yields good results for the convergence speed, while still maintaining a sharp error bound. After the second application, the argument lies within $\left(-\frac{1}{1+\sqrt{2}}, \frac{1}{1+\sqrt{2}}\right) \approx (-0.41, 0.41)$, the third application takes it to approximately $(-0.2, 0.2)$ and the forth application reduces the argument to the final range of about $(-0.1, 0.1)$.

To undo the argument reduction after the evaluation of the Taylor expansion, a simple multiplication of the result by 16 is required.

3.4.10.2 Argument Splitting

Wide arguments to the arctangent are split in order to suppress the effects of the interval dependency problem in the evaluation of the Taylor expansion. We split wide arguments into two separate High Precision numbers X_u and X_l for the upper and lower bound respectively. Then the arctangent is evaluated as described above separately for both X_u and X_l, yielding $Y_u = \arctan(X_u)$ as well as $Y_l = \arctan(X_l)$.

Due to the monotonic nature of the arctangent, the final interval is then computed by merging Y_u and Y_l into one interval covering the range $[Y_l, Y_u]$.

3.4.11 Hyperbolic Sine and Cosine

Transforming the well known form of the hyperbolic sine and cosine into an equal, yet computationally more favorable form, we obtain the relations

$$\begin{aligned} \sinh X &= \frac{\exp(X) - \exp(-X)}{2} = \frac{\exp(X)^2 - 1}{2\exp(X)} \\ \cosh X &= \frac{\exp(X) + \exp(-X)}{2} = \frac{\exp(X)^2 + 1}{2\exp(X)} \end{aligned} \tag{3.29}$$

These equation can be evaluated in the verified High Precision arithmetic to directly obtain the result. As can be seen, we completely rely on the expression of the hyperbolic functions in terms of the real exponential function[12]. The overhead due to this is minimal, since each hyperbolic function only requires one evaluation of the exponential function.

On the other hand, we automatically gain all verification, argument checking, argument reduction and argument splitting that is performed by the exponential routine. The computation of the final result of the hyperbolic functions does not require argument splitting, as it only involves operations that perform equally well for wide as for narrow arguments (addition, multiplication, division).

3.4.12 Hyperbolic Tangent

The hyperbolic tangent is slightly more involved than the other two hyperbolic functions. This is because of its asymptotic behavior. First, we again express the hyperbolic tangent by an exponential function:

$$\tanh X = \frac{\exp(X) - \exp(-X)}{\exp(X) + \exp(-X)} = \frac{\exp(X)^2 - 1}{\exp(X)^2 + 1} \tag{3.30}$$

Note that there is no benefit in rewriting $\exp(X)^2$ as $\exp(2X)$, as the argument reduction in the exponential function would just remove this power of two again. Instead, it is faster to square the result directly.

As in the preceding cases, this equation can be evaluated directly in high precision interval arithmetic. However, such a naive evaluation produces rather large remainder bounds for positive arguments. This is because for even moderately large values of X, the expression $\exp(X)^2$ becomes quite large quickly. Thus, even if the relative error remains optimally sharp, in absolute value the error of the two terms $\exp(X)^2 - 1$ and $\exp(X)^2 + 1$ is rather large. The result of their division, of course, is almost 1, but due to the large absolute value of the error bounds, the resulting error bound will be large as well.

3.4.12.1 Argument Reduction

To prevent the effect described above from happening, we introduce a very simple argument reduction based on the following symmetry of the hyperbolic tangent:

$$\tanh(-x) = -\tanh(x)$$

Whenever a positive argument is evaluated, we replace it with its negative and after the evaluation negate the result. Especially for large positive arguments, this reduces the remainder bound of the result drastically.

We also test the incoming argument to lie within the somewhat arbitrarily chosen interval $[-325, 325]$ (half of the interval chosen for similar purposes in the exponential function in Section 3.4.5). For values outside of this interval, we do not compute a result, but instead just return the value ±1 with a sufficiently small error bound (e.g. 10^{-50}). This prevents calling the exponential function with an argument that is too small to be evaluated.

3.4.12.2 Argument Splitting

While not strictly necessary, we also perform the usual argument splitting for monotonic functions. Wide arguments are split in order to suppress the effects of the interval dependency problem in the evaluation of the Taylor expansion. We split wide arguments into two separate High Precision numbers X_u and X_l for the upper and lower bound respectively. Then the logarithm is evaluated as described above separately for both X_u and X_l, yielding $Y_u = \tanh(X_u)$ as well as $Y_l = \tanh(X_l)$.

Since the hyperbolic tangent is monotonic, the final interval is then computed by merging Y_u and Y_l into one interval covering the range $[Y_l, Y_u]$.

3.5 Comparisons and Bounding

The following section describes algorithms for bounding and comparisons of high precision intervals. For the bounding operations we obtain an upper and lower bound as a double precision number. The comparison operations allow rigorous comparisons between two high precision intervals, or a high precision interval and a double precision number. Due to the interval nature of the high precision intervals, the meaning of the comparison operation also has to be specified.

3.5.1 Approximation

In our implementation of intrinsic functions, it is often required to compute a floating point approximation of the value of the center point a High Precision number. There are of course various ways to obtain such an approximation, we will briefly discuss three different approaches here.

1. The fastest method to estimate the value of a High Precision number is to only consider the value of the first limb. For well conditioned High Precision numbers, this will be a good approximation of the actual value, as all higher limbs are considered to be corrections to the preceding ones. However, for ill conditioned numbers this method can produce completely wrong results.

2. The second method involves the simple summation of all limbs in double precision arithmetic. For well conditioned numbers, this summation will not change the result significantly from the preceding method, but in case of ill conditioned numbers this method provides at least some protection against completely misleading approximations. For added accuracy, the summation should be performed starting with the last, and usually smallest, limb, working its way up to the first, and largest, limb.

3. The third method is similar to the second, but instead of a naive summation of double precision floating point numbers, a Kahan summation (see Section 3.1.5) is performed. This provides additional safeguards against ill conditioned High Precision numbers being approximated poorly. As in method number two, the summation should be performed backwards, starting with the last limb of the number.

In our implementation, we decided to use the Kahan summation algorithm to compute approximations of High Precision numbers. Its accuracy is far better than the other two approaches, especially for not well-conditioned numbers. Such numbers regularly occur when there is heavy cancellation during a computation, e.g. when representing an identity in a very complicated way such as $\sin^2(x) + \cos^2(x)$ or $\log(\exp(x))$. The additional cost of such a summation is negligible compared to the cost of performing the other high precision operations.

3.5.2 Bounding

The bounding operation essentially converts a high precision interval into a double precision interval rigorously containing the high precision interval. The upper and lower bound of the result are, of course, upper and lower bounds for the high precision number.

To calculate such an interval enclosure the unevaluated sum $A = \sum_i a_i \pm a_{err}$ is evaluated in double precision interval arithmetic. The initial interval is just the error bound $[-a_{err}, a_{err}]$. Then each limb is added to this interval. To avoid excessive and unnecessary overestimation, the summation is carried out backwards, beginning with the last, and usually smallest, limb.

This method of bounding is particularly accurate for High Precision numbers close to zero. The reason for this is that in the floating point model, zero is an "accumulation point" of sorts. The density of floating point numbers around zero is much higher than around any non-zero number. This fact allows sharp bounds for values very close to zero without the overestimation typically experienced near non-zero values.

Consider for example a bound for the High Precision number $A = 1 - 10^{-20} + 10^{-40}$. This is a number very close to, but not exactly 1. In double precision arithmetic, even the most accurate double precision interval enclosure of A would be given by an interval of the form $[1 - 10^{-16}, 1]$, because there are no double precision floating point numbers closer to the real value of A. Consider, on the other hand, a similar number near 0 instead of near 1, e.g. $B = 0 - 10^{-20} + 10^{-40}$. Then the double precision interval enclosure would be much better and correctly reflect the fact that B is strictly negative and bounded away from zero. This is simply because there are plenty floating point numbers between 10^{-20} and 0, despite their small absolute distance from each other.

3.5.2.1 Absolute Value

The interval nature of high precision intervals also requires a new definition of the absolute value. For this operation, we follow the currently existing implementation of double precision intervals in COSY INFINITY. This implementation returns a single double precision number that is guaranteed to be an upper bound on the absolute values of every element in the interval.

Definition 3.7 (Absolute Value)**.** The absolute value of a High Precision number A returns one double precision number v such that

$$v \geqslant |x| \; \forall \; x \in A$$

Note that there is no requirement for v to be in any way optimal. Any v that satisfies the condition in Definition 3.7 is acceptable. To make this intrinsic useful, however, it is clearly in the interest of the implementation to avoid large overestimation.

In our implementation of High Precision numbers, the upper bound v is calculated by first obtaining a double precision interval enclosure of the high precision number A (see 3.5.2). Then v is set to be the larger of the absolute values of the upper and lower bounds of the double precision interval. Since the absolute value is an exact operation in floating point arithmetic, this result satisfies the above definition.

For well conditioned High Precision numbers, or in cases where the sharpness of the absolute value is not absolutely essential, a much faster method to compute a suitable value for v can be employed. Since certainly

$$\left|\sum_i a_i \pm a_{err}\right| < \sum_i |a_i| + a_{err},$$

we can just add up the absolute values of each limb and the error term and round the result outward in double precision arithmetic. If the input is well conditioned, the resulting bound will still be quite sharp, since all the smaller corrections just cause round-off in the last few bits of the result. In case of ill conditioned input, however, this method can produce significant overestimation.

3.5.3 Splitting and Merging

In various places throughout the intrinsics it will become necessary to split High Precision numbers into a sharp upper and lower bound, each again given as a High Precision number. Conversely, an operation merging two High Precision numbers into one which covers the entire interval of values between them is needed. These operations are of particular interest for the argument splitting which is performed for wide interval argument.

In the following, we will describe two simple and fast algorithms to perform these operations.

3.5.3.1 Splitting

Let $A = \sum a_i \pm a_{err}$ be a given High Precision number with limbs a_i and error term a_{err}. In order to compute an upper and lower bound of A, the floating point error term a_{err} is either added or subtracted from the center point $A' = \sum a_i$, respectively. This operation is performed in the usual verified High Precision arithmetic, so the result is rigorous.

Furthermore, the error bound on the resulting number is on the order of the computation precision. This is because neither A' nor a_{err} have an error term of their own, so any round-off errors that appear are only due to the limited precision.

This means that, obviously, for error bounds smaller than the relative computation precision of A this operation has no effect at all and returns the same result for both upper and lower bound.

3.5.3.2 Merging

The merging operation, or mathematically speaking the interval union, of two intervals A and B can be performed as follows.

First, note that the width W of the resulting interval is mathematically given by

$$W = \frac{|A - B|}{2},$$

where the absolute value of the interval is meant to be the largest absolute value of all values in the interval $A - B$. Furthermore, the new center point C can be computed as

$$C = \frac{A + B}{2}$$

In High Precision number arithmetic, these operations can be performed rigorously, and W can be bounded from above by a floating point number w using any of our bounding schemes. Then the resulting interval $C = A \cup B$ is simply given by adding w to the error term of C with outward rounding.

This method is very fast, and works well for merging sharp intervals, i.e. A, B with small remainder errors. Another big advantage of this method is that it does not require any previous knowledge about A and B. In particular, it is not necessary to compare both values to determine which one is the larger and which is the smaller.

The drawback of this method is that the over-estimation of the result is quite large if the intervals to be merged themselves have large error bounds. While the expression for W still provides the correct width of the resulting interval, the expression for C will have an error bound roughly the size of the average of the individual error bounds of A and B. As this is then added to W, the resulting width is not optimal any more if the error of C was large.

This problem is due to the fact that the optimal center point of the resulting interval depends on the widths of A and B. Ideally, the center point would be computed up to the computation precision. However, such a computation is much more involved, requiring the splitting of both A and B into upper and lower bounds and then merging the maximum and minimum of each using e.g. the above algorithm.

Fortunately, in our use of the intrinsic operations, we usually merge intervals with very sharp remainder bounds. For these cases the algorithm described here is perfectly sufficient to produce very good results.

3.5.4 Sign of a Floating Point Series

In Section 3.5.5, comparisons between floating point numbers and High Precision numbers will be reduced to the determination of the sign of a series of floating point numbers. Here we will start by presenting various methods to determine the sign of a finite series of floating point numbers.

In the following, let the series be given by n floating point numbers a_i. The goal then is to rigorously determine sign of the sum $\sum_{i=1}^{n} a_i$.

3.5.4.1 Double Precision Sign Determination

An easy and fast way to determine the sign of such a sum is to use rigorous double precision interval arithmetic. By summing up the a_i as described in Section 3.5.2, the resulting double precision interval can easily be tested for its sign.

This is particularly effective if the a_i are well conditioned in the sense of a High Precision number. In that case, performing the additions starting with the smallest number a_i and continuing with the next larger one yields a very sharp interval enclosure of the value which has a width of about 10^{-16} times the correct value of the sum. Thus, as long as the precise sum is non-zero (and not below the underflow threshold) this interval method will yield the correct sign.

This method is not suitable to test for exactly zero, since due to the unavoidable interval overestimation the resulting interval will always have a non-zero width and thus contain positive and negative values.

Similarly, this method fails if the a_i are not well conditioned. If there is cancellation in the double precision interval computation, the typical effects of interval arithmetic cause the resulting interval to be overly wide, possibly preventing the determination of the sign.

Consider for example the case when$a_1 = -1$, $a_2 = 1$, and $a_3 = 10^{-30}$. Clearly, then the result of the sum is 10^{-30} and hence the sign is positive. However, in double precision interval arithmetic $a_3 + a_2$ will yield an interval of width about 10^{-16} around 1 since a_3 is too below the floating point precision for the value 1. Consequently, after adding a_1

the resulting interval indeed is an interval around 0 containing both positive and negative numbers.

If, however, either a_1 or a_2 in the above series are dropped, the series is well conditioned and the double precision interval computed will easily and safely allow determination of the sign.

3.5.4.2 High Precision Sign Determination

To work around the problems of the double precision interval method of determining the sign, the following algorithm was developed. It is based on a simple bounding argument, and allows to determine the sign of a series of floating point numbers very precisely, only using floating point arithmetic. Without loss of generality, let $|a_1|$ and $|a_2|$ be the largest and second largest absolute values of the n numbers a_i.

Then we have

$$\sum_{i=1}^{n} a_i = a_1 + \sum_{i=2}^{n} a_i \leqslant a_1 + |\sum_{i=2}^{n} a_i| \leqslant a_1 + \sum_{i=2}^{n} |a_i| \leqslant a_1 + (n-1) \cdot |a_2|, \tag{3.31}$$

and

$$\sum_{i=1}^{n} a_i = a_1 + \sum_{i=2}^{n} a_i \geqslant a_1 - |\sum_{i=2}^{n} a_i| \geqslant a_1 - \sum_{i=2}^{n} |a_i| \geqslant a_1 - (n-1) \cdot |a_2|. \tag{3.32}$$

Let n^* be an integer power of two that is greater than or equal to $n - 1$. Then the inequalities obviously still hold, if $n - 1$ is replaced by n^*. Furthermore, all operations needed to calculate $n^* \cdot |a_2|$ are exact in floating point arithmetic. Combined with the fact that comparisons of floating point numbers are always exact, we can therefore conclude that the sign of the whole sum will be the same as the sign of a_1 if

$$|a_1| > n^* \cdot |a_2| \tag{3.33}$$

If, however, a_1 and a_2 are too close together, we can perform a Dekker addition of the two floating point numbers. That is guaranteed to result in two different floating point numbers, which rigorously replace a_1 and a_2 in the above sums without changing their value. Also they fulfill $|a_1| > 1/\varepsilon \cdot |a_2|$ and, if n^* in the above equation is smaller than $1/\varepsilon$, also $|a_1| > n^* \cdot |a_2|$. Repeating the process with this new series of numbers will provide different values for a_1 and a_2, and thus may allow to determine the sign.

Unfortunately, it cannot be guaranteed that this algorithm terminates. It is possible that for very specific input, the largest number either becomes smaller than the underflow threshold of the underlying floating point arithmetic, or that it becomes infinite in the floating point sense. In such cases, no definitive answer can be given, but in practice such cases do not appear and are thus largely irrelevant.

There are a few other special cases to be considered. In the trivial case of $n = 1$ then the sign of the sum is clear. If the largest value turns out to be $a_1 = 0$ then the sign is also clear, since then the whole sum has to be equal to 0.

As mentioned before, over- and underflow of the floating point numbers need to be considered separately. This can be done by e.g. testing the size of the number with the largest absolute value, and returning an undetermined result if this number is too large or too small to perform further operations. In those cases, no statement can be made about the sign of the sum.

3.5.5 Comparisons

Although high precision intervals are intervals, it is possible to define certain useful comparison operators to act on them. When comparing two real numbers, there are exactly three choices: one of them is either larger, smaller, or equal to the other. However, due to the interval nature of high precision intervals, these concepts have to be redefined.

Note that in the following definitions, one of the intervals can be assumed to have zero width, yielding the condition for comparisons to real numbers.

Definition 3.8. We say an interval A is larger than an interval B if the same is true for each pair of elements in the intervals:

$$A > B \Leftrightarrow x > y \mid \forall x \in A \ \forall y \in B$$

Definition 3.9. Similarly, we say an interval A is smaller than an interval B if the same is true for each pair of elements in the intervals:

$$A < B \Leftrightarrow x < y \mid \forall x \in A \ \forall y \in B$$

Equality, and subsequently inequality, is somewhat harder to define. Of course, one could use the mathematical definition of equality of two intervals

$$A = B \Leftrightarrow A \subseteq B \ \wedge \ B \subseteq A$$

However, for practical purposes this is not useful. After lengthy calculations, the chances that two intervals represented by a high precision intervals are exactly equal, are practically zero. This is particularly true if a program intends to check for a certain condition, such as $A = 0$. In practical situations, this is never be true since even if A had only one limb of value 0, the error bound very likely would not be zero.

Instead, we use the following definition for equality.

Definition 3.10 (Equality)**.** We say A is equal to B iff there is a non-zero intersection between the two intervals:

$$A = B \Leftrightarrow A \cap B \neq \emptyset$$

Inequality, on the other hand, is defined as the negation of equality.

Definition 3.11 (Inequality)**.** We say two high precision intervals A and B are not equal iff their intersection is the empty set:

$$A \neq B \Leftrightarrow A \cap B = \emptyset$$

This definition of equality has mainly practical value. In typical interval code, it is often necessary to test if two intervals overlap, or if a certain value is included in an interval. With the above definition of equality, these kinds of tests can be carried out easily and somewhat elegantly in a program.

Note that, while being useful, these definitions do not have all mathematical properties of the same relations on real numbers. Clearly, equality in the above sense is not transitive, as is illustrated by the intervals $[0, 2] = [1, 4] = [3, 5]$ but $[0, 2] \neq [3, 5]$. Furthermore, as discussed in the previous section, there is always a chance that a comparison cannot be decided. In that case, we a third value, “undecided”, is returned instead of “true” or “false”.

3.5.5.1 Comparing a High Precision Number to a Floating Point Number

To compare a High Precision number A to some floating point number b, we have to understand what this means in terms of the representation of A. Let $\lceil X \rceil = \sum_i x_i + x_{err}$ be the upper bound of the interval represented by High Precision number X, and $\lfloor X \rfloor = \sum_i x_i - x_{err}$ the lower bound. Then we can write the different conditions defined above as:

- $A > b$:

$$\lfloor A \rfloor > b \;\Rightarrow\; \sum_i a_i - a_{err} > b \;\Rightarrow\; \sum_i a_i - a_{err} - b > 0$$

- $A < b$:

$$\lceil A \rceil < b \;\Rightarrow\; \sum_i a_i + a_{err} < b \;\Rightarrow\; \sum_i a_i + a_{err} - b < 0$$

- $A = b$:

$$\begin{aligned} \lceil A \rceil \geqslant b \quad &\wedge \quad \lfloor A \rfloor \leqslant b \\ \Rightarrow \sum_i a_i + a_{err} \geqslant b \quad &\wedge \quad \sum_i a_i - a_{err} \leqslant b \\ \Rightarrow \sum_i a_i + a_{err} - b \geqslant 0 \quad &\wedge \quad \sum_i a_i - a_{err} - b \leqslant 0 \end{aligned}$$

There are two ways to implement these comparisons, as described in section 3.5.4. First, we will consider the double precision interval test. It is fast, and in many cases where the sign is rather clear it can provide a quick and definitive answer.

Unfortunately, evaluation of the sums in double precision intervals causes the tests to be inaccurate for close cases, however. The accuracy for distinguishing the sign of a series would only be about 10^{-16}, i.e. only if the result of the sum is larger in absolute value than 10^{-16} can a decision be made. This is due to the cancellation in the above series, which causes the interval bounding method to become inaccurate, as described above.

Consider for example the test of $A = 1 + 10^{20}$ versus the number 1. The sum to evaluate would end up being $1 + 10^{20} - 1$, causing the double precision intervals to include values both positive and negative. Note that it also does not help to compute a double precision enclosure of only A and then test its bounds against 1, since already here the result of the addition will most likely be an interval including values on both sides of 1.

If the simple double precision interval arithmetic test does not succeed, there are several options. One method is to perform a verified subtraction of the value b from A, and then comparing the result to 0. The High Precision subtraction operation ideally removes the cancellation happening in the above sums, yielding a well conditioned number that can easily be compared to 0. However, in doing so one has to be very careful in the interpretation of the result. Since the subtraction may result in overestimation, the inclusion of 0 in the difference does not guarantee the inclusion of b in A.

While in general quite good, this method has some drawbacks. It very slightly increases the error bound of the result, making it slightly more likely that a comparison fails. This effect is very small and in practice can be ignored. More importantly, the subtraction is not always guaranteed to return a well conditioned number, so that in some cases it may still be possible that the double precision interval bounding fails.

The high precision comparison described in Section 3.5.4.2 is essentially a boiled down version of the full addition. Unlike full addition, this algorithm stops as soon as a rigorous determination of the sign is possible. So in general, it is probably slightly faster and more accurate in determining the sign.

3.5.5.2 Comparing two High Precision Intervals

Comparing two high precision intervals A and B is a little bit more involved. Nonetheless, this case, too, can be reduced to the determination of the sign of a series of floating point numbers. Unfortunately, here one cannot simply take the difference $C = A - B$ and compare that to 0, because the subtraction will only yield an interval enclosure of the result. The result C, however, may be wider than it has to be due to overestimation. Thus it is possible that, if two disjoint intervals A and B are very close to each other, the subtraction yields in interval that includes 0. This leads to an incorrectly detected intersection of the two intervals.

In terms of the representation of A and B, and using the same notation as before, the comparison of the two high precision intervals can be written as:

- $A > B$:

$$\lfloor A \rfloor > \lceil B \rceil \;\Rightarrow\; \sum_i a_i - a_{err} > \sum_j b_j + b_{err} \;\Rightarrow\; \sum_i a_i - a_{err} - \sum_j b_j - b_{err} > 0$$

- $A < B$:

$$\lceil A \rceil < \lfloor B \rfloor \;\Rightarrow\; \sum_i a_i + a_{err} < \sum_j b_j - b_{err} \;\Rightarrow\; \sum_i a_i + a_{err} - \sum_j b_j + b_{err} < 0$$

- $A = B$:

$$
\begin{aligned}
&\lceil A \rceil \geqslant \lfloor B \rfloor \ \ \wedge \ \ \lfloor A \rfloor \leqslant \lceil B \rceil \\
\Rightarrow \ & \sum_i a_i + a_{err} \geqslant \sum_j b_j - b_{err} \ \ \wedge \ \ \sum_i a_i - a_{err} \leqslant \sum_j b_j + b_{err} \\
\Rightarrow \ & \sum_i a_i + a_{err} - \sum_j b_j + b_{err} \geqslant 0 \ \ \wedge \ \ \sum_i a_i - a_{err} - \sum_j b_j - b_{err} \leqslant 0
\end{aligned}
$$

As in the case for the comparison to a single double precision number, the bounding can be performed either in double precision or using the high precision bounding algorithm. However, in these cases the naive double precision interval bounding is usually quite useless for accurate comparisons of nearby intervals due to overestimation.

Instead, one has to carefully subtract the right expressions in verified high precision arithmetic, and bound the typically well conditioned result. Alternatively, the sign of the sum can be determined using the high precision sign determination algorithm.

3.6 Input / Output

Rigorously printing floating point numbers, and reading input from a string is essential to verified calculations. The functions provided by the underlying programing language are usually insufficient for this task. This is because the output will be converted to decimal fractions, which leads to truncation and thus incorrect results. The input functions provided are implemented to read input into double precision numbers. However, they are non-rigorous, and typically not even correct rounding is guaranteed.

Therefore, it is necessary to implement specific functions to perform these tasks. Although it might be possible to use some conversion functions provided by the system libraries, we will not rely on any of those. Instead, we implement all functions on our own. This not only guarantees portability, it also allows to ensure that every step of the conversion procedure is performed rigorously.

There are generally two types on input and output formats:

1. Human readable input/output. This format is used for human users to input values into code or a program and to read output generated by the program. As such it is usually in decimal notation. In most computer programming languages it is common to write decimal fractions in what is known as *scientific notation* (see below).

2. Machine readable input/output. This format is not intended to be read or manipulated by humans. It is stored in a format suitably close to the internal representation of numbers (usually binary), and if implemented properly this input can be read into program memory without loss of precision. Output in this format is typically generated by a computer program, and serves the purpose to restore certain data structures from persistent storage into the programs memory.

3.6.1 Rigorous Binary Output

When printing rigorous numbers, it is necessary to avoid conversion from the floating point number system used in the computer representation of a number. We print double precision numbers rigorously, by outputting their floating point representation exactly.

In order to extract the relevant parts of the bit pattern of a double precision number, we rely on IEEE 754[2] conform storage. This means that the number is represented internally as

$$s \cdot m \cdot 2^{e} \tag{3.34}$$

where the numbers are integers of different size: s is the sign (1 bit), m is the mantissa (52 bits[2]) and e is the exponent (11 bits). This results in a total of 64 bits.

To achieve rigorous output we extract all of these fields form the floating point number a, and write it as

$$\pm m_a b e_a$$

where m_a is the mantissa and e_a is the exponent. The character "b" as a separator is reminiscent of the "e" in scientific notation common in most programming languages. Thus the decimal number 4.25, for example, is output as $17b - 2$ meaning $17 \cdot 2^{-2}$.

It is obvious that this representation is not always unique. Powers of two can be shifted between the mantissa and the exponent somewhat freely. To make it both unique and easy to read, we attempt to keep the absolute value of the exponent as small as possible. This convention means that whenever possible, a number is output as an integer without the need for multiplications by powers of 2.

When outputting a High Precision number, each limb is printed once in this rigorous way, and once as a non-rigorous, decimal fraction approximation. That means the output is both human and machine readable at the same time.

3.6.1.1 Implementation

To extract mantissa, exponent and sign from a `DOUBLE PRECISION` variable in FORTRAN, we use the bit field functions introduced into FORTRAN by DoD military standard 1753[1] (also part of the Fortran 90 standard), and the concept of `EQUIVALENCE`, which allows different FORTRAN variables to share the same physical memory. We assume that the FOR-

[2] Actually it is 53 bits because the first bit is implicitly assumed to be one for non-denormalized numbers.

TRAN type `DOUBLE PRECISION` represents an IEEE 754 compliant double precision number in memory. This is not guaranteed by the FORTRAN standard, but seems to be universally true on all platforms and compilers we tested.

We declare a 64 bit integer variable using `INTEGER*8` as data type[3]. Then we declare this variable to be equivalent to a local `DOUBLE PRECISION` variable. Assigning the value in question to the local double precision variable, we can now operate on the bit pattern of the double precision number. Using bit field functions, we extract the relevant bits from the double precision number.

According to IEEE 754[2], the bit layout of a double precision number is as follows. Bits are numbered from 0 through 63, starting with the right most bit. This is in accordance with the numbering scheme used by the FORTRAN bit field functions.

The layout of the memory of a double precision number is given by the following table:

63	62	...	52	51	...	40	...	30	...	20	...	10	...	0
sign	exponent			mantissa										

To store the mantissa, a 64 bit integer is required, since there are more than 32 bits. The exponent and the sign are both stored in a regular FORTRAN integer (32 bits). To extract the bits, the FORTRAN bit field extensions provide the very useful function `IBITS(m, i, len)`. It extracts `len` bits from `m` starting at `i`, and returns a new integer that contains those bits flushed to the right, and the rest of it padded with zeros. This, of course, is exactly what we need to extract the parts:

```
EXPONENT
```

[3] This construct is not FORTRAN 77 standard, but it seems to be widely supported by all current compilers.

```
= IBITS( TEMP, 52, 11 ) MANTISSA = IBITS( TEMP, 0, 52 ) SIGN = IBITS(
TEMP, 63, 1 )
```

After extracting this information from the double precision number, we have another function interpret it and actually converting it into a character string. All strings returned are 27 characters long and right justified.

According to the IEEE754 standard, there are several cases:

MANTISSA = EXPONENT = 0 The number has the value 0.0. IEEE allows zeros to be signed, so the sign bit is still valid. The value of this number is ±0.0.

EXPONENT = 2047 signifies special cases, depending on the value of the mantissa:

- **MANTISSA** = 0 The number is infinity. The sign bit is valid. This number is converted into the string $Infinity$.

- **MANTISSA** $\neq 0$ The number is not valid, i.e. not a number. This is converted into the string NaN (“not a number”).

EXPONENT = 0, MANTISSA $\neq 0$ signifies a denormalized number, where the implicit bit of the mantissa is not set. Instead, m is directly used as extracted from the bit pattern. The exponent in this case is -1074, so that the value of this number is then $\pm m \cdot 2^{-1074}$.

Other values For all other combinations, the correct value must be calculated according to these rules: The exponent is biased with a biasing constant of 1075 which is subtracted from it, while in the mantissa the implicit leading bit, 52, is set to one. The value of this number is then $\pm m \cdot 2^{exponent}$.

The reason for the biasing constant being different from the one given in [2] is that we shift the decimal point of the mantissa to the end, so it becomes an integer instead of a binary fraction. That way, we have to subtract an additional 52 from the exponent.

3.6.2 Rigorous Decimal Output

While the storage format as an unevaluated series of double precision numbers is a good choice for computations and developing algorithms, it is not particularly intuitive for humans to read the resulting High Precision numbers. For easily human readable output, it is necessary to provide a mechanism to rigorously convert a series of floating point numbers into a decimal fraction.

The format in which we want to print a number is the same as the "scientific notation" for decimal fractions described in Section 3.6.3. In the case of rigorous output, this representation is followed by an additional rigorous error bound. For example:

$$1.28475635465859683635565786939273544363 8E-001$$

$$1.284756354658596E-001 \pm 1.21E-14,$$

where the notation in the second line is meant to be read as the interval $[1.284756354658596 \cdot 10^{-1} - 1.21 \cdot 10^{-14}, 1.284756354658596 \cdot 10^{-1} + 1.21 \cdot 10^{-14}]$.

To that end, we implemented an very simple decimal arithmetic based on character strings. While not efficient in the least, this allows for a relatively straightforward implementation of the needed operations based on "pencil-and-paper" type computations.

More precisely, any number in this arithmetic is represented by a mantissa M, which is represented as a decimal integer in a string of characters, along with a binary integer e representing the decimal exponent associated with this number and a sign s. The value of such a number is then given by $s \cdot M \cdot 10^{e}$. On these numbers, three operations will be implemented:

1. Addition

2. Multiplication by 2

3. Division by 2

These operations are sufficient to convert numbers from the binary fractions stored in a double precision floating point number into a decimal fraction. Furthermore, all of these operations result in finite decimal fractions, thus as long as the length of the mantissa character string is chosen large enough these operations are absolutely exact.

Then, in order to compute the correct decimal representation of a series of double precision numbers $S = \sum_{i=1 \ldots n} a_i$, the following steps are performed:

1. The sign s_i, mantissa m_i, and exponent e_i of each number a_i are extracted. The mantissa is stored as an integer in a 16 digit character string, while the exponent and sign are stored as binary integers. Then

$$S = \sum_{i=1 \ldots n} s_i \cdot m_i \cdot 2^{e_i}$$

2. The smallest exponent e_{min} is determined. Then

$$S = 2^{e_{min}} \cdot \sum_{i=1 \ldots n} s_i \cdot m_i \cdot 2^{e_i - e_{min}}$$

3. For each $i = 1 \ldots n$ the exact decimal expansion of $m_i \cdot 2^{e_i - e_{min}}$ is computed. This is done by repeated multiplication by 2 of the initial mantissa m_i. At this point no division by 2 is required yet, since by choice of e_{min} all exponents $e_i - e_{min}$ are positive. The resulting expansions are then summed into a final number M. Then

   $$S = 2^{e_{min}} \cdot M$$

   Note that all these operations so far are all purely integer operations, the decimal exponent of M can be chosen to be zero at this point.

4. Finally, S is computed by either multiplying or dividing M repeatedly by 2 for e_{min} times.

5. Depending on the desired rounding mode the result may need to be rounded up or down. This is performed by examining the first non-printed digit in the expansion and the sign of the result. If the result then needs to be rounded towards zero, it is just truncated. If rounding away from zero is required, the last printed digit is "bumped up" by adding a 1 in that place of the mantissa.

This results in an exact decimal representation of the value of S, with proper rounding if desired, which then can be formatted properly for output.

In the case of rigorous output, some additional steps are required. After computing S as described above, the following additional steps are performed to obtain an exact decimal representation of the truncation error and the error term *err* of the High Precision number being printed.

1. To compute the remainder error between the value for the center point that is printed

and the correct value, the actually printed digits after rounding are subtracted from the exact decimal expansion S. This yields a new number E which contains an exact decimal representation of the truncation error due to rounding and printing. The sign is modified to always be positive, effectively taking the absolute value of the truncation error.

2. As described for each of the a_i above, the double precision error term err of the incoming High Precision interval is converted to a decimal representation and then added to E. This now is an exact decimal expansion of the error term to be displayed.

3. Again as described above, the error term is rounded upward (towards positive infinity) at the last printed digit of the error term. This then results in a rigorously rounded representation of the error term that can be formatted and printed on the screen.

3.6.2.1 Implementation

First, a safe length for the mantissa character string has to be chosen. We suggest a maximum length of 1440 decimal digits. This is sufficient to add at least 100 completely arbitrary double precision numbers.

First, note that the maximum number of decimal digits that can occur in the integer resulting from the multiplication within the sum in step 2 is 632. This is because, after subtracting the smallest possible exponent -1074, the absolute value of the largest possible element under the summation sign is less than $2^{53} \cdot 2^{2045} = 2^{2098} < 4 \cdot 10^{632}$. Summing 100 of these values then certainly results in an integer less than $100 \cdot 4 \cdot 10^{632} < 4 \cdot 10^{634}$.

In the last step, this integer of at most 634 digits is then either divided or multiplied $|e_{min}| \leqslant 1074$ times by 2. If the number divided is odd, each division by two creates one more digit at the end of the decimal expansion. However, since $2^4 = 16 > 10$, for every 4 divisions, at least one leading decimal digit disappears. Thus division by 2 adds at most $1074 - \lfloor 1074/4 \rfloor = 806$ digits to the result. Multiplication by 2 adds at most one new digit at the beginning of the decimal expansion if the first digit is greater or equal to 5, but since $2^3 = 8 < 10$ does so no more often than in every third multiplication. Thus multiplication by 2 at most adds $\lceil 1074/3 \rceil = 359$ digits to the result.

Thus, the final result S will have at most $806 + 634 = 1440$ digits. This is only a rough upper bound, but it will provide sufficient digits for all of these operations to be carried out precisely.

In the following, a brief description of the implementation of each of the three required operations is given. All algorithms are essentially implementations of the digit-by-digit operations taught in middle school.

For ease of implementation, our implementation stores decimal expansions right-justified within their respective string, with the right-most digit being the least significant. The end of the decimal number is signified by a blank character to the left of the most significant digit.

For convenience, we also store the sign of the number in the left-most character of the string. This zeroth character does not count towards the character limit and is also not counted in the following descriptions. If the sign character is 'X', the corresponding number is defective due to some error during the computation and should not be trusted. Otherwise the sign is '+' or '-'. For printing purposes the sign can also be '~', resulting in a ± sign. The last sign is not valid for arithmetic operations and causes an error if used in that context.

All of the following operations carry out checks at runtime to insure that no attempts are made to write beyond the maximum length of a decimal expansion string. While the length of such strings is chosen large enough, as described above, this additional test ensures that in no case incorrect numbers are ever returned. Instead, an error is returned. These test will not be mentioned any more explicitly in the following descriptions, to keep them easily understandable.

3.6.2.1.1 Addition Addition surprisingly is the most complex of the operations. First, the decimal exponents e_1 and e_2 of the two input numbers are made equal. This is done by picking the smaller one, without loss of generality e_1, and adding $e_2 - e_1$ zeros on the right hand side of the decimal expansion corresponding to e_2. Now the addition has been reduced to an integer operation on the mantissae of the two numbers.

For two numbers with the same sign, addition is rather straightforward, and will be described first. The case of different signs is then discussed afterwards.

Addition of numbers with the same sign is performed digit by digit, starting with the right-most (least significant) digit. The corresponding digits of both expansions are added to form a number k, and the carry, if any is also added to k. If k is greater than or equal to 10, it is reduced by 10, and carry is set to 1. Otherwise, carry is set to zero. Lastly, the current digit in the result is set to the k. Then the process is repeated with the next digit to the left. If one of the two decimal expansions reaches the end, it is treated as if it were filled with 0 digits on the left.

Note that in this process the maximum value k can take is $9 + 9 + 1 = 19$. Thus, after the subtraction of 10, k is always a single digit.

If after reaching the last digit of both input numbers the carry is 1, it is added as an extra digit in the beginning of the expansion. The decimal exponent of the result is the same as e_1, the sign of the result is that of both input numbers.

In the case of differing signs of the two input numbers, the algorithm is modified as follows. Without loss of generality, let the first number be positive, and the second one negative.

Again, addition is performed digit by digit, starting with the right-most (least significant) digit. However, in this case the corresponding digit in the second number is subtracted from the digit in the first number to form k. Then the carry, if any, is added to k. If k is greater than or equal to 10, it is reduced by 10 and carry is set to 1. If k is less than 0, 10 is added to k and carry is set to -1. Otherwise, carry is set to zero. Lastly, the current digit in the result is set to the k. Then the process is repeated with the next digit to the left. If one of the two decimal expansions reaches the end, it is treated as if it were filled with 0 digits on the left.

Note that in this process the maximum value k can take is $9 - 0 + 1 = 10$, while the minimum is $0 - 9 - 1 = -10$. Thus, after the addition or subtraction of 10, k is always a single, positive digit.

If after reaching the last digit of both input numbers the carry is 1, it is added as an extra digit in the beginning of the expansion. The decimal exponent of the result is the same as e_1, the sign of the result is positive. The same is true if the carry is 0. If the carry in the end is -1, the 10-complement of the result has to be taken.

This is done by applying the following transformation to each digit in the result, starting with the rightmost digit. To the digit the carry, if any is added forming k. If k is greater than zero, it is subtracted from 10 and the carry is set to 1, otherwise the carry is set to 0. Then the current digit is replaced by the value in k. Since the maximum value for k is

$10 - 1 = 9$, it is always a single, positive digit. Then the process is repeated with the next digit to the left.

After this process has reached the left-most digit of the result, the sign of the result is then set to negative and its decimal exponent is e_1.

3.6.2.1.2 Multiplication by 2 Multiplication by 2 is the easiest of all operations. Starting from the end of the string, each digit is multiplied by 2 and the carry over from the previous digit, if any, is added. If the result is greater than or equal to 10, 10 is subtracted from the result and the carry is set to 1. Then the process is repeated with the next digit to the left.

Note that the maximum carry over to the next digit that can occur is 1. This is because the result of the multiplication of any digit by two is at most $2 \cdot 9 = 18$. Thus the first carry that can occur is at most 1, but with a carry of 1 the maximum value of the next digit after adding the carry is 19, which still causes a carry of only 1.

If after reaching the last digit of the input number the carry is 1, it is added as an extra digit in the beginning of the expansion.

Multiplication by 2 obviously does not affect the decimal exponent or the sign of the number.

3.6.2.1.3 Division by 2 Division is the only operation that "grows" the decimal expansion to the right instead of the left. Thus temporarily the decimal expansion is shifted so that it begins on the left of the character string with the most significant digit (i.e. it is left justified within the string). The end of the expansion is denoted by a blank character after the right-most (least significant) digit.

From there on, division follows a similar pattern as multiplication, except that operations start on the left. Each digit is divided by 2 and the carry is added to form k. The current digit is then replaced by the integer part of k. If k has a fractional part, the carry is set to 5, otherwise it is set to 0. Then the process is repeated with the next digit to the right.

Note that in each step k is always less than 10, since the carry is at most 5 and the result of the division is at most $9/2 = 4.5$.

If after reaching the last digit of the input number the carry is 5, it is added as an extra digit at the end of the expansion.

Finally the entire decimal expansion is shifted back to the end of the string (i.e. right justified within the string). If the left-most digit in the expansion is a 0, it is removed and the decimal exponent is reduced by 1. Division by 2 obviously does not affect the sign of the number.

Normalization Before printing a number, it should be normalized. This process removes excess zeros at the beginning and the end of the number, in the latter case adjusting the decimal exponent accordingly. Implementation of this routine is straightforward.

3.6.3 Rigorous Decimal and Binary Input

In this section we will present a string conversion algorithm that will read decimal fractions given in a special format called scientific notation, as well as the binary floating point notation introduced above. This conversion algorithm has to perform several tasks:

1. Parse the input string, and make sure it conforms to the specified format

2. Verify that the number is within the limitations of the appropriate type

3. Rigorously convert the number into the representation used internally by the computer system

First, it is necessary to define the input format. Our input routines accept a format that is based on the scientific notation. More specifically, numbers consist of several fields, some of which are optional. They are separated by specific character constants.

$$[+-]^{?}[0123456789]^{*}[.,]^{?}([0123456789]^{*})[eEhHbB]^{?}([+-]^{?}[0123456789]^{*}) \tag{3.35}$$

In this representation, which is loosely based on POSIX regular expressions, characters in square brackets represent the allowed character set for the respective field. Any character listed in the square brackets is an allowed character. The superscript behind the square brackets designates the number of characters from the preceding set that are allowed in the field. ? means exactly none or one character appear. $*$ means zero or more characters appear. Expressions in parentheses are only allowed to appear, if the field directly before is not empty.

Expression 3.35 dictates that each number has an optional sign of either $+$ or $-$. A missing sign is the same as a positive sign. This is then followed by zero or more digits representing the integer part of the number represented in base 10. This then is followed by an optional decimal point, which, to accommodate users from Europe as well as from the U.S., can be either a decimal dot "." or a comma ",". Only if this decimal separator is present, it is followed by an optional fractional part in base 10. The mantissa is then followed by an optional exponent separator which can be one of e, E, h or H. If this separator is present, it can be followed by another optional sign and a base 10 integer of zero or more digits.

The exponent for decimal expansions is limited in its range to lie within the interval $[-300, 300]$. If the exponent is outside of this range, an error is returned.

The exponent separator b or B is a special case, indicating a binary fraction in what we call the B-Format. If the input string is in this format, there are some additional restrictions on the input. B-Format numbers are not allowed to have any fractional mantissa part, i.e. the mantissa must be an integer. Furthermore, the value of the mantissa must lie in the interval $[-2^{53} + 1, 2^{53} - 1]$, i.e. the absolute value of the mantissa must be less than 2^{53}, and the exponent must be lie in the interval $[-1074, 971]$.

These values are derived directly from the definition of double precision floating point numbers in [2], chosen such that any number that passes these tests represents a valid double precision number of that precise value. The result may possibly be a denormalized underflow, but even in this case the number accurately represents the correct value. The smallest (denormalized) value representable in double precision is $2^{-1022-52} = 2^{-1074}$, while the largest number representable is $\left(2^{53} - 1\right) \cdot 2^{1023-52} = \left(2^{53} - 1\right) \cdot 2^{971}$.

It is easily verified that this convention guarantees that any number output in the format described in Section 3.6.1 is also a valid number according to these criteria and can be read back exactly.

3.6.3.1 Examples

To illustrate allowed formats, here are some examples of valid and invalid numbers:

- $0505, 1983E24$ is a valid number

- .666 is a valid number

- $1,0000000000000000000$ is a valid number

- 17 is a valid number

- $1b4$ is a valid number

- $-0.000911000E$ is a valid number (the missing exponent will be interpreted as 0)

- $-1E-99$ is a valid number

- $.E17$ is a valid number (it will be read as zero since the empty mantissa will be interpreted as 0.0)

- $-.$ is a valid number (it will again be read as zero)[4]

- $5;1983E17$ is an invalid number (; is not a decimal delimiter)

- $.\ \ 8888$ is an invalid number (illegal spaces after the decimal delimiter)

- $1,000.0$ is an invalid number (two decimal delimiters)

- $\ \ 17$ is an invalid number (illegal spaces in front)

- $1.5b-3$ is an invalid number (no fractional part allowed for B-Format)

- $12345678901234567890b1$ is an invalid number (mantissa too large for B-Format)

- $1*10**22$ is an invalid number (illegal character $*$)

- $1A7EFF$ is an invalid number (illegal characters A and F, all input has to be base 10)

[4] Although this and the preceding examples are valid, it is not recommended to use this representation since it is confusing. Use 0 instead.

In addition to this format, the conversion functions also accept certain special strings, which are handled as completely separate cases in the beginning of the conversion process. Currently, those character constants are (case-insensitive):

1. “INFINITY”, “INF”, “INFTY” - Positive infinity double precision constant,
2. “-INFINITY”, “-INF”, “-INFTY” - Negative infinity double precision constant,
3. “NAN”, “–NAN–” - Not a number double precision constant,
4. ”PI” - The value of the pre-calculated High Precision number representing π in the current precision,
5. ”2PI”, ”2*PI” - The value of the pre-calculated High Precision number representing 2π in the current precision,
6. ”PI/2” - The value of the pre-calculated High Precision number representing $\pi/2$ in the current precision,
7. “LN2” - The value of the pre-computed High Precision number representing $\log(2)$ in the current precision.

This allows for certain constants to be obtained easily, without requiring the user to calculate them in their code.

3.6.3.2 Implementation

Another important issue is the accuracy of the resulting numbers in the case of decimal fractions. Some numbers that have a finite fractional representation in decimal notation, do not have such a representation in binary. Numbers as simple as 0.1 cannot be represented by a finite binary fraction. By design this is not an issue for binary fractions in the B-Format.

The following conversion algorithm attempts to encode at least the current number of digits exactly, possibly more. In any case, it always attaches an error bound to result, so that the number represented by the string is guaranteed to be contained in the resulting high precision interval.

Parsing of the string is done by stepping through the input character by character, and checking if it matches the allowed character set of the current field. If it does, as many following characters are consumed as possible until a non-matching character is hit. At that point, the current field must have ended, and the parsing continues with the next field description. Whenever a new field is entered, the beginning index is stored for later use. Similarly, the ending index is stored whenever a field has ended. After completing all fields, we thus have beginning and end indices for all fields. If there are characters left at the end of the string, this means the input did not match our format. In that case an error is returned.

In the following, we will make use of the fact that certain integer operations, if performed in double precision, are exact. The mantissa of a regular double precision number has 53 bits. Thus, it can store numbers between $-2^{53} \approx -9 \cdot 10^{15}$ and $2^{53} \approx 9 \cdot 10^{15}$ exactly. Since our floating point model is round to nearest, if the result of an operation is representable as a floating point number, the operation is exact. Therefore, integer addition and multiplication performed on double precision numbers are exact, as long as the integers lie between $-9 \cdot 10^{15}$ and $9 \cdot 10^{15}$. Consequently, any 15 digit integer in decimal representation can be read exactly into a floating point number.

3.6.3.2.1 B-Format For the B-Format, the actual conversion is rather simple. Due to the bound checks performed previously during parsing, the value of the mantissa is guaranteed to fit into a double precision number precisely. Thus it can just be assembled digit by digit in floating point arithmetic without any roundoff errors, since all integers less than 2^{53} can be represented precisely in double precision arithmetic.

The last remaining step is to multiply the result by $2^{|exp|}$ or $\frac{1}{2}^{|exp|}$ respectively. Here it is important to note that one cannot just compute $2^{|exp|}$ and multiply or divide by this value. This is because for the minimum allowed exponent, -1074, the value 2^{1074} is floating point infinity, while $\frac{1}{2}^{1074}$ is still a valid (denormalized) value. Apart from this caveat, the computation of $2^{|exp|}$, as well as the subsequent multiplication of the mantissa, can be carried out exactly in double precision floating point arithmetic.

In general, to evaluate $B^{|exp|}$ efficiently, we have devised the following computation scheme. It requires very few operations, and thus is rather fast. While not relevant in the case of $B = 2$, for other values of B represented as High Precision numbers this also means less overestimation of the result.

Expressing the exponent in binary form, $exp = b_0 b_1 b_2 b_3 \ldots$ with $b_i \in \{0, 1\}$, we obtain:

$$B^{exp} = B^{(b_0 \cdot 2^0 + b_1 \cdot 2^1 + b_2 \cdot 2^2 + b_3 \cdot 2^3 + \ldots)} = \prod_i B^{b_i \cdot 2^i}$$

In the product, only the i with $b_i = 1$ contribute to the result since for $b_i = 0$ we have $B^{b_i \cdot 2^i} = B^0 = 1$. Calculating those remaining factors is easy, considering that:

$$B^{\left(2^i\right)} = \left(B^{\left(2^{i-1}\right)}\right)^2 = \left(\left(B^{\left(2^{i-2}\right)}\right)^2\right)^2 = \cdots = \left(\left(\ldots\left(B^2\right)^2\ldots\right)^2\right)^2.$$

Let E be a High Precision number or double precision number containing 1 initially. Starting with a High Precision number or double precision number A containing $B^{\left(2^0\right)} = B^1 = B$, A is squared successively. If the corresponding bit is set in the exponent, then E is multiplied by A. Since the exponent is limited to be smaller than 512 (2048 in the B-Format), there are at most 9 (11) bits to check.

Rigorous Decimal Format To convert a string in decimal format into a verified interval enclosure N, first the value of the exponent *exp* including its sign is extracted from the string and converted into an integer in a straightforward integer conversion.

Next, this exponent is adjusted for use in the following steps. Let d be the number of digits before the decimal point in the input string, or 0 if there are none. Then we set

$$e = exp + d - 1 - 15$$

and set E to be a rigorous High Precision number enclosure of 10^e. For an efficient way to compute $10^{|e|}$, see the previous paragraph. In this case, it is sufficient to compute $10^{|e|}$, and then compute its multiplicative inverse afterwards if the sign of e is negative.

This adjustment ensures that each digit in the mantissa is assigned its correct value. The first digit in the mantissa represents a decimal power given by its position in the mantissa string, d, and the decimal exponent *exp* as $10^{exp+d-1}$. The additional correction to the exponent of subtracting 15 stems from reading 15 digits of the mantissa at once.

The last step is actually reading the mantissa from the string. To convert the mantissa digits, the mantissa is stepped through from left to right, and the digits are assembled into a `DOUBLE PRECISION` number m. The decimal point, if present, is simply skipped over completely. Every 15 digits, m is multiplied by the High Precision number E and the result is added to the final number N. Both of these operations are of course performed in rigorous High Precision arithmetic. Afterwards, m is set to zero and the exponent E is multiplied by a precomputed, rigorous High Precision enclosure of 10^{-15} in preparation for the next round. Reading now continues with the next 15 digits, until there are no more digits to be read.

If the number of digits in the mantissa is not a multiple of 15, the last read value of m is padded with zeros at the end by multiplying it by the appropriate power of 10 before multiplying it by E and adding to the final result N, to make up for the "missing" digits.

After this process, the sign of the variable N is adjusted according to the input string to complete the conversion into a rigorous enclosure of the decimal expansion given by the input string.

The method described here works for all decimal expansions representing valid numbers, regardless of their length. Digits of the mantissa which fall below the current precision are read correctly and are added to the error bound of the result.

3.7 Future Work

The algorithms presented in this chapter, combined with the implementation of addition and multiplication of High Precision Taylor Models, form the basis of a new, unified datatype in the COSY INFINITY programming environment. The work of implementing all required helper functions and related operations for this datatype is still ongoing. Once finished, this will result in a new major version of COSY INFINITY version 10.

3.7.1 Automated Plausibility Testing

To automatically check that the results of these algorithms are indeed plausible (a test for correctness is unfortunately not possible), Ravi Jagasia and I are working on a special version of COSY INFINITY that after each intrinsic operation will call the external high precision interval library MPFI[40]. The argument to the intrinsic will be converted to an MPFI number, evaluated using the MPFI provided intrinsic function, and then compared to the result computed by the algorithms as described above. If the resulting intervals do not overlap, this is an indicator that clearly one of them is wrong. If this happens an error and debugging information is printed and the program is aborted.

This method is very slow and inefficient. It is not intended for everyday use, but as a development tool to test the plausibility of the results of our intrinsic operations while performing real world computations.

Chapter 4

Fixed Points

In this chapter, various algorithms for finding and verifying the existence of fixed and periodic points in discrete maps are described. In the rest of this chapter, we will use the terms periodic point and fixed point somewhat interchangeably, considering how a periodic point of period m of a given map M is a fixed point of the map M^m.

More precisely, given a continuously differentiable map

$$M : \mathbb{R}^n \mapsto \mathbb{R}^n,$$

we want to identify verified enclosures of periodic points $\vec{x}$ of period m of that map, i.e. a point $\vec{x} \in \mathbb{R}^n$ such that

$$M^m(\vec{x}) = \vec{x}.$$

The direct analytical proof, of course, would be to compute the exact periodic point $\vec{x}$. In exact arithmetic, applying the map M m times then yields the initial argument $\vec{x}$. Unfortunately, this direct method is only possible for simple maps where the periodic point can be easily calculated in closed form. As soon as this becomes impossible, that way of proving the existence of a periodic point cannot be employed anymore. Furthermore, if performed on a computer, round-off errors in floating point numbers, as well as overestimation in any verified numerical method will cause the result $M(\vec{x})$ to always differ from the initial argument $\vec{x}$.

Non-verified floating point arithmetic in particular is subject to misleading results. Consider, for example, the following map

$$M(x) = x + 1 + 10^{-20} - 1.$$

This map clearly simplifies to the expression $M(x) = x + 10^{-20}$, which does not have any fixed point. However, when evaluated in floating point arithmetic, every x will appear to be a fixed point. This is because in floating point arithmetic either already the addition of 10^{-20} is below the round-off threshold for $x + 1$, or if it is not then the subsequent subtraction of 1 will put it there.

In the following, several methods will be introduced to rigorously prove the existence of a periodic point of a given map M using rigorous numerical methods. In particular, we are interested in methods that do not require exact knowledge of the periodic point, but rather allow us to prove that a certain subset $K \in \mathbb{R}^n$ contains a periodic point. That way, if K is small enough, one can obtain very good approximations to the position of a periodic point.

Furthermore, the uniqueness of the periodic point within K is subject of our study.

4.1 General Concepts

We begin by introducing some general mathematical theorems that prove the existence of fixed points under various conditions.

4.1.1 Brouwer's Fixed Point Theorem

To only prove the existence of one or more fixed points in certain sets K, one can make use of the following version of the well known Brouwer fixed point theorem[13] (sometimes also called the Schauder fixed point theorem):

Theorem 4.1 (Brouwer Fixed Point Theorem)**.** *Let a continuous map $M : \mathbb{R}^n \mapsto \mathbb{R}^n$ and a convex compact set $K \subset \mathbb{R}^n$ be given. If $M(K) \subset K$ then K contains at least one fixed point $\vec{x}$ of M.*

Note that this statement of the theorem uses the well known fact that any convex compact set in $\mathbb{R}^n$ is homeomorphic to a closed unit ball of some dimension $k \leqslant n$[20].

The conditions of Brouwer's theorem are easily verified by rigorous computations, which makes this theorem very useful for our purposes. All that is required are certain bounding routines, which are readily available in rigorous computations.

4.1.2 Banach's Fixed Point Theorem

Probably the best known fixed point theorem in mathematics is the Banach fixed point theorem[3].

Theorem 4.2 (Banach Fixed Point Theorem)**.** *Let a K be a compact metric space and $M : K \mapsto K$ a continuous map of the space into itself. If M is a* contraction*, then K*

contains exactly one unique, attractive fixed point $\vec{x}$ of M.

Unlike the Brouwer fixed point theorem, this theorem can be applied to prove the existence and uniqueness of a fixed point. Note, however, that the Banach Fixed Point Theorem assumes a map of some compact metric space into itself. Taking this space to be $K \subset \mathbb{R}^n$ used in Brouwer's Fixed Point Theorem, the Banach Fixed Point Theorem can be viewed as an extension of Brouwer's Fixed Point Theorem.

For a mere proof of existence, it is numerically more favorable to use Brouwer's theorem for several reasons. The Banach fixed point theorem requires knowledge of a Lipschitz constant k for the map. Often this is obtained through the derivatives of the map. Unfortunately, it is not always convenient to obtain an analytic expression of the derivative of a map. But even if the derivative is known, evaluating the derivative of a high period iteration of a map by the chain rule is computationally expensive.

4.1.3 Contraction Maps

Definition 4.3 (Contraction Map)**.** Recall that a map M is said to be a contraction on K if $\exists k, 0 \leqslant k < 1$ such that

$$|M(\vec{x}) - M(\vec{y})| < k\,|\vec{x} - \vec{y}| \;\; \forall \vec{x}, \vec{y} \in K. \tag{4.1}$$

The numerical verification of the contraction property 4.1, unfortunately, is not quite as straight forward as the verification of the inclusion mapping property in Theorem 4.1. The following provides an overview of various ways to assert that a map M satisfies equation 4.1, and discuss their fitness for use in verified numerical computations.

4.1.3.1 Operator Norm

We begin with the following definition of the operator norm of a matrix:

Definition 4.4 (Matrix Operator Norm)**.** Let A be a real valued $n \times n$ matrix. The matrix operator norm of A is given by

$$||A|| = \sup_{|\vec{x}|=1} |A \cdot \vec{x}| = \sup_{\vec{x} \neq 0} \frac{|A \cdot \vec{x}|}{|\vec{x}|}$$

where the vector norm $|\cdot|$ is the Euclidean norm on $\mathbb{R}^n$ (or $\mathbb{C}^n$). It is easy to see that due to the linearity of A both expressions are equivalent.

Lemma 4.5. *Let $C \subset \mathbb{R}^n$ be a non-empty, convex, and compact set and f by a continuously differentiable function on C such that $||Df(\vec{x})|| < k < 1 \ \forall \vec{x} \in C$. Then f is a contraction on C.*

Proof. Given any two points $\vec{x}, \vec{y} \in C$, consider the function $G : [0,1] \to \mathbb{R}$ given by the dot product

$$G(t) = (f(\vec{y}) - f(\vec{x})) \cdot f\left((1-t)\vec{x} + t\vec{y}\right)$$

Applying the Mean Value Theorem to this function, one finds that for some $\xi \in (0,1)$

$$\begin{aligned} (f(\vec{y}) - f(\vec{x})) \cdot (f(\vec{y}) - f(\vec{x})) &= G(1) - G(0) \\ = \quad G'(\xi) &= (f\left(\vec{y}\right) - f\left(\vec{x}\right)) \cdot \left(Df\left((1-\xi)\vec{x} + \xi\vec{y}\right) \cdot (\vec{y} - \vec{x})\right) \end{aligned}$$

Applying the Cauchy-Schwarz inequality to the expression on the right hand side one thus obtains

$$|f(\vec{y}) - f(\vec{x})|^2 \leqslant |f\left(\vec{y}\right) - f\left(\vec{x}\right)| \, ||Df\left((1-\xi)\vec{x} + \xi\vec{y}\right)|| \, |\vec{y} - \vec{x}|$$

and hence

$$|f(\vec{y}) - f(\vec{x})| < k\,|\vec{y} - \vec{x}|$$

□

Corollary 4.6. *Let $K \subset \mathbb{R}^n$ be non-empty, compact, convex and $M : K \to K$ continuously differentiable with $||DM(\vec{x})|| < 1 \; \forall \vec{x} \in K$. Then*

$$\exists! \vec{x} \in K \text{ such that } M(\vec{x}) = \vec{x}.$$

Proof. Apply Lemma 4.5 and the Banach fixed point theorem. □

The problem with Corollary 4.6 is to determine a rigorous upper bound for the operator norm of the derivative. Definition 4.4 defines what the value is, but provides no direct way of computing it for a given matrix. To that end, recall the following standard result of operator theory[15]:

$$||A|| = \sqrt{\lambda_{\max}\,(A^* A)} \tag{4.2}$$

where $A \in \mathbb{C}^{n \times n}$, and A^* denotes the adjoint operator, i.e. the transpose conjugate of A, and $\lambda_{\max}$ denotes the largest eigenvalue of a matrix.

Unfortunately, while $A^* A$ can be computed easily, determining the largest eigenvalue of a matrix rigorously is non-trivial in general. This is particularly true in higher dimensions. For the special case of dimension 2, however, the eigenvalues can be computed by an analytic expression. So in this special case this approach is rather straight forward and provides an analytic way to compute the matrix norm and verify the contraction property.

4.1.3.2 Infinity Norm

Since direct calculation of of the operator norm of a matrix is not straight forward, another option is to use a more readily computed quantity to provide an upper bound of $||A||$ for $A \in \mathbb{C}^{n\times n}$. Consider, for example, the following upper bound:

$$||A|| \leqslant \sqrt{n}||A||_\infty \tag{4.3}$$

where $||A||_\infty$ indicates the maximum absolute sum of the rows of A, i.e.

$$||A||_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |a_{i,j}| \tag{4.4}$$

Proof. For any $\vec{x} \in \mathbb{R}^n$ we have

$$\begin{aligned} |A\vec{x}|^2 &= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} a_{ij} x_j \right)^2 \\ &\leqslant n \max_{i=1,\ldots,n} \left(\sum_{j=1}^{n} |a_{ij}||\vec{x}| \right)^2 \\ &= n|\vec{x}|^2 \left(\max_{i=1,\ldots,n} \sum_{j=1}^{n} |a_{ij}| \right)^2 \end{aligned}$$

□

Clearly, $||A||_\infty$ is easily computed in a verified setting for any matrix A. Unfortunately, in general this inequality is a rather crude upper bound of the operator norm $||A||$.

4.1.3.3 Coordinate Transformation

Using inequality 4.3, it is possible to obtain a much more accurate condition to test for the contraction property of M under certain circumstances. Since we are only interested in showing the uniqueness of a fixed point, it is irrelevant in what coordinate system the map is a contraction. The following approach to compute the operator norm $||A||$ does so by changing into a more suitable coordinate system.

Let $T : C \to \mathbb{R}^n$ be an invertible, smooth coordinate transformation. Then the following are equivalent:

- f has a unique fixed point in C
- $F = T \circ M \circ T^{-1}$ has a unique fixed point in $T(C)$.

This is clear since for $x \in C$ such that $M(x) = x$ one finds that $T(x)$ satisfies $F \circ T(x) = T(x)$ and vice versa for $y \in T(C)$ such that $F(y) = y$, $T^{-1}(y)$ satisfies $M \circ T^{-1}(y) = T^{-1}(y)$.

Let the derivative of F be decomposed into a purely diagonal part L and a purely off-diagonal part N such that

$$DF = L + N \tag{4.5}$$

Then applying the triangle inequality yields

$$||DF|| \leqslant ||L|| + ||N||.$$

The operator norm $||L||$ is simply the largest absolute value of the diagonal entries of L, while $||N||$ can be estimated by inequality 4.3.

Let H denote the convex hull of $T(C)$. By the assumptions of Corollary 4.6, Theorem

4.1 applies and yields at least one fixed point of M in C and hence a fixed point of F in $T(C) \subset H$. If $||DF|| < 1$ on H, Lemma 4.5 applied to the function F on H establishes the uniqueness of that fixed point and thus the uniqueness of the corresponding fixed point of f in $C \subset T^{-1}(H)$.

Note that if T can be chosen such that the derivative of F on $T(C)$ is approximately diagonalized, the entries in N are small compared to those in L, and the overestimation introduced by the estimation of $||N||$ is negligible compared to $||L||$ and the bound in $||DF||$ will be a rather tight one. Furthermore, if $T(C)$ itself is already convex, there is no additional overestimation introduced by evaluating the derivative over the convex hull of $T(C)$.

Such a coordinate transformation T can be found relatively easily if $DM(x)$ is diagonalizable at some point $x \in C$. Then T can be chosen to be an approximation of the matrix of eigenvectors. In that case, also $T(C)$ will be convex if C was convex, as T is a linear transformation.

To obtain a sharp bound, the derivative is only required to be approximately diagonal. It is thus not necessary to compute eigenvalues or eigenvectors rigorously. Only the matrix inversion T^{-1} has to be performed rigorously.

4.2 Periodic Point in Near Standard Hénon Map

We now apply the mathematical tools presented above to an example. In the following, we consider one of the simplest maps exposing chaotic behavior, the Hénon map[21], given by

$$H : \begin{cases} \mathbb{R}^2 & \to \mathbb{R}^2 \\ (x, y) & \mapsto (1 + y - Ax^2, Bx). \end{cases}$$

In the original map studied by Hénon himself, and many others after him, the parameters were chosen to be $A = 1.4$ and $B = 0.3$. We consider a slightly different value $A = 1.422$ while keeping $B = 0.3$.

In this map, the existence of an attracting period 15 point was suggested by non-verified numerical experiments by S. Newhouse[39]. The coordinates of this suggested periodic point are approximately

$$\begin{aligned} x &\approx -0.0869282203452939, \\ y &\approx 0.2391536750716747. \end{aligned}$$

As this point seems to be attractive, simple iteration of the Hénon map, starting with those coordinates, or even just the origin, improves the approximate coordinates until the change between iterations is of the size of floating point errors. In standard double precision calculations this will yield about 16 valid decimal digits.

Non-verified floating point calculations help gaining some insight into the general behavior of this periodic point. Simply iterating the coordinates given above by the map H^{15} a few times in double precision yields the following interesting results.

$$
\begin{array}{lll}
H^{15}: & x = -0.086928220345\underline{4442} & y = 0.2391536750716\underline{964} \\
H^{30}: & x = -0.086928220345\underline{2939} & y = 0.2391536750716\underline{747} \\
H^{45}: & x = -0.086928220345\underline{4442} & y = 0.2391536750716\underline{964} \\
H^{60}: & x = -0.086928220345\underline{2939} & y = 0.2391536750716\underline{747} \\
 & \vdots & \vdots
\end{array}
$$

Note how the last few underlined digits alternate between two different points. Looking at this result naively, one may suspect that the periodic point is actually of period 30, instead of period 15. It will turn out, however, that these oscillations are caused by floating point round-off errors, and have no mathematical significance. This just happens to be an orbit for which this particular effect appears.

The location of this orbit within the attractor of this Hénon map is shown in figure 4.1.

4.2.1 Existence of a Periodic Point

Given an non-verified approximation of a suspected attractive fixed point, the following method provides a verified proof thereof[48].

Let M be a continuously differentiable map. Without loss of generality, we assume the presumed periodic point of order m is near the origin. If that was not the case, we simply consider the new map

$$M_{new}(\vec{x}) = M(\vec{x} + \vec{z}) - \vec{z}, \tag{4.6}$$

where $\vec{z}$ is an approximation of the presumed periodic point. This new map is still continu-

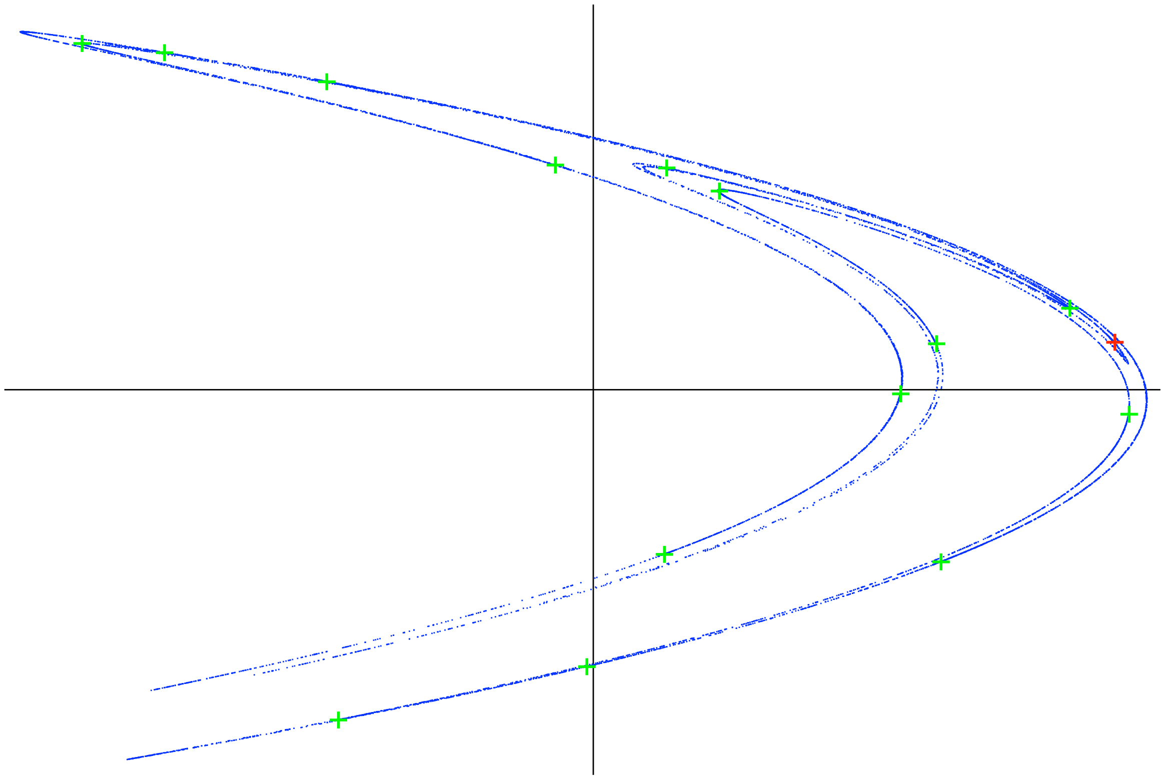


Figure 4.1: The attractor of the Hénon map with $A = 1.422$ and $B = 0.3$ (blue) with the orbit of the suspected period 15 periodic point (green). The point used for verification in Section 4.2.2 is highlighted in red.

ously differentiable, and will have a presumed periodic point of the same order m near the origin.

Furthermore, if Q is a regular $n \times n$ matrix, then instead of M^m, we can consider the conjugated map

$$\widetilde{M}(\vec{x}) = Q^{-1} \circ M^m \circ Q(\vec{x}). \tag{4.7}$$

By continuity, if $\widetilde{M}$ has a fixed point x_0 near the origin, then M has the periodic point $Q(x_0)$ of order m which is also near the origin.

Let K be a n-dimensional interval vector $K = ([-\epsilon, \epsilon], \ldots, [-\epsilon, \epsilon])$, where $\epsilon > 0$ is small. Being a Cartesian product of closed intervals, K is certainly compact and convex for any choice of ϵ.

We now want to show that $\widetilde{M}(K) \subset K$. Once this has been proven, applying Brouwer's theorem 4.1 leads to the conclusion that there is a fixed point of $\widetilde{M}$ in K and hence a periodic point of period m of M in $Q(K)$. □

Note that, depending on the map M and the choice of ϵ, the proof may not yield a conclusive result. If $\widetilde{M}(K) \bigcap K \neq \emptyset$ but $\widetilde{M}(K) \not\subset K$, then no statement can be made about the existence or non-existence of a periodic point.

4.2.1.1 Preconditioning

Assume now the Jacobian J of M^m at the origin is known approximately through some non-verified, numerical computation, e.g. using non-verified differential algebra techniques. Assume furthermore all eigenvalues of J have norm less than 1. There then is a linear transformation Q, such that $Q^{-1}JQ$ contracts each face of the box K towards the origin for sufficiently small ϵ.

This preconditioning by Q transforms, to first order, the attractive region around the origin to a rectangle, instead of a more general parallelogram. With this preconditioning, the proof will succeed in more cases that would otherwise not yield a conclusive result. Consider, for example, in dimension 2 the linear map Φ_α given by

$$\Phi_\alpha(\vec{x}) = \alpha \cdot \begin{pmatrix} -1 & 2 \\ 0 & 1 \end{pmatrix} \cdot \vec{x}$$

with $0 < \alpha < 1$. The eigenvalues are obviously $\pm\alpha$, and thus less than one in absolute value, so the origin is an attractive fixed point. Yet the point $(-1, 1)$ is mapped to $\alpha \cdot (3, 1)$, which depending on the value of α can even grows in norm, and thus for such values of α Φ_α is not a contraction. After a change of coordinates such that the coordinate axes become eigenvectors, however, the map becomes a contraction.

For attractive fixed points, the change into eigen-coordinates is sufficient to ensure contraction. For hyperbolic or repelling fixed points, a more general preconditioning method is described in Section 4.3.1.

4.2.1.2 Rigorous Coordinate Transformation

When performing the preconditioning described above, it is necessary to perform a translation and an affine transformation. While this is easy to do mathematically, some care has to be taken when this is done in rigorous computer arithmetic. The translation is simple, as the calculation of the inverse is just the negation of the periodic point coordinates, which is exact in floating point arithmetic.

The more complicated part is the linear transformation into eigen-coordinates. To obtain this transformation, the approximate eigenvectors of the map at the periodic point have to be determined first. Here it is not necessary to be rigorous, as the proof above clearly is correct with any regular linear transformation.

To obtain the expansion of the map to first order, we use non-verified, automatic differentiation through the DA data type in COSY INFINITY, but any other numerical or analytical method can be used. Once the matrix representation of the linear part of the map is obtained, any of the well known numerical methods to calculate eigenvalues and eigenvectors can be applied. This allows the construction of the matrix transforming eigen-coordinates into Cartesian coordinates.

To complete the coordinate transformation, this matrix has to be inverted. This inversion has to be performed rigorously to ensure the resulting matrix really represents a rigorous enclosure of the inverse. In this example, the Hénon map is only two dimensional and thus the inversion of the matrix is easily performed analytically. Instead of performing the calculations in floating point arithmetic, either interval or Taylor Model arithmetic is used. That way, one obtains a matrix of intervals or Taylor Models, which can be used to rigorously transform from Cartesian coordinates into eigen-coordinates.

Once this is done, a rigorous representation of $\widetilde{M}$ can be constructed.

4.2.1.3 Proof using Intervals

Following the discussion above, using intervals in principle is a straightforward procedure. First, a small interval box K is chosen. Then K is mapped through $\widetilde{M}$ and the result $\widetilde{M}(K)$ is verified to lie within K. With intervals, this test can easily be performed. All that has to be done is to compare the boundaries of the resulting intervals for each coordinate with the corresponding coordinates boundaries of the initial interval box K.

However, due to the overestimation in interval arithmetic, the initial box as a whole will not be mapped into itself right away. In particular, if the map $\widetilde{M}$ is of high order, there will be significant dependency introduced in the interval arithmetic, and the result of $\widetilde{M}(K)$ will be several orders of magnitude larger than the smallest enclosure of the mathematically correct result. Nonetheless, as long as there is some overlap between $\widetilde{M}(K)$ and K, there is a chance that K could actually be mapped into itself.

To overcome this problem, it is necessary to split the starting box K into smaller boxes and map each of these boxes separately. As the error in interval arithmetic decreases linearly with the box size, we can expect that, after a sufficient number of splits, the small boxes will eventually map into the original box K. Clearly, all of K is mapped into itself if and only if every small sub-box is mapped entirely into K.

For the actual implementation, certainly one does not want to split the box into small pieces of a fixed, predefined size. The best box size is a priori unknown, and, in fact, the optimal size of the split boxes varies depending on the position within the starting box K. Pieces close to the center can be larger than pieces along the boundary of K.

Instead of pre-splitting, an algorithm that automatically determines the optimal box size for the region under consideration is preferable. To implement this efficiently, it is best to use a stack based approach. Whenever a given box fails to map fully into K, it is split and the process is continued with the newly created boxes until all boxes were successfully mapped into K.

The complete algorithm for the verification procedure with automatic box size control is as follows:

1. Start with initial box K on the stack

2. Take the top box S off of the stack and evaluate $\widetilde{M}(S)$

3. Compare $\widetilde{M}(S)$ and K

    - If $\widetilde{M}(S) \subset K$, discard S as it has been successfully mapped into K.
    - Otherwise, if $\widetilde{M}(S) \bigcap K \neq \emptyset$, split S along each coordinate axis once, and push the resulting four pieces onto the stack.
    - If neither of the above is true, we have $\widetilde{M}(S) \bigcap K = \emptyset$. In this case, declare failure and stop, as we just mapped a piece of K completely outside of K.

4. Continue with 2 until there are no more boxes on the stack.

While this looks trivial, there is a pitfall hidden in the implementation of this algorithm. For the interval based algorithm to work, we require relatively small boxes. In all but the most simple cases, the split boxes have to be so small, that double precision intervals are not precise enough to carry out the operations anymore.

In 4.6 $\vec{x} + \vec{z}$ is calculated, where $\vec{z}$ is the presumptive periodic point, and $\vec{x}$ is one of the small boxes. In typical maps such as the Hénon map, the coordinates of the periodic point $\vec{z}$ will be of order one, whereas the box size can easily be of order 10^{-15}, depending on the required accuracy. It is evident, that this addition cannot be carried out precisely in double precision.

As a result, as the search space is split into "smaller" pieces, those pieces are then effectively lost due to the limited precision in the coordinate transformation. That is the reason why an interval package with high precision (e.g. the High Precision intervals introduced in Chapter 3) is required to carry out these calculations.

4.2.1.4 Proof using Taylor Models

The proof using Taylor Models differs significantly from the interval version of the proof. Taylor Models significantly reduce the overestimation that caused the intervals to grow rapidly. This is due to the drastic reduction of the dependency problem intrinsic to Taylor Model arithmetic. Thus, instead of mapping a very small box by successively splitting it, just one single box will be mapped. This is already sufficient to map the box into itself and thus to conclude the proof.

To set up the calculation, first the map $\widetilde{M}$ is generated with an inverse Taylor Model matrix as described in Section 4.2.1.2. Next, two Taylor Models x_i and y_i with only a linear coefficient in the first and second independent variable, respectively, are created. The image of the domain of these two Taylor Models then represents an initial box centered around the origin. Lastly, these Taylor Models are mapped by $\widetilde{M}$ in Taylor Model arithmetic, and the image of the resulting Taylor Models x_f and y_f is verified to lie within that of x_i and y_i.

COSY INFINITY provides intrinsic functions to calculate an outer interval enclosure of the range of a Taylor Model, so one can easily obtain intervals containing the images of x_f and y_f. To verify that these two intervals actually lie within the range of the initial Taylor Models is a little more complicated. To test for enclosure, it first is necessary to obtain an inner interval enclosure of the image of the initial Taylor Models x_i and y_i. In this case, those enclosures can be obtained easily, since the initial Taylor Models are constructed to have a simple structure.

The initial Taylor Models only consist of one linear coefficient and a remainder bound. To obtain an inner enclosure one can simply take the remainder bound and add, in interval arithmetic, the linear coefficient extracted from the Taylor Model. Thus one obtains an enclosure of the upper bound of the image of the Taylor Model. Taking the lower bound of that interval yields an upper bound of the inner enclosure. Applying the same technique, but this time subtracting the linear coefficient from the remainder bound, and taking the upper bound of the resulting interval one obtains a lower bound for the inner enclosure (see figure 4.2).

With the inner enclosure of the initial Taylor Models known, comparing those two inner interval bounds of the image of x_i and y_i to the outer interval enclosure of the image of x_f and y_f is trivial. If the outer range enclosures of the final coordinates are fully contained within the inner range enclosures of the initial coordinates, the initial box was mapped into itself.

4.2.2 Implementation and Results

Using COSY INFINITY[32], we implemented the algorithms described above for the Hénon map using the traditional double precision Taylor Models, as well as High Precision Intervals and High Precision Taylor Models (see Chapter 3). This allows a direct side-by-side comparison of the strengths and weaknesses of each of these verified numerical tools.

Implementation in each case is straightforward, since in two dimensions the calculation of eigenvectors and eigenvalues, as well as the matrix inversion, can be performed analytically. To ensure the matrix inversion is rigorous, it is carried out using either Taylor Models with only a constant part, or intervals. The result is a matrix with Taylor Model or interval entries, respectively.

For ease of notation, in the following results we will denote an interval of the form $[1.234\underline{5678}, 1.234\underline{6789}]$ by 1.234^{6789}_{5678} to both conserve space and make the width of the interval more clearly visible.

4.2.2.1 Double Precision Taylor Models

Theorem 4.7 (Existence of Period 15 Point in Hénon Map)**.** *Given the Hénon map with parameters $A = 1.422$ and $B = 0.3$, there exists at least one periodic point of period 15 within the interval*

$$\begin{aligned} X &= 1.1957^{80721557596}_{58008577504}, \\ Y &= 0.050^{52194963414509}_{49328335698421}, \end{aligned}$$

Proof. For the following calculations, Taylor Models of order 10 in two independent variables were used.

By repeated iteration of the origin by the map H in double precision arithmetic, a suitable candidate for the presumed periodic point is obtained. Mathematically there is no preference for any particular point in the orbit of the periodic point candidate. Numerically, however, it is favorable to choose a point that has a well conditioned eigenvector matrix. If that is the case, the inversion can be carried out nicely, and will not produce much overestimation.

After a non-verified heuristic analysis of the eigenvectors at each point in the orbit, we chose the following periodic point candidate:

$$\begin{aligned} x &\approx 1.195769365067588, \\ y &\approx 0.050507761649554453. \end{aligned}$$

Starting with the origin, this approximation required about 400 iterations through H^{15}, or equivalently 6000 iterations through H.

Using automatic differentiation with the DA datatype of COSY INFINITY, the linearization of H^{15} is computed, and the following approximation of the eigenvector matrix is derived

$$\begin{pmatrix} 0.8852161763463854 & 0.2504328280425341 \\ -0.4651798804061557 & 0.9681339776284161 \end{pmatrix},$$

which is then rigorously inverted using Cramer's rule in rigorous Taylor Model arithmetic on Taylor Models with only constant parts.

The x and y coordinates, in eigen-coordinates, of the initial box are chosen to be Taylor Models with only a linear coefficient of 10^{-5} in the first and second independent variable respectively. This initial box is then transformed from eigen-coordinates into Cartesian coordinates to yield the periodic point enclosure in Cartesian coordinates. This Taylor Model enclosure then is bounded, yielding the interval representation given in the statement of the theorem.

To conclude the proof, the initial box in Cartesian coordinates is mapped 15 times by H, converted back to eigen-coordinates, and then tested for inclusion in the initial box as described in Section 4.2.1.4. □

Note that, while seemingly counter-intuitive, an initial box as big as $2 \cdot 10^{-5}$ is required to successfully carry out the proof in double precision Taylor Model arithmetic. In fact, due to the specific structure of Taylor Models, the verification process fails if the initial box is chosen significantly smaller, due to the order loss effect (see Section 2.3.4). This problem can be overcome by using High Precision Taylor Models as described in Section 4.2.2.3.

It is also worth noting that the computational time required for this proof was negligible, taking only fractions of a second on a 2 GHz Core 2 Duo processor. The whole initial box is mapped at once, requiring only 15 consecutive evaluations of the map H in Taylor Model arithmetic and a few simple operations afterwards to verify the enclosure property.

4.2.2.2 High Precision Intervals

To obtain a more precise rigorous enclosure of the periodic point, we also implemented an interval version of the proof. As described in Section 4.2.1.3, it is necessary to use a High Precision interval package to successfully carry out this proof using intervals. We use the High Precision Intervals introduced in Chapter three.

Theorem 4.8 (High Precision Enclosure of Period 15 Point in Hénon Map)**.** *Given the Hénon map with parameters $A = 1.422$ and $B = 0.3$, there exists at least one periodic point of period 15 within the interval*

$$\begin{aligned} x &= 1.1957693650675503360411009839655489 \\ &\quad 352337235594806801053003707350839 68^{32853}_{10139}, \end{aligned}$$

$$\begin{aligned} y &= 0.0505076164955646488882884801756161 \\ &\quad 016841426808283706281410555165782 29^{4397960}_{1531331}. \end{aligned}$$

Before proving this theorem, an estimate of the computational complexity of the proof can be made. Mapping a single interval box around the presumed periodic point from eigen-coordinates into Cartesian coordinates, then through H^{15} in interval arithmetic, and back to eigen-coordinates, results a blow-up in the size of the box by a factor of about 1300 in x, and 1100 in y. This is not uncommon for interval arithmetic, and is to be expected given the type of map and the high period of the periodic point.

Mathematically H^{15} is a polynomial of order $2^{15} = 32768$. Of course the full expansion of H^{15} is never computed or used, instead H acts 15 times on the argument. While this reduces the total number of operations carried out significantly, in interval arithmetic this still results in a significant amount of overestimation because of the dependency problem.

These heuristics allow an estimate of the minimum number of boxes necessary to map the complete initial box into itself. Since the splitting always happens in both x and y, the maximal length the split boxes can have is about 1/1300 of the initial box size. Thus 1300^2, or about 1.7 million, is a lower bound for the number of boxes required to successfully complete the proof.

Proof. The calculations for this proof are carried out with intervals set to a precision of about 75 significant decimal digits. In the first step, a high precision approximation of the presumed periodic point is obtained by repeatedly iterating the origin through the map H. By iterating H until the same point as in 4.7 is reached, the following periodic point candidate is obtained:

$$\begin{aligned} x &\approx 1.19576936506755033604110098396554893 5\\ &\quad 23372355948068010530037073508396821495,\\ y &\approx 0.05050761649556464888828848017561610 16\\ &\quad 84142680828370628141055516578229296464 5. \end{aligned}$$

As before, using a double precision approximation of this point, automatic differentiation is used to calculate the linearization of H^{15} around this point. This process yields the following matrix of approximate eigenvectors:

$$\begin{pmatrix} 0.8852161763463209 & 0.2504328280405625 \\ -0.4651798804062782 & 0.9681339776289262 \end{pmatrix}.$$

While this step is performed in double precision since at the time this code was written the DA datatype of COSY INFINITY did not support high precision yet, this does not pose any problem. The transformation only requires approximate eigenvectors for the verification to be successful. This eigenvector matrix is then rigorously inverted in interval arithmetic using Cramer's rule on High Precision intervals.

The initial box in eigen-coordinates is chosen to be the interval $[-10^{-70}, 10^{-70}]$, both in x and y. This initial box is then transformed from eigen-coordinates to Cartesian coordinates to yield the actual periodic point enclosure given in the statement of the theorem.

To conclude the proof, the splitting-mapping-cycle described in Section 4.2.1.3 is initiated with the initial box on the stack and iterated until all boxes on the stack have been successfully mapped into the initial box. □

Note that, unlike the Taylor Model version of the proof, this process requires about 12 to 13 splits of the initial box in each direction in order to compensate the effects of overestimation in the interval arithmetic. This is consistent with the earlier heuristic estimate, as it results in a box size of about $1/2^{12} \approx 1/4000$. The proof takes about 130 minutes on a 2 GHz Apple MacBook with 2 GB of RAM to complete. In that time, about 70 million boxes are successfully mapped into the initial box.

4.2.2.3 High Precision Taylor Models

Theorem 4.9 (High Precision Taylor Model Enclosure of Period 15 Point in Hénon Map)**.** *Given the Hénon map with parameters* $A = 1.422$ *and* $B = 0.3$*, there exists at least one periodic point of period* 15 *within the interval*

$$\begin{aligned} X &= 1.19576936506755033604110098396 \\ &\quad 554893523372355948068010530037^{188}_{6812}, \\ Y &= 0.0505076164955646488882884801 \\ &\quad 75616101684142680828370628141 05^{797}_{403}, \end{aligned}$$

Proof. For the following calculations, High Precision Taylor Models of order 11 with a precision of about 75 in two independent variables were used.

By repeated iteration of the origin by the map H in High Precision Taylor Model arithmetic, a candidate for the same presumed periodic point as in 4.7 is computed:

$$\begin{aligned} x &\approx 1.19576936506755033604110098396 \\ &\quad 554893523372355948068010530037, \\ y &\approx 0.05050761649556464888828848017 \\ &\quad 561610168414268082837062814105555. \end{aligned}$$

Starting with the origin, this approximation required about 2400 iterations through H^{15}, or equivalently 36000 iterations through H.

Using automatic differentiation with the DA datatype of COSY INFINITY, the linearization of H^{15} is computed, and the following approximation of the eigenvector matrix is derived

$$\begin{pmatrix} 0.8852161763462412 & 0.2504328280382762 \\ -.4651798804064298 & 0.9681339776295176 \end{pmatrix}.$$

As in the case of High Precision intervals, this computation is performed using only double precision. This matrix is then rigorously inverted using Cramer's rule in rigorous High Precision Taylor Model arithmetic on High Precision Taylor Models with only constant parts.

The x and y coordinates, in eigen-coordinates, of the initial box are chosen to be High Precision Taylor Models with only a linear coefficient of 10^{-60} in the first and second independent variable respectively. This initial box is then transformed from eigen-coordinates into Cartesian coordinates to yield the periodic point enclosure in Cartesian coordinates. This High Precision Taylor Model enclosure then is bounded, yielding the interval representation given in the statement of the theorem.

To conclude the proof, the initial box in Cartesian coordinates is mapped 15 times by H, converted back to eigen-coordinates, and then tested for inclusion in the initial box. □

The computational time required for this proof of a High Precision enclosure still is negligible, amounting to about one second on a 2 GHz Apple MacBook. As was the case with the double precision Taylor Models, the whole initial box is mapped at once, requiring only 15 consecutive evaluations of the map H in High Precision Taylor Model arithmetic and a few simple operations afterwards to verify the enclosure property.

Almost all of this computation time is actually spent on the iteration of H to compute a suitable initial fixed point candidate. The actual verification step is virtually instantaneous.

4.2.3 Uniqueness and Attractiveness

Now existence of at least one periodic point within the intervals given in the above theorems is established. To establish uniqueness, Corollary 4.6 is applied to the Hénon map.

To evaluate the Jacobian of its fifteenth iterate, $J(H^{15})(x,y)$, the chain rule is applied repeatedly, yielding:

$$\begin{aligned} J(H^{15})(x,y) &= J(H^{14})(H(x,y)) \cdot JH(x,y) \\ &= JH(H^{14}(x,y)) \cdot JH(H^{13}(x,y)) \cdots JH(H(x,y)) \cdot JH(x,y). \end{aligned}$$

To compute the necessary bound of the operator norm of the Jacobian matrix $J(H^{15})$ of the Hénon map, equation 4.2 yields

$$\left\| J(H^{15}) \right\| = \sqrt{\lambda_{\text{max}} \left(J(H^{15})^T J(H^{15}) \right)},$$

where λ_{max} represents the largest eigenvalue. The eigenvalues of the 2×2 matrix $J(H^{15})^T J(H^{15})$ are computed using the analytic expression for the roots of the characteristic polynomial

$$\lambda^2 - tr(J(H^{15})^T J(H^{15}))\lambda + \det(J(H^{15})^T J(H^{15})) = 0$$

given by

$$\lambda_{1,2} = \frac{tr(J(H^{15})^T J(H^{15})) \pm \sqrt{tr(J(H^{15})^T J(H^{15}))^2 - 4 \cdot \det(J(H^{15})^T J(H^{15}))}}{2}.$$

Theorem 4.10 (Uniqueness and Attractiveness of Periodic Point)**.** *The enclosures of the* *period* 15 *point given in Theorems 4.7 and 4.8 contain exactly one periodic point of order* 15. *Furthermore, this periodic point is attractive within the given enclosures.*

Proof. The Jacobian of the Hénon map H is given explicitly by

$$JH(x,y) = \begin{pmatrix} -2Ax & 1 \\ B & 0 \end{pmatrix}.$$

The following computations are carried out in double precision Taylor Model arithmetic with computation order 10. The initial Taylor Model is chosen such that it covers the entire enclosure given in the statement of Theorem 4.7.

Computing JH^{15} as described above, results in a matrix with Taylor Model entries for JH^{15} representing a rigorous enclosure of the correct Jacobian of H^{15} valid over the entire fixed point enclosure. It is then trivial to compute the matrix product

$$A = J\left(H^{15}\right)^T J\left(H^{15}\right).$$

The eigenvalues of the resulting two dimensional matrix A are then calculated directly as Taylor Models and finally bounded by intervals. The resulting enclosures of the two eigenvalues e_1 and e_2 over the box given in Theorem 4.7 are:

$$\begin{aligned} e_1 &= [-0.8210105861696513 \cdot 10^{-9}, 0.8210017711776557 \cdot 10^{-9}] \\ e_2 &= [0.9002450764758135, 0.9809161063129307] \end{aligned}$$

Since both are bounded in absolute value from above by 1, so is $||J(H^{15})||$, and the map indeed represents a contraction. Together with the previously proven self mapping property from Theorem 4.7, this establishes all requirements for the Banach fixed point theorem, thus concluding the proof. □

First, note that by proving uniqueness within the rather large fixed point enclosure given in the statement of Theorem 4.7, uniqueness in the sharper high precision enclosures contained within it follows automatically.

Note further that for this computation double precision Taylor Models are the natural choice. They allow for computations to be performed over large volumes without significant overestimation. While it could have been performed in high precision, there is no need for High Precision Taylor Models in this part of the proof.

4.2.4 Conclusion

The results from this example, shown in Table 4.1, nicely demonstrate the power of the High Precision Taylor Models introduced in Chapter 3. While the mathematics of the proof are the same for both the Taylor Model based implementations as well as the interval based implementation, the algorithms required to successfully complete the proof in the interval case are significantly more complicated due to the splitting of the initial box required.

Furthermore, the time difference is striking. Both High Precision Taylor Models as well as regular double precision Taylor Models finish the proof in less than one second on an ordinary household computer. The High Precision intervals, however, take over two hours to complete the proof.

High Precision Taylor Models very efficiently counteract the effect of order loss that prevents the computation of sharp enclosures in double precision Taylor Model arithmetic. However, High Precision Taylor Models still exhibit order loss, so that, at the same precision,

	Double Precision Taylor Models	High Precision Taylor Models	High Precision Intervals
Approx. Enclosure Width	10^{-5}	10^{-60}	10^{-69}
Computation Time	< 1 sec.	1 sec.	130 min.
Number of Boxes	1	1	$\sim$ 16.7 million

Table 4.1: Comparison of the performance of double precision Taylor Models, High Precision Taylor Models and High Precision intervals in this example.

High Precision intervals provide a slightly sharper enclosure at a much higher cost. Increasing the precision of the Taylor Model computation easily overcomes this problem.

4.3 Global Fixed Point Finder

In the previous example, a good approximation of the presumed fixed point was known, and only its verification had to be performed rigorously. Simple numerical techniques, such as Newton methods, have been around for a long time and are a well known tool for identifying isolated, approximate fixed points.

However, these techniques suffer from a major drawback. Given a starting point, they converge to a single fixed or periodic point approximation, but there is no information obtained about possible other fixed or periodic points in a given area. In more general terms, one can ask for sharp, verified enclosures of *all* periodic points of given order $p > 0$ of a map M in some compact set. Based on the verification tools developed in the previous section, such an automated global fixed point finder can be implemented[46].

As before, let $K \in \mathbb{R}^n$ be a compact set and $M : K \to \mathbb{R}^n$, $n \in \mathbb{N}$, a continuously differentiable map. The goal is to identify enclosures of all periodic points of given order $p > 0$ of M in K. That is, we want to find a collection X of sets $X_i \subset K$ such that for each such set $X_i \in X$ there exists some $\vec{x} \in X_i$ such that

$$M^p(\vec{x}) = \vec{x}, \tag{4.8}$$

and conversely, that for any $\vec{x} \in K$ satisfying equation 4.8 there is some i such that

$$\vec{x} \in X_i. \tag{4.9}$$

It is then possible to further classify each $X_i \in X$ by whether the fixed point contained in it is unique or not.

Unfortunately, as is typical for verified computation, there is also the possibility that for some set $Y \subset K$ neither of the above statements can be made. That is, there is an additional collection Y of "undecided" sets Y_i for which it is not known wether they contain any periodic points or not.

Since the case $p > 1$ coincides with $\vec{x}$ being a fixed point of the map M^p, without loss of generality only fixed points need to be considered.

4.3.1 Preconditioning

Theorem 4.1 is well suited for proving the existence of a fixed point through numerical computations. However, without any preconditioning it only succeeds in very few cases. Clearly, in the case of a hyperbolic, elliptic, or repelling fixed point, Theorem 4.1 does not apply directly at all, as there is generally no set C around the fixed point that is directly mapped into itself.

But even if applied directly to an attractive fixed point, the self mapping property may not succeed, as described in Section 4.2.1.1. In this case it is sufficient to switch to a coordinate system aligning the coordinate axes to the eigenvectors of the derivative.

There is, however, a much more powerful method to transform the map into a form that is applicable to Theorem 4.1 loosely based on the well known Newton Method (see Section 3.1.1) used to approximate roots of differentiable functions.

Consider the following operator acting on continuously differentiable functions $f : K \subset \mathbb{R}^n \to \mathbb{R}^n$

$$\mathcal{C}_A : f \mapsto A \cdot (f - \mathrm{id}) + \mathrm{id}, \tag{4.10}$$

where A is a regular $n \times n$ matrix and id represents the identity operator on $\mathbb{R}^n$.

Since

$$f(x) - x = 0 \Leftrightarrow A\,(f(x) - x) = 0$$

for any regular $n \times n$ matrix A, we find that any fixed point of the map $\mathcal{C}_A f$ is also a fixed point of the map f and vice versa. We say that f and $\mathcal{C}_A f$ are equivalent with respect to their fixed points.

Assume x is a fixed point of f and hence $\mathcal{C}_A f$. We want to choose the matrix A in such a way, that x becomes a strongly attracting fixed point of $\mathcal{C}_A f$. To that extend, consider the derivative of $\mathcal{C}_A f$:

$$D(\mathcal{C}_A f) = A \cdot (Df - I) + I.$$

If $Df(x) - I$ is a regular matrix, i.e. 1 is not a singular value of Df, then setting $A = -(Df(x) - I)^{-1}$ yields

$$D(\mathcal{C}_A f)(x) = -I + I = 0.$$

Hence with this choice of A, $\mathcal{C}_A f$ has a strongly contracting fixed point at x. Note that this is true independently of the type of fixed point the map f has at x. Furthermore, if $Df - I$ is not regular, it is possible to perturb it a little such that it becomes invertible. In that case, the derivative of $\mathcal{C}_A f$ will not vanish completely, but if the perturbation was is small enough, $\mathcal{C}_A f$ will still be a strong contraction.

Thus, by applying Theorem 4.1 on $\mathcal{C}_A f$ and a non-empty, convex, compact set $C \subset \mathcal{K}$ around x, it is possible to verify the required self mapping property much more easily than by direct application to f.

The operator $\mathcal{C}_A$ is also very advantageous in the proof of uniqueness of a fixed point in C using Theorem 4.2. Since A is chosen such that the derivative of $\mathcal{C}_A f$ vanishes at the fixed point x, by continuity the ∞-norm of $D(\mathcal{C}_A f)$ is small on all of C, provided C itself is sufficiently small. It is then possible to apply Lemma 4.5 to easily show uniqueness of the fixed point of $\mathcal{C}_A f$, and thus of f, in C by bounding $D(\mathcal{C}_A f)$ over C and utilizing inequality 4.3.

The effect of this preconditioning is shown in figure 4.3. The dashed box has width 10^{-4} in each direction in cartesian coordinates and is centered around one of the period 15 points of the Hénon map used in Section 4.2. The solid black object indicates the return of that box after 15 iterations. Without preconditioning, the resulting object is mostly linear with relatively small non-linear curvature. Clearly, the image does not map into the original box. In the preconditioned case, the same initial box is shrunk into a very highly non-linear object at least one order of magnitude less in size than the original box.

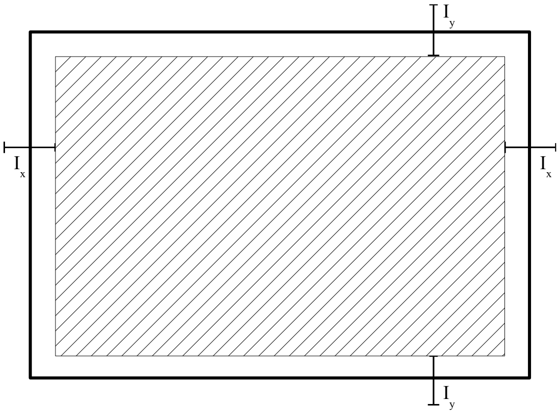


Figure 4.2: The image of a Taylor Model box (solid black), the interval bounds I_x and I_y, and the rigorous inner enclosure (hatched).

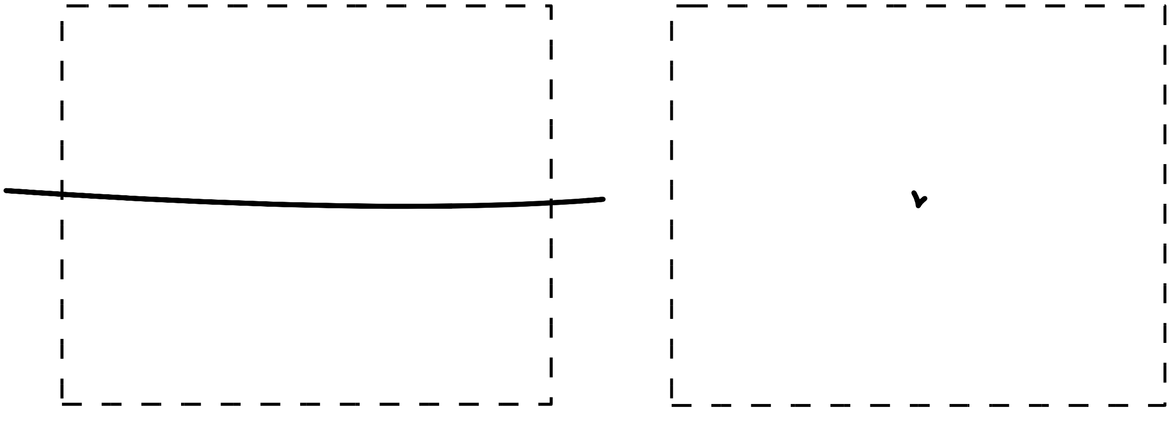


Figure 4.3: The effect of preconditioning. A box of size $10^{-4} \times 10^{-4}$ in a Hénon map and its 15-th iterate without (left) and with (right) preconditioning.

4.3.2 Implementation

The operator described above provides a versatile method to classify single boxes around a fixed point. In the following, we extend this to an algorithm for a global fixed point finder as described in the introduction to this Section. To easily satisfy the requirement of Theorem 4.1 that sets must be convex, compact and non-empty, our implementation will only operate on interval boxes. Interval boxes are subsets of $\mathbb{R}^n$ given by a non-empty, finite interval in each component, i.e.

$$B = [\underline{b}_1, \overline{b}_1] \times [\underline{b}_2, \overline{b}_2] \times \cdot \times [\underline{b}_n, \overline{b}_n]$$

Clearly, interval boxes are convex, and if $\underline{b}_i, \overline{b}_i \in \mathbb{R}\ \forall i = 1, \ldots, n$ they are compact, and if $\underline{b}_i < \overline{b}_i\ \forall i = 1, \ldots, n$ they are non-empty.

Let K be the interval box within which one wants to identify all fixed points. The simplest method is to divide the search space into an even grid of fixed size boxes, and then checking each single small box for a fixed point using the classification just presented. However, even in two dimensional maps, that approach often is prohibitively expensive in terms of computations required. In order to compute all fixed points in a box of size 1×1 to an accuracy of 10^{-4}, already $10^4 \cdot 10^4 = 10^8$ boxes must be checked.

Instead of fixing a grid, it is much more advantageous to adaptively select the size of boxes to test in each region of the map based on the complexity of the map in that region. We use a stack-based "divide-and-conquer" style algorithm to successively remove parts of the search space that are guaranteed not to contain fixed points. This algorithm is similar in structure to the algorithm used for the High Precision interval verification in Section 4.2.1.3.

Each box is classified based on the following criteria:

1. Boxes that are mapped completely outside of themselves, i.e. boxes B for which it can

be verified that $M(B) \cap B = \emptyset$, certainly contain no fixed points. These boxes are discarded without further testing.

2. Boxes for which there is no conclusive result, i.e. boxes for which $M(B) \cap B \neq \emptyset$ but not $M(B) \subset B$. These boxes may or may not contain a fixed point, and they are split into smaller boxes which are pushed onto the stack. Analysis then continues with one of those smaller boxes.

3. Boxes which contain a fixed point, i.e. boxes for which $M(B) \subset B$. These boxes may be split further to obtain sharper enclosures if their size is too large. Otherwise, these boxes are then tested for uniqueness of their fixed point using Corollary 4.6.

At this point, it is intentionally left open how exactly the "size" of an interval box is measured. Such a measure depends on the application. Commonly employed measures include the volume of the interval box, or the length of the longest side. Furthermore, it is not necessary at this point to specify an algorithm for the "splitting" of a box, as long as the employed algorithm results in a reduced measure of each of the split boxes. A simple splitting algorithm is to divide one of the longest sides of the box in half.

To guarantee that the above search for fixed points finishes, there are certain pre-prescribed thresholds relating to the size of the boxes under consideration. The first is the desired maximum size of the verified fixed point enclosures, denoted by Ω_1. All boxes that contain or may contain fixed points will be at most of this size. Boxes larger than this size in step 3 are split further.

The second constant, Ω_2, provides a lower limit on the size of boxes that are considered. Once the size of a box falls below this threshold, it will not be split any further, independently of its current status. We will discuss these limits in more detail in Section 4.3.4.

4.3.2.1 Algorithm

The basic algorithm for the verification procedure with automatic box size control is as follows:

1. Start with initial box K on the stack
2. Take the top box S off of the stack and classify the box:
   - If S is verified to contain no fixed points, discard it.
   - Else, if the size of the S is larger than Ω_1, split S and push the two resulting pieces onto the top of the stack.
   - Else, if the size of S is larger than Ω_2, split along the longest side of S and push the two pieces onto the top of the stack.
   - Else, if there is a unique fixed point in S, store S in the list of found unique fixed points.
   - Else, if there is at least one fixed point in S, store S in the list of possibly non-unique fixed points.
   - Otherwise, place S in the list of undecided boxes.
3. Continue with 2 until there are no more boxes on the stack.

In practice, the classification of a box S is performed in several steps, checking after each if further tests are necessary. The following order of tests is the most efficient:

1. Compute $M(S)$ in Taylor Model arithmetic. If $M(S) \bigcap S = \emptyset$, stop as there is certainly no fixed point of M in S.

2. Compute an approximate $A = (DM(x) - I)^{-1}$ in floating point arithmetic, for some $x \in S$. If the inverse does not exist or is very badly conditioned, set A to a more suitable "approximate" inverse, or just the identity.

3. Compute $\mathcal{C}_A M(S)$ in Taylor Model arithmetic. If $\mathcal{C}_A M(S) \not\subset S$, stop as existence is not verified.

4. Compute $D\left(\mathcal{C}_A M\right)(S)$ in Taylor Model arithmetic using a provided analytic expression for DM and subsequently test the bound on the operator norm of the derivative to be less than one to verify uniqueness.

4.3.3 Analysis

The use of Taylor Models to carry out the verified computations makes the computation of the matrix A in the construction of the operator $\mathcal{C}_A$ particularly easy. The initial Taylor Model used to evaluate M only consists of a constant part and a linear part in one variable for each dimension. Thus, very good approximations of the derivatives at the center point of the box can be obtained very easily from the first order coefficients of the resulting Taylor Model. Due to the differential algebra approach of Taylor Models, no separate computation is necessary.

The tests for inclusion of a Taylor Model in the original set are done by computing interval bounds on each Taylor Model. In general, it is true that interval enclosures of arbitrary Taylor Models suffer from the wrapping effect, and are substantially larger than the Taylor Model itself. In this particular case, however, the use of interval bounds and the associated wrapping effect introduces no additional overestimation beyond that of the quality of the generated interval enclosure. This is because the initial box S is itself an interval box. Thus for $\mathcal{C}_A M(S)$ to lie within S, each separate coordinate bound of $\mathcal{C}_A M(S)$ must lie in the corresponding interval of S.

Note that the use of Taylor Models for these computations is essential for another reason. For the contraction operator $\mathcal{C}_A$ to work properly, it is required to have strong control over the dependency problem. This is because $\mathcal{C}_A$ is constructed above such that the first derivative is nearly cancelled out. This cancellation, however, can only occur if it is computed symbolically, e.g. using Taylor Models. With mere interval evaluation, the dependency problem caused by this cancellation would lead to vast blowup of the resulting interval, and thus render the operator ineffective.

The reader familiar with verified global optimization will find the above type of algorithm very familiar. Most verified global optimizers, such as for example COSY GO[33], use very similar methods to successively examine the entire search space and prune regions where there is no extremum. However, it is worth pointing out one big difference between such global optimizers and our fixed point finder. Global optimizers produce a list of small enclosures where there they cannot rule out the existence of an extremal point. However, there is usually no statement made about which box contains the global extremum, or if a box really contains one at all.

This global fixed point finder, on the other hand, not only produces a list of enclosures of fixed points, but also verifies and classifies them in the same step, thus typically guaranteeing the existence and in most cases also the uniqueness of a fixed point in each resulting box.

4.3.4 Constants

As described above, there are two constants controlling the behavior of the algorithm with respect to box sizes. The first constant, Ω_1, is the maximum size of boxes stored in the results list. Any boxes larger than this size are either discarded because they do not contain fixed points, or are split into smaller boxes to be analyzed. To the user, this parameter specifies the desired accuracy of the fixed point enclosures.

The second constant, Ω_2, is the minimum size below which a box is never split any further. Boxes that have been split blow this size, and could not be classified yet, will be added to the "unclassified" list. From the user's perspective, this value is a cutoff value when the algorithm should give up attempts to classify a box.

Therefore, resulting boxes in any of the three lists (unique fixed points, fixed points, and undecided) will always range in size between Ω_1 and Ω_2.

The ideal choice of these constants is problem dependent and can greatly affect the performance of the overall algorithm. In general, of course, $\Omega_1 \geqslant \Omega_2$ should hold. In practice, however, there is a lower limit for the size of Ω_1 below which the algorithm becomes significantly less effective due to limited floating point precision caused by the order loss effect (see Section 2.3.4). In our experiments, a good choice for the constant Ω_1 in double precision computations is about 10^{-7}.

4.3.5 Example

To demonstrate the effectiveness of this global periodic point finder, we apply this algorithm to the standard Hénon map with $A = 1.4$ and $B = 0.3$. The approximate attractor is covered by 8 interval boxes as shown in blue in figure 4.4. Within these initial boxes the

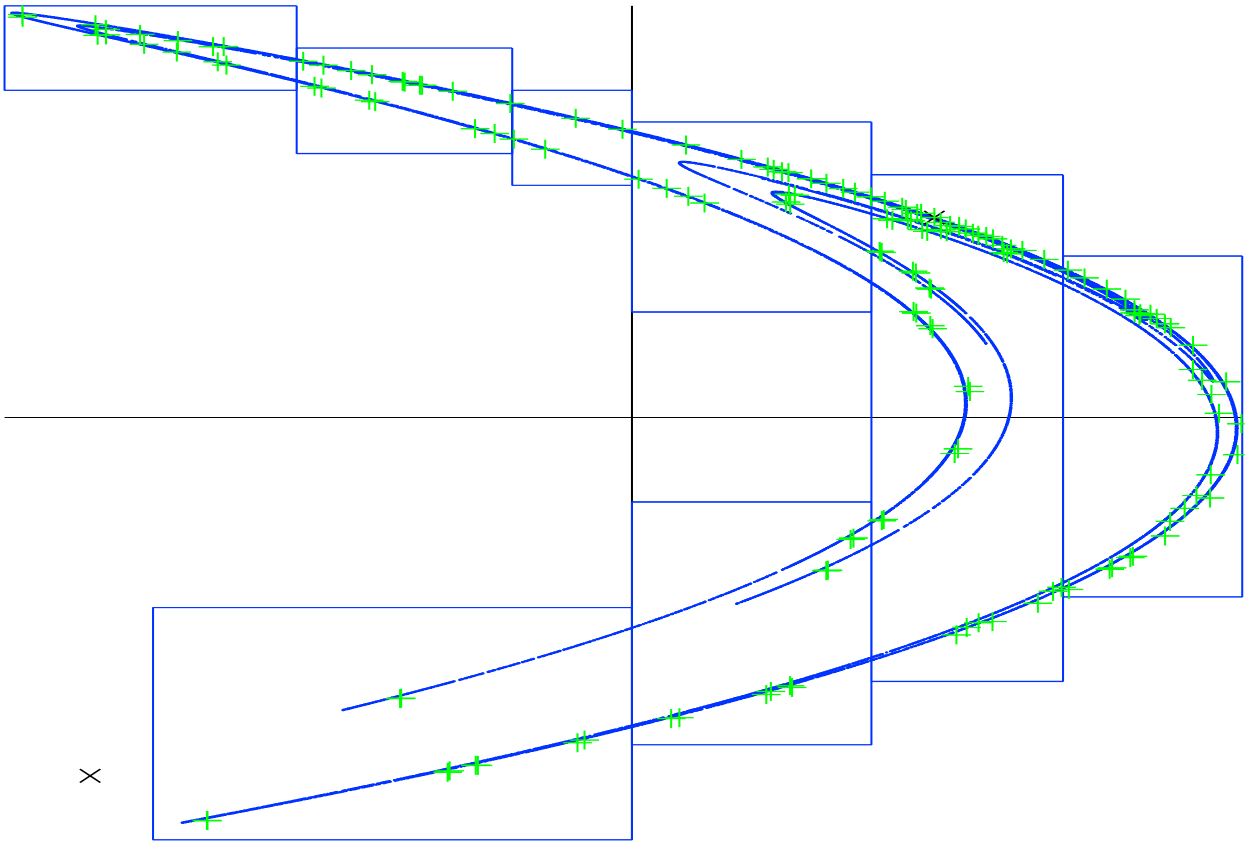


Figure 4.4: All period 11 periodic points (green) and the two fixed points (black) within initial boxes (blue) in the standard Hénon map.

global periodic point finder was started to identify all fixed points of period 11.

The computation took 54 seconds running on a single core of our Intel Xeon X5677 system at 3.5 GHz. There were 155 resulting periodic points found in these boxes. This corresponds to 14 orbits of period 11 and one of the two fixed points of the Hénon map. figure 4.4 shows each of these periodic points in green, which were computed to an accuracy of about 10^{-5}, which is below the printer resolution. Computations were carried out using 5-th order double precision Taylor Models.

4.4 Application to Beam Physics

The study of fixed points and periodic points of maps, such as transfer maps for beam lines, can reveal lots of information about the behavior of such a dynamical system. In accelerator physics, undesirable chaotic behavior in an accelerator is typically observed near periodic points. One major design goal in accelerator construction is therefore to avoid low-period periodic points in the transfer map.

In order to do so, it is of course first necessary to find the fixed and periodic points of a given map. Automated detection of such periodic points can be of great help with this task. To illustrate this, we will now apply the rigorous fixed point finder to a real world transfer map of the Tevatron accelerator at Fermilab. This map is represented as a polynomial of order 11, presenting particular numerical challenges to overcome.

4.4.1 Implementation

The transfer map is an 11-th order map, computed by Pavel Snopok using the COSY INFINITY beam physics package[42]. The model of the Tevatron used is a slightly simplified

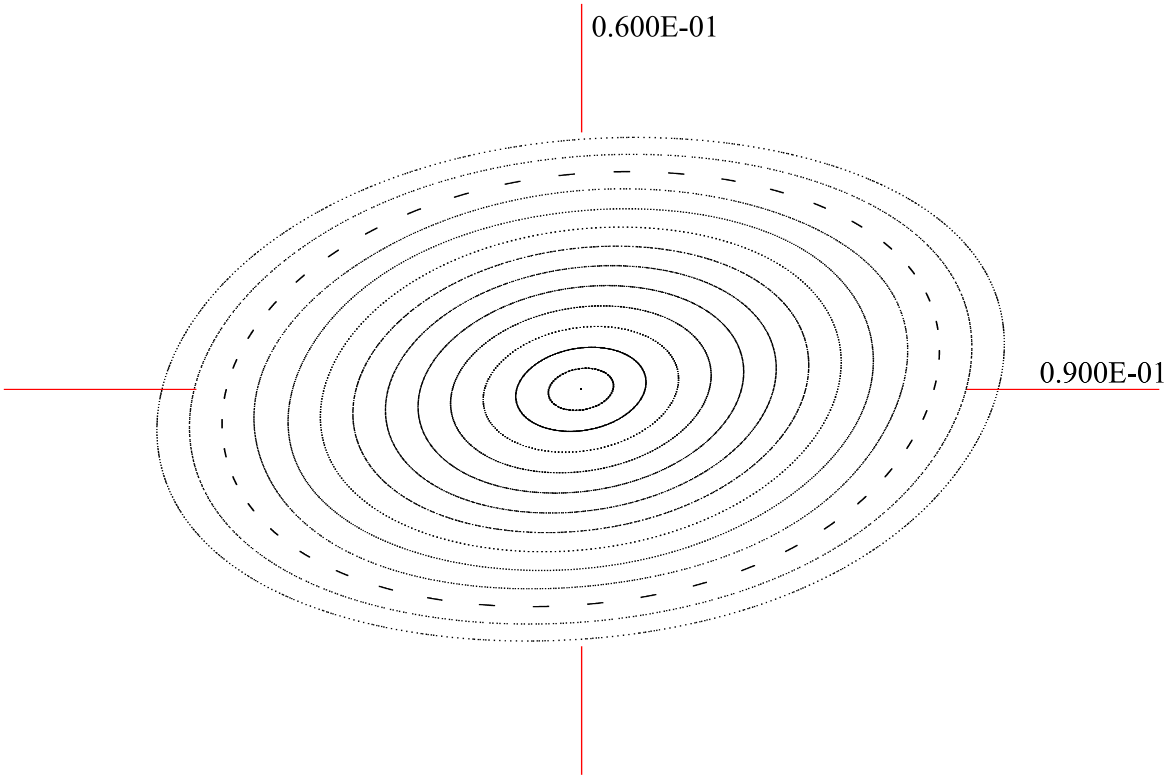


Figure 4.5: Tracking of evenly spaced particles through the transfer map in the $x - a$ plane (1400 iterations, 5 mm spacing between rings).

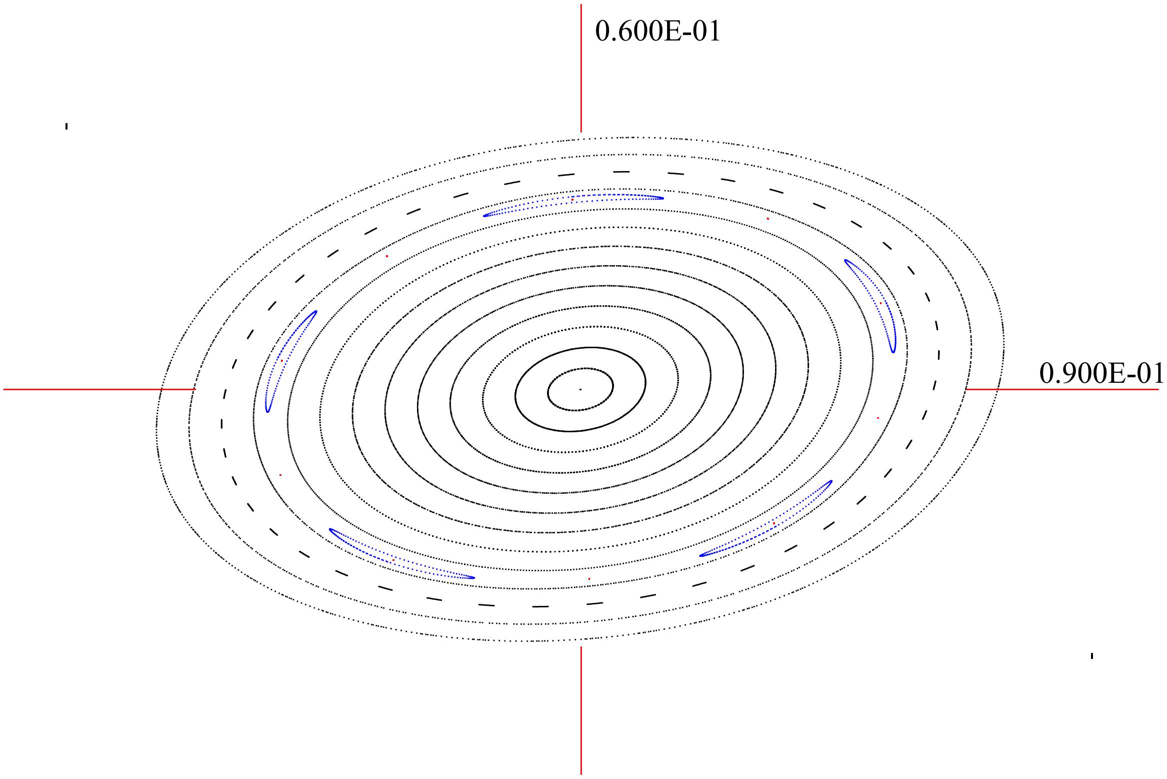


Figure 4.6: Location of the periodic points identified in the transfer map. Red are two sets of period 5 points, in blue are particles trapped in the islands around the elliptic period 5 points. Two period 8 points are in black at the top left and bottom right corner.

model consisting of all dipole, quadrupole and sextupole bending magnets, but omitting certain corrective beam line elements, such as electrostatic separators and solenoids. This omission still produces a very close approximation of the real dynamical system. Moreover, it results in a symmetric map in which the x-a plane is invariant.

For this application, we restrict ourselves to the two dimensional motion in the invariant x-a plane. For this purpose, the from full four dimensional map the two dimensional map of the x-a plane into itself is extracted.

Using the particle tracking facilities of COSY's beam physics package, a tracking picture of the system using evenly spaced initial particle coordinates is generated (figure 4.5). Estimating the region of interest to be within the region shown, $[-0.09, 0.09] \times [-0.06, 0.06]$, the periodic point finder is started to work on that region.

Table 4.2 shows the number of enclosures containing a unique periodic point found for periods up to 20. As all COSY beam maps, this map is origin preserving since the origin represents the reference trajectory. Hence for every period, there is at least one periodic point identified, namely the origin. Similarly, periodic points of period 5 are also listed for periods 10, 15, and 20. But since all of the periodic point enclosures identified contain exactly one periodic point, and we know that periodic points of lower period must be present, we can subtract out those points of lower period from the results. The count of periodic points without those duplicate points is given in the second to last column.

This calculation rigorously proves that there are only periodic points of periods 5 and 8 within the region of interest. Note that for the period 5 points, there are two separate sets of 5 points each, while for the period 8 points only two were found. The missing 6 points of their orbit lie outside the search area and thus are not identified. Figure 4.6 illustrates the location of all periodic points identified up to period 20.

Due to the splitting algorithm we chose in our implementation, always cutting along the longest side of a box, the resulting periodic point enclosures are roughly square. Their area is less than the parameter Ω_1 specified, given in the area column. As is expected due to the broken scaling law it was necessary to increase the targeted area as the period increases to guarantee successful verification. Note, however, that even the enclosures at period 20 still are accurate up to an error of about 10^{-6}.

The time given is the CPU time consumed for the search rounded to the nearest full second and measured running on a single core of our Intel Xeon X5677 system at 3.5 GHz. It is interesting to note the vast differences in execution times between different periods. Particularly periods 13 and 18 take significantly longer than all other periods. This phenomenon will be analyzed in more detail in the next section.

4.4.2 Tune Shifts

The Tevatron map analyzed exhibits a linear tune of about 0.5915. The accelerator was intentionally designed such that the linear tune is close to the fundamental resonance $3/5 = 0.6$. Normal form theory predicts a shift of the tune as the distance form the reference orbit increases. Those tune shifts can help explain why we found period 5 periodic points.

Normal form transformations based on differential algebra methods[6] applied to our beam transfer map $\mathcal{M} : \mathbb{R}^2 \to \mathbb{R}^2$ provide a coordinate transformation $\mathcal{T} : \mathbb{R}^2 \to \mathbb{R}^2$ such that the map $\mathcal{T} \circ \mathcal{M} \circ \mathcal{T}^{-1}$ takes elliptic orbits into circular orbits around the origin up to a given order, and the tune shift $t(r)$ only depends on the radius of the circle[5].

When the tune shift $t(r)$ attains a rational value of the form n/k with $n, k \in \mathbb{N}$, pe-

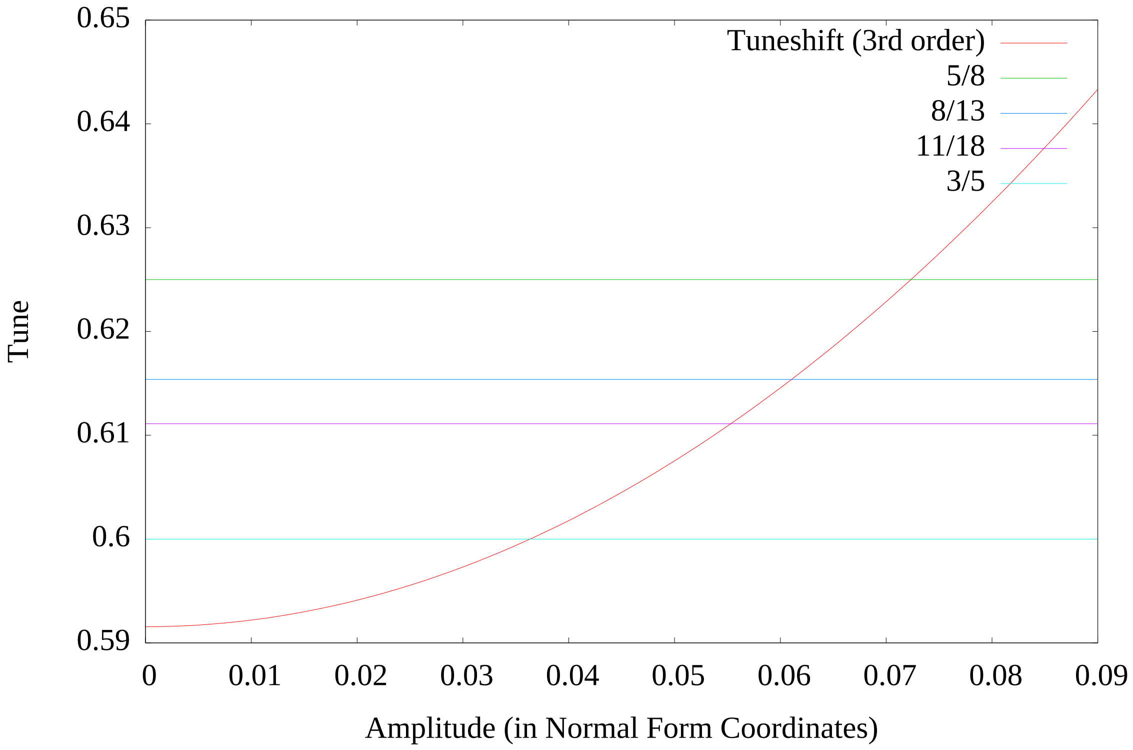


Figure 4.7: Polynomial expansion of tune shifts computed to orders 3.

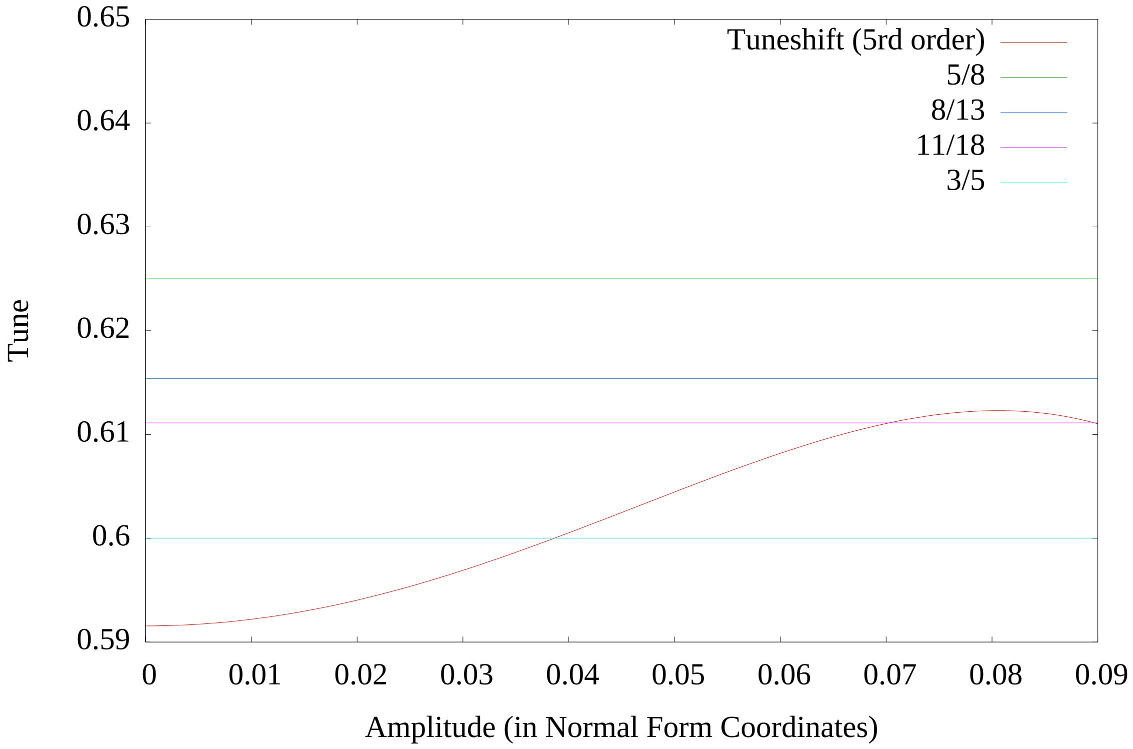


Figure 4.8: Polynomial expansion of tune shifts computed to order 5

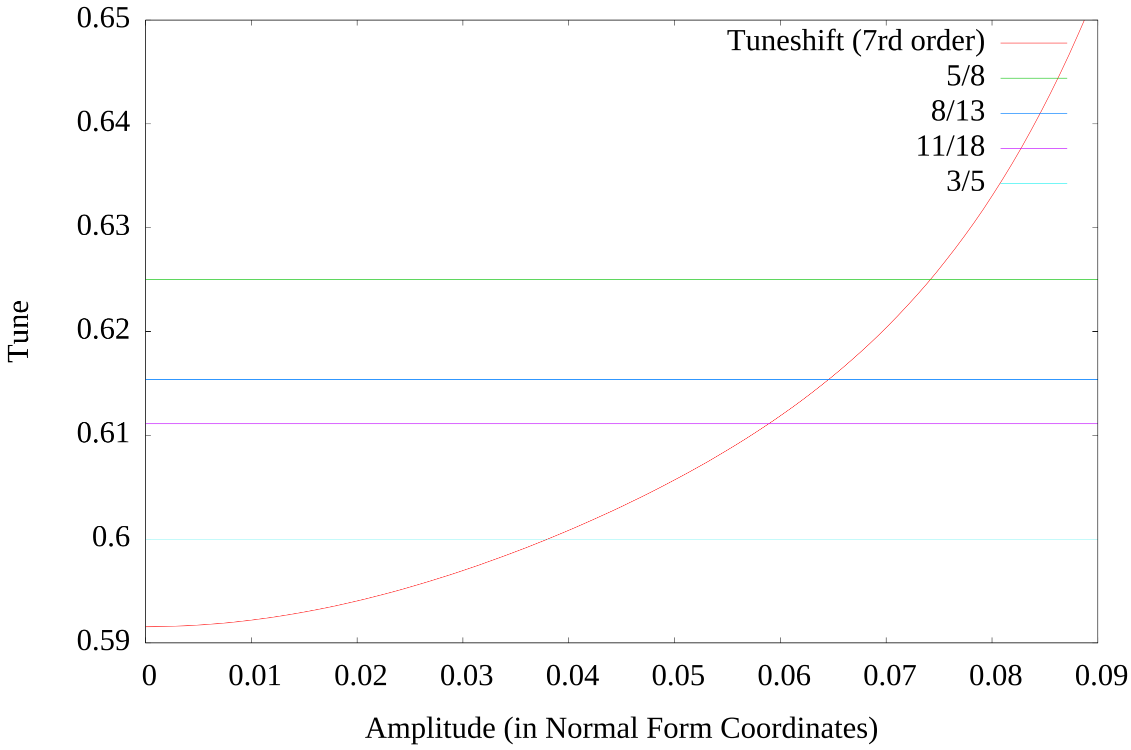


Figure 4.9: Polynomial expansion of tune shifts computed to order 7

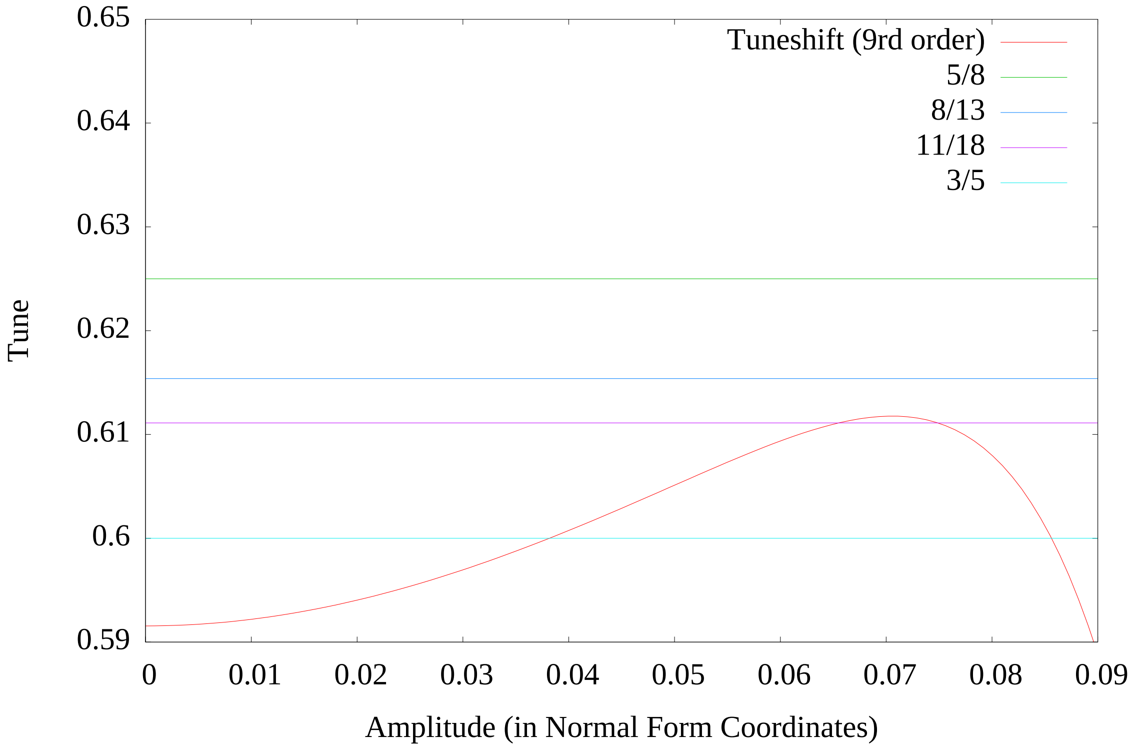


Figure 4.10: Polynomial expansion of tune shifts computed to order 9

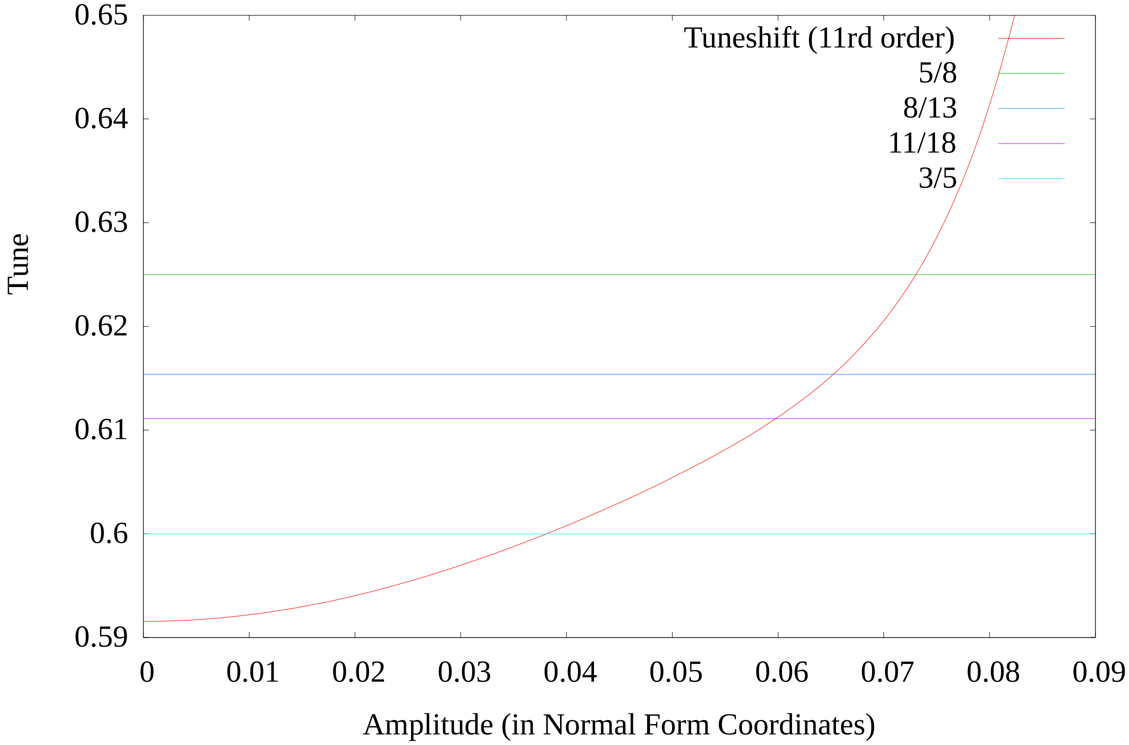


Figure 4.11: Polynomial expansion of tune shifts computed to order 11.

riodic points of period k are expected to appear in the map in normal form coordinates. Consequently, the original map $\mathcal{M}$ also should exhibit periodic points of the same period.

We used the COSY INFINITY beam physics code to compute an expansion of the tune shift in normal form coordinates[6] to various orders. In figures 4.7 through 4.11, those tune shifts are plotted as a function of the radius from the reference orbit in normal form coordinates. Also marked are several interesting resonances.

The wild oscillations of the function at different orders for large radii indicate that the polynomial expansion of the tune shift does not converge very well for values above approximately 0.06. However, for values below about 0.05 the plots are in good agreement, suggesting good convergence in that region.

The last column of Table 4.2 shows the nearest fraction $n/p, n \in \mathbb{N}$ for each period p that is larger than the tune of the Tevatron, 0.5915. We chose to list larger values as the tune appears to be shifting upwards in the tune shift plots. As mentioned before, the resonance for the period 5 points is by design very close to the tune of the Tevatron. This is because then automatically other low order resonances occur significantly further away from the Tevatron tune.

The next two closest resonances, occurring for periods 13 and 18, are $8/13 \approx 0.615$ and $11/18 \approx 0.611$. Interestingly enough, those two periods were also the two periods taking the longest for the global periodic point finder to treat, taking up to 12 times longer than all other periods. This indicates that those periods are "closer" to produce periodic points, i.e. points do return to within very close proximity of their origin. However, our periodic point finder proved that none actually return exactly to themselves (except for the origin). This complicates the search for our algorithm and results in significantly longer execution times.

From the normal form tune shift plots in figure 4.7 we can determine the radius for which the tune shift reaches $3/5 = 0.6$. In all orders it is consistently located at about 0.0387. At this tune in the normal form we would expect period 5 points to appear. To compare this prediction with the results of the periodic point finder, we transformed the coordinates (x, a) of the center of one of the identified period 5 enclosures into normal form coordinates $(X, A) = \mathcal{T}(x, a)$ and computed its radius $R = \sqrt{X^2 + A^2}$.

The point we chose for this is $x \approx 0.00137$ and $a \approx -0.0295$, which, in COSY's normal form coordinates, becomes $X \approx 0.00631$ and $A \approx -0.0375$, and thus has a radius $R \approx 0.0381$. As can be seen, both the radius predicted from the normal form tune shift plot as well as the periodic point identified by the periodic point finder are in agreement within about 2% of each other.

Unfortunately, this is the only valid prediction that can be made from the tune shift plot. In all of the plots, the next tune, 11/18, seems to be crossed. However, the periodic point finder proved that there are no periodic points of period 18 related to this tune within the search region.

This is probably caused by the slow convergence of the normal form transformation due to the closeness of the linear tune to the 3/5 resonance. This causes the tune shift expansions shown above to not converge for the values at which 11/18 would be obtained.

Converting one of the period 8 points we identified by the same normal form coordinate transformation as used for the period 5 point above, we find that its amplitude would be about 0.238. This is clearly far beyond the convergence of the normal form expansion, hence its existence could not be predicted by the tune shift plots.

This comparison highlights a general problem associated with the normal form methods applied to this type of problem. Since accelerators are intentionally designed to operate near one of the fundamental resonances, the convergence of the tune shift expansion is slow and the region in which reliable predictions can be made is rather small. Unfortunately, with non-verified computations it is not clear a priori how large that region is. Verified tools, such as this periodic point finder, can help in identifying rigorously such regions.

4.5 Conclusion

We succeeded in implementing a fast, rigorous, global periodic point finder. The use of Taylor Models produces fast results even for high period points in numerically difficult maps. The main restriction to obtaining sharper enclosures and higher periods is the limited precision of the underlying double precision floating point numbers. We have shown that by using

High Precision Taylor Models the effect of order loss associated with limited precision can be addressed and very sharp enclosures are possible.

Applying these techniques to dynamical systems such as beam transfer maps allows the automated analysis of those maps for regions of stability and aides in identifying regions of potential chaos. The rigorous character of the method presented augments the qualitative results obtained from existing methods of analysis.

In future work, the rigorous global fixed point finder will be reworked to use High Precision Taylor Models. It can then be applied to a wide range of problems. In its reincarnation as a global rigorous verified zero finder, it can for example be used to compute rigorous enclosures of intersections between Taylor Models, which is useful in automatically enclosing homoclinic intersections e.g. between manifold enclosures.

Period	Unique boxes	Area (Ω_1)	Time (sec)	Periodic points	Tune
1	1	$4 \cdot 10^{-19}$	0	1	0.592
2	1	$4 \cdot 10^{-19}$	0	0	1
3	1	$4 \cdot 10^{-19}$	0	0	$2/3 \approx 0.666$
4	1	$4 \cdot 10^{-19}$	0	0	$3/4 = 0.75$
5	11	$4 \cdot 10^{-19}$	6	10	$3/5 = 0.6$
6	1	$4 \cdot 10^{-19}$	0	0	$4/6 \approx 0.666$
7	1	$4 \cdot 10^{-19}$	0	0	$5/7 \approx 0.714$
8	3	$4 \cdot 10^{-19}$	6	2	$5/8 = 0.625$
9	1	$4 \cdot 10^{-19}$	1	0	$6/9 \approx 0.666$
10	11	$4 \cdot 10^{-16}$	13	0	$6/10 = 0.6$
11	1	$4 \cdot 10^{-16}$	1	0	$7/11 \approx 0.636$
12	1	$4 \cdot 10^{-16}$	2	0	$8/12 \approx 0.666$
13	1	$4 \cdot 10^{-16}$	107	0	$8/13 \approx 0.615$
14	1	$4 \cdot 10^{-16}$	2	0	$9/14 \approx 0.643$
15	11	$4 \cdot 10^{-16}$	21	0	$9/15 = 0.6$
16	3	$4 \cdot 10^{-12}$	18	0	$10/16 = 0.625$
17	1	$4 \cdot 10^{-12}$	4	0	$11/17 \approx 0.647$
18	1	$4 \cdot 10^{-12}$	359	0	$11/18 \approx 0.611$
19	1	$4 \cdot 10^{-12}$	10	0	$12/19 \approx 0.632$
20	11	$4 \cdot 10^{-12}$	33	0	$12/20 = 0.6$

Table 4.2: Periodic points found for periods from 1 through 20 in the region $[-0.09, 0.09] \times [-0.06, 0.06]$ in the Tevatron transfer map.

Chapter 5

Rigorous Manifold Enclosures

In this chapter we will introduce methods to obtain tight, verified enclosures of local invariant manifold of various dynamical systems. This work is based on previous work by Johannes Grote in his dissertation[19] and in a joint paper[47], which describes a method to compute verified enclosures of manifolds of planar diffeomorphisms.

We extend the methods introduced there, to provide an algorithm for the construction of a polynomial manifold approximation in arbitrary dimension to both discrete maps as well as autonomous ordinary differential equations with proper treatment of resonances. We then proceed to describe the rigorous verification of a very sharp enclosure of the invariant manifolds of dynamical systems in three dimensions, which can be extended to higher dimensional cases as well. Lastly, we examine the inclusion of parameter dependence in the manifold generation and verification, to construct polynomial approximations of the manifolds of maps depending on parameters.

5.1 General Concepts

First, we will briefly introduce the general concept of invariant manifolds and describe some well known properties thereof. Those concepts introduced here will be used in the construction and verification of the manifold later on.

While well known, proofs will be provided for some of the theorems presented in this section, as they provide some valuable insight into invariant manifolds and their properties.

5.1.1 Invariant Manifolds of a Diffeomorphism

In the following, let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a diffeomorphism, i.e. f is a bijection such that both f and f^{-1} are continuously differentiable.

Furthermore, assume $f(0) = 0$ and the eigenvalues $\lambda_i, i = 1, \ldots, N_u$ and $\mu_j, j = 1, \ldots, N_s$, $N_u + N_s = n$, of $Df|_0$ are real and satisfy

$$\begin{aligned} |\lambda_i| &> 1 \ \forall i = 1, \ldots, N_u, \\ |\mu_j| &< 1 \ \forall j = 1, \ldots, N_s. \end{aligned}$$

Without loss of generality, let the eigenvectors corresponding to λ_i be aligned with the x_i-axis and the eigenvectors corresponding to the μ_j be aligned with the x_{N_u+j}-axis. Given the above assumptions, this can always be achieved by a suitable linear coordinate transformation.

Remark 5.1. The restriction to real eigenvalues is not of mathematical nature. The following theorems hold as well for complex eigenvalues (see e.g. [14]). However, in this research we restrict ourselves to real eigenvalues to simplify the analysis in the following sections. An expansion of the results presented here to the general case of complex eigenvalues is very likely possible and may be explored in a future paper.

Definition 5.2 (Invariant Stable and Unstable Sets of a Diffeomorphism)**.** For the map f as defined above, let M_s denote all points $x \in \mathbb{R}^n$ such that

$$\forall x \in M_s \lim_{k\to\infty} f^k(x) = 0.$$

Equally, let M_u denote all points $x \in \mathbb{R}^n$ such that

$$\forall x \in M_u \lim_{k\to\infty} f^{-k}(x) = 0$$

Certainly M_u and M_s are non-empty sets, since $0 \in M_u$ and $0 \in M_s$. Where necessary, we denote by $M_s(f)$ the stable set of the map f, and similarly by $M_u(f)$ the unstable set of the map f.

Theorem 5.3 (Invariant Manifold Theorem for Diffeomorphisms)**.** *The invariant sets M_s and M_u as defined above are N_s and N_u dimensional manifolds respectively. Furthermore M_u at 0 is tangent to the linear expanding eigenspace of $Df|_0$ spanned by $\vec{e}_1, \dots, \vec{e}_{N_u}$ and M_s at 0 is tangent to the linear contracting eigenspace of $Df|_0$ spanned by $\vec{e}_{N_u+1}, \dots, \vec{e}_n$. If $f \in C^r$ then also $M_u, M_s \in C^r$.*

Proof. See, for example, [24, 14]. □

From Definition 5.2 one can infer the following lemma:

Lemma 5.4. *The manifolds of f and f^k for any $k \in \mathbb{N}$ are identical.*

Proof. Let $x \in M_s(f)$. Then $\{x_n\}_{n\in\mathbb{N}} = \{f^n(x)\}_{n\in\mathbb{N}}$ by definition converges to 0. Hence the subsequence $\{x_{kn}\}_{n\in\mathbb{N}} = \{\left(f^k\right)^n(x)\}_{n\in\mathbb{N}}$ also converges to 0. Thus $M_s(f) \subset M_s(f^k)$.

Conversely, let $x \in M_s(f^k)$ and $1 > \varepsilon > 0$. Then since f is continuously differentiable in $U = \overline{B}_1(0)$, it is Lipschitz in U with some Lipschitz constant L. Chose $0 < \delta < \frac{\varepsilon}{L^k}$. By the definition of $M_s(f^k)$ there exists some $N_0 \in \mathbb{N}$ such that for all $N > N_0$ $f^{kN}(x) \in B_\delta(0)$. By the Lipschitz condition, we have the following estimate

$$\left|f(f^{kN}(x))\right| = \left|f(f^{kN}(x)) - f(0)\right| < L\left|f^{kN}(x) - 0\right| < L\delta.$$

By induction we hence find that for $i = 1, \ldots, k-1$

$$\left|f^i(f^{kN}(x))\right| < L^i\delta,$$

since all $L^i\delta < \varepsilon < 1$ and hence $f^i(f^{kN}(x)) \in U$.

But then all $f^i(x) \in B_\varepsilon(0)$ for all $i > N_0k$, and since $\varepsilon > 0$ is arbitrary this shows that $\lim_{n\to\infty} f^n(x) = 0$ and thus $M_s(f^k) \subset M_s(f)$.

Replacing f by f^{-1} in the above argument proves the same for the unstable manifold M_u. □

5.1.2 Invariant Manifolds of autonomous ODEs

Consider now the manifolds of an autonomous dynamical system given by the continuously differentiable vector field $F : \mathbb{R}^n \to \mathbb{R}^n$ through the ordinary differential equation

$$\frac{d}{dt}x = F(x).$$

Furthermore, assume $F(0) = 0$ and the eigenvalues $\lambda_i, i = 1, \ldots, N_u$ and $\mu_j, j = 1, \ldots, N_s$, $N_u + N_s = n$, of $DF|_0$ are real and satisfy

$$\begin{aligned} \lambda_i &> 0 \; \forall i = 1, \ldots, N_u, \\ \mu_j &< 0 \; \forall j = 1, \ldots, N_s. \end{aligned}$$

Without loss of generality, let the eigenvectors corresponding to λ_i be aligned with the x_i-axis and the eigenvectors corresponding to the μ_j be aligned with the x_{N_u+j}-axis. Given the above assumptions, this can always be achieved by a suitable linear coordinate transformation.

Denote by $\Phi_t(x)$ the time t map of the vector field F, i.e. $\Phi_0(x) = x$ and $\frac{d}{dt}\Phi_t(x) = F(\Phi_t(x)) \; \forall t \in \mathbb{R}$, assuming solutions exist for all time.

Note that since F is continuously differentiable by assumption, it is Lipschitz and thus from the uniqueness of solutions of an ODE it follows that

$$\Phi_{t_1}(\Phi_{t_2}(x)) = \Phi_{t_1+t_2}(x).$$

We will also refer to the map $\Phi_t(x)$ for some fixed x as the orbit $x(t)$ of x. For $t \geqslant 0$ we refer to $x(t)$ as the forward orbit of x, and similarly for $t \leqslant 0$ as the backward orbit of x.

Lemma 5.5. *Any map Φ_t, with $t > 0$ has an N_s dimensional stable and N_u dimensional unstable manifold at the origin.*

Proof. The existence of stable and unstable manifolds for any positive real t follows directly from theorem 5.3 and the facts that $\Phi_t(0) = 0$ and that $D\Phi_t(0)$ is diagonal with eigenvalues $\exp(\lambda_i t) > 1$, $i = 1, \ldots, N_u$ and $\exp(\mu_j t) < 1$, $j = 1, \ldots, N_s$.

To see this, consider the integral equation formulation for $\Phi_t(x)$ at the origin where $\Phi_t(0) = 0$. Then

$$\begin{aligned} \Phi_t(x) &= \int_0^t F(\Phi_s(x))\, ds, \\ D\Phi_t(x) &= \int_0^t DF(\Phi_s(x)) \cdot D\Phi_s(x)\, ds, \\ D\Phi_t(0) &= \int_0^t DF(0) \cdot D\Phi_s(x)\, ds. \end{aligned}$$

But $DF(0)$ is a diagonal matrix with the λ_i, $i = 1, \ldots, N_u$ in the first N_u diagonal entries, and the μ_j, $j = 1, \ldots, N_s$ in the last N_s diagonal entries. Thus the solution of this matrix integral equation for the Jacobian matrix $D\Phi_t(0)$ is simply given by

$$D\Phi_t(0) = \exp\left(DF(0) \cdot t\right) = \begin{pmatrix} \exp(\lambda_1 t) & & 0 \\ & \ddots & \\ 0 & & \exp(\mu_{N_s} t) \end{pmatrix}.$$

□

Definition 5.6 (Invariant Stable and Unstable Set of an ODE)**.** For the vector field F as defined above, let M_s denote all points $x \in \mathbb{R}^n$ such that

$$\forall x \in M_s \ \lim_{t \to \infty} \Phi_t(x) = 0.$$

Equally, let M_u denote all points $x \in \mathbb{R}^n$ such that

$$\forall x \in M_u \ \lim_{t \to \infty} \Phi_{-t}(x) = 0.$$

Theorem 5.7 (Invariant Manifold Theorem for ODEs)**.** *The invariant sets M_s and M_u given in Definition 5.6 for the vector field F are N_s and N_u dimensional manifolds respectively. The manifold M_u at* 0 *is tangent to the linear expanding eigenspace of $DF|_0$ and M_s at* 0 *is tangent to the linear contracting eigenspace of $DF|_0$.*

Furthermore, the manifolds M_s and M_u are identical to the manifolds of the time one map Φ_1 of F.

Proof. Clearly, all points of the stable set of F are also in the stable manifold of the time one map Φ_1, i.e. $M_s(F) \subset M_s(\Phi_1)$. This is because for any point $x \in M_s(F)$ such that $\lim_{t \to \infty} \Phi_t(x) = 0$, the subsequence $\{\Phi_N(x)\}_{N \in \mathbb{N}}$ also converges to 0.

Conversely, consider any point $x \in M_s(\Phi_1)$. It suffices to show for every $\varepsilon > 0$ there is some time T such that for $t > T$ $|\Phi_t(x)| < \varepsilon$. So fix some $1 > \varepsilon > 0$.

Since $\Phi_t(x)$ is continuously differentiable in both x and t for $x \in \overline{B}_1(0)$ and $t \in [0, 1]$, there exists a Lipschitz constant L such that

$$\left|\Phi_{t_1}(x_1) - \Phi_{t_2}(x_2)\right| \leqslant L\sqrt{(t_1 - t_2)^2 + |x_1 - x_2|^2}$$

for all $x_1, x_2 \in \overline{B}_1(0)$ and $t_1, t_2 \in [0, 1]$.

Choose some δ such that $0 < \delta < \min(1, \varepsilon/L)$. Then by the definition of the stable manifold $M_s(\Phi_1)$, there is some $N_0 \in \mathbb{N}$ such that $\left|\Phi_N(x)\right| < \delta$ for all integers $N \geqslant N_0$.

This then implies that $|\Phi_t(x)| < \varepsilon$ for all times $t > N_0$. This holds true since such t can be written as $N + \Delta t$, where $N \geqslant N_0$ is an integer and $\Delta t \in [0, 1)$. Letting $x_0 = \Phi_N(x)$, due to the Lipschitz condition

$$\begin{aligned} |\Phi_t(x)| &= \left|\Phi_{N+\Delta t}(x)\right| = \left|\Phi_{\Delta t}(\Phi_N(x))\right| = \left|\Phi_{\Delta t}(x_0)\right| \\ &= \left|\Phi_{\Delta t}(x_0) - 0\right| = \left|\Phi_{\Delta t}(x_0) - \Phi_{\Delta t}(0)\right| \\ &\leqslant L\,|x_0 - 0| = L\,|x_0| < L\delta < \varepsilon \end{aligned}$$

Since ε is arbitrary, this shows that $\lim_{t\to\infty} \Phi_t(x) = 0$, and thus $M_s(\Phi_1) \subset M_s(F)$. Therefore $M_s(F) = M_s(\Phi_1)$.

The same argument holds for the unstable manifold, replacing F by $-F$ and Φ_1 by Φ_{-1}. □

Corollary 5.8. *The invariant manifolds of F at the origin are identical to the invariant manifolds of the time t_0 map Φ_{t_0} of F at the origin for any time $t_0 > 0$.*

Proof. In the proof for theorem 5.7, replace the Lipschitz constant L by $\widetilde{L}$ valid for any times $t_1, t_2 \in [0, t]$. Similarly, chose $N_0 \in \mathbb{N}$ such that $\left|\Phi_{N \cdot t_0}(x)\right| < \delta$ for all $N \geqslant N_0$. Then, any time t can be written as $N \cdot t_0 + \Delta t$ where $N \in \mathbb{N}$ and $\Delta t \in [0, t_0]$. The rest of the proof is identical. □

Corollary 5.9. *The invariant manifolds of any two time t maps Φ_{t_1} and Φ_{t_2} of F at the origin are identical for any $t_1, t_2 > 0$.*

Proof. By transitivity and corollary 5.8, $M_s(\Phi_{t_1}) = M_s(F) = M_s(\Phi_{t_2})$ and the same for M_u. □

Definition 5.10. Let $K \subset \mathbb{R}^n$ such that $0 \in K$, and F as before. We refer to the connected part of the invariant manifolds containing 0 within K as local invariant manifolds W_u^{loc} for the unstable and W_s^{loc} for the stable manifold.

5.1.3 Notation

For ease of notation we will introduce the following equivalence relation for two sufficiently differentiable functions f and g. We say $f =_n g$ if each of the first n derivatives of f and g at the origin are equal.

Furthermore, we say α is a n-multi-index if it is a set of non-negative integers $\alpha_1 \dots \alpha_n$. If n is clear from the context, we will simply speak of a multi-index α.

We define

$$|\alpha| = \sum_{i=1}^{n} \alpha_i,$$

and given a vector $\vec{x} \in \mathbb{R}^n$ and a n-multi-index α, we use the short hand notation

$$\vec{x}^{\alpha} = \prod_{i=1}^{n} x_i^{\alpha_i}.$$

With that notation, it is possible to uniquely identify each coefficient a_α of a n-dimensional polynomial P by a n-multi-index α, and write it as

$$P(\vec{x}) = \sum_{\alpha} a_\alpha \vec{x}^{\alpha},$$

where $\vec{x} \in \mathbb{R}^n$. Furthermore, we use the notation

$$\sum_{|\alpha|=k} a_\alpha \vec{x}^\alpha$$

to denote a sum over all possible multi-indices α such that $|\alpha| = k$, i.e. all terms of exact order k in P.

5.2 High Order Invariant Manifold Approximation in Discrete Systems

In this section, we will develop a constructive algorithm to obtain very accurate high order polynomial expansions of the invariant manifolds of a hyperbolic point in arbitrary dimension. In case resonances between the eigenvalues are encountered, several methods are proposed to overcome those resonances and continue the construction. This method is loosely based on the parametrization method introduced in [14].

Let $\phi : \mathbb{R}^m \to \mathbb{R}^m$ be a C^r-diffeomorphism with the following properties:

1. There is a fixed point at the origin

$$\phi(0) = 0$$

2. The fixed point is hyperbolic and the map is diagonal with real eigenvalues:

$$D\phi(0) = \begin{pmatrix} \Lambda & 0 \\ 0 & M \end{pmatrix}$$

where

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_k \end{pmatrix}$$

with $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_k)^T$, $|\lambda_i| > 1, \ i = 1 \ldots k$ and

$$M = \begin{pmatrix} \mu_{k+1} & 0 & 0 & 0 \\ 0 & \mu_{k+2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \mu_m \end{pmatrix}$$

with $\vec{\mu} = \left(\mu_{k+1}, \mu_{k+2}, \ldots, \mu_m\right)^T$, $0 < |\mu_i| < 1, \ i = k+1 \ldots m$.

Remark 5.11. Any diffeomorphism Ψ with a fixed point x and real, non-zero and non-unity eigenvalues of the derivative at x can be brought into this form by applying an affine coordinate transformation.

By theorem 5.3 there exists a k dimensional unstable manifold and a $m-k$ dimensional stable manifold of ϕ at the origin. The following describes an algorithm to obtain a polynomial approximation of this local manifold around the origin of any order up to r. Without loss of generality, only the unstable manifold construction will be considered here, the construction for the stable manifold is performed analogously.

Let $f : \mathbb{R}^m \to \mathbb{R}^m$ be a polynomial such that $f =_r \phi$, i.e. f is the Taylor expansion up to order r of the diffeomorphism ϕ. In the construction of the polynomial manifold approximation f will be used instead of ϕ. Note that f can then be written as

$$f(\vec{x}) = \begin{pmatrix} \Lambda & 0 \\ 0 & M \end{pmatrix} \cdot \vec{x} + \widetilde{f}(\vec{x}) \tag{5.1}$$

where $\widetilde{f}$ is a polynomial without constant or linear part.

5.2.1 Parameterization

Let

$$\gamma^p(\vec{t}) = \begin{pmatrix} \gamma_1^p(\vec{t}) \\ \gamma_2^p(\vec{t}) \\ \vdots \\ \gamma_m^p(\vec{t}) \end{pmatrix} \text{ where } \vec{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_k \end{pmatrix} \in \mathbb{R}^k$$

be the polynomial approximation to the manifold of order p. Since it is an approximation to the invariant manifold, we then demand that it satisfies the condition

$$f(\gamma^p(\vec{t})) =_p \gamma^p(P^p(\vec{t})) \tag{5.2}$$

where

$$P^p(\vec{t}) = \begin{pmatrix} P_1^p(\vec{t}) \\ P_2^p(\vec{t}) \\ \vdots \\ P_k^p(\vec{t}) \end{pmatrix}, \ \vec{t} \in \mathbb{R}^k$$

is a polynomial of order p.

It will become apparent during the following construction of the unstable manifold that the introduction of P^p is required to avoid resonances that can otherwise occur in dimensions higher than 2.

Remark 5.12. The condition on the manifold parametrization given in equation 5.2 is not sufficient to produce a unique parametrization. Clearly, if $\gamma^p(\vec{t})$ and $P^p(\vec{t})$ satisfy equation 5.2, so do $\gamma^p(\theta(\vec{t}))$ and $P^p(\theta(\vec{t}))$, where $\theta : \mathbb{R}^k \to \mathbb{R}^k$ is any C^p diffeomorphism.

5.2.2 Construction

We now proceed to give an inductive construction algorithm for such γ^p and P^p for any $p \leqslant r$.

First, consider the following polynomials for $p = 1$:

$$\gamma^1(\vec{t}) = \begin{pmatrix} \gamma^1_1(\vec{t}) \\ \gamma^1_2(\vec{t}) \\ \vdots \\ \gamma^1_k(\vec{t}) \\ \gamma^1_{k+1}(\vec{t}) \\ \vdots \\ \gamma^1_m(\vec{t}) \end{pmatrix} = \begin{pmatrix} \eta_1 t_1 \\ \eta_2 t_2 \\ \vdots \\ \eta_k t_k \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

that is a simple scaling in the first k variables and zeros for the rest, and

$$P^1(\vec{t}) = \Lambda \cdot \vec{t} = \begin{pmatrix} \lambda_1 t_1 \\ \lambda_2 t_2 \\ \vdots \\ \lambda_k t_k \end{pmatrix}.$$

The values $\eta_i > 0$, $i = 1, \ldots, k$ are free parameters in the construction, according to Remark 5.12. Their value determines the size of the derivative of the manifold approximation at the origin. They can be used to control the convergence of the higher order terms of the resulting polynomial, the smaller these values are chosen, the smaller the higher order terms of γ^p and P^p will be.

Then it is apparent that condition (5.2) holds since

$$f\left(\gamma^1\left(\vec{t}\right)\right) = \begin{pmatrix} \Lambda & 0 \\ 0 & M \end{pmatrix} \cdot \begin{pmatrix} \eta_1 t_1 \\ \eta_2 t_2 \\ \vdots \\ \eta_k t_k \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \widetilde{f} \begin{pmatrix} \eta_1 t_1 \\ \eta_2 t_2 \\ \vdots \\ \eta_k t_k \\ 0 \\ \vdots \\ 0 \end{pmatrix} =_1 \begin{pmatrix} \lambda_1 \eta_1 t_1 \\ \lambda_2 \eta_2 t_2 \\ \vdots \\ \lambda_k \eta_k t_k \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \gamma^1\left(P^1\left(\vec{t}\right)\right)$$

because $\widetilde{f}$ only contains terms of order 2 or higher and hence $\widetilde{f} =_1 0$.

5.2.3 Induction step

Assume γ^p, P^p satisfy condition (5.2) up to order p.

Let $a_{n,\alpha}$, $b_{n,\alpha}$ be such that

$$\gamma_n^{p+1}\left(\vec{t}\right) = \gamma_n^p\left(\vec{t}\right) + \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{t}^{\alpha},$$

$$P_n^{p+1}\left(\vec{t}\right) = P_n^p\left(\vec{t}\right) + \sum_{|\alpha|=p+1} b_{n,\alpha}\vec{t}^{\alpha},$$

i.e. they are the $p+1$ order coefficients of γ_n^{p+1} and P_n^{p+1} respectively.

It remains to identify values for $a_{n,\alpha}$, $b_{n,\alpha}$ such that condition (5.2) holds up to order $p+1$.

First note that for $1 \leqslant n \leqslant k$

$$f_n\left(\gamma^{p+1}\left(\vec{t}\right)\right) =_{p+1} f_n\left(\gamma^p\left(\vec{t}\right)\right) + \lambda_n \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{t}^{\alpha}, \tag{5.3}$$

and equivalently for $k < n \leqslant m$

$$f_n\left(\gamma^{p+1}\left(\vec{t}\right)\right) =_{p+1} f_n\left(\gamma^p\left(\vec{t}\right)\right) + \mu_n \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{t}^{\alpha}. \tag{5.4}$$

This follows from equation (5.1) and the fact that $\widetilde{f}\left(\gamma^{p+1}\left(\vec{t}\right)\right) =_{p+1} \widetilde{f}\left(\gamma^p\left(\vec{t}\right)\right)$ since $\widetilde{f}$ has no constant or linear part, and γ^p and γ^{p+1} have no constant parts respectively.

Secondly, by a similar argument, for $1 \leqslant n \leqslant k$

$$\gamma_n^{p+1}\left(P^{p+1}\left(\vec{t}\right)\right) =_{p+1} \gamma_n^p\left(P^p\left(\vec{t}\right)\right) + \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{\lambda}^\alpha\vec{t}^\alpha + \eta_n \sum_{|\alpha|=p+1} b_{n,\alpha}\vec{t}^\alpha, \tag{5.5}$$

and for $k < n \leqslant m$

$$\gamma_n^{p+1}\left(P^{p+1}\left(\vec{t}\right)\right) =_{p+1} \gamma_n^p\left(P^p\left(\vec{t}\right)\right) + \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{\lambda}^\alpha\vec{t}^\alpha. \tag{5.6}$$

This holds since

$$\begin{aligned} \gamma_n^{p+1}\left(P^{p+1}\left(\vec{t}\right)\right) &= \gamma_n^p\left(P^{p+1}\left(\vec{t}\right)\right) + \sum_{|\alpha|=p+1} a_{n,\alpha}P^{p+1}\left(\vec{t}\right)^\alpha \\ &=_{p+1} \gamma_n^p\left(P^{p+1}\left(\vec{t}\right)\right) + \sum_{|\alpha|=p+1} a_{n,\alpha}P^1\left(\vec{t}\right)^\alpha \\ &= \gamma_n^p\left(P^{p+1}\left(\vec{t}\right)\right) + \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{\lambda}^\alpha\vec{t}^\alpha \end{aligned}$$

because P^{p+1} has no constant part, and

$$\begin{aligned} \gamma_n^p\left(P^{p+1}\left(\vec{t}\right)\right) &= \gamma_n^1\left(P^{p+1}\left(\vec{t}\right)\right) + \left(\gamma_n^p - \gamma_n^1\right)\left(P^{p+1}\left(\vec{t}\right)\right) \\ &=_{p+1} \gamma_n^1\left(P^{p+1}\left(\vec{t}\right)\right) + \left(\gamma_n^p - \gamma_n^1\right)\left(P^p\left(\vec{t}\right)\right) \\ &=_{p+1} \begin{cases} \gamma_n^p\left(P^p\left(\vec{t}\right)\right) + \eta_n \sum_{|\alpha|=p+1} b_{n,\alpha}\vec{t}^\alpha & 1 \leqslant n \leqslant k \\ \gamma_n^p\left(P^p\left(\vec{t}\right)\right) & k < n \leqslant m \end{cases} \end{aligned}$$

as $\left(\gamma_n^p - \gamma_n^1\right)$ has no constant or linear part and γ_n^1 is linear.

Inserting equations (5.3) through (5.6) into condition (5.2) then reads component-wise:

$$f_n\left(\gamma^p\left(\vec{t}\right)\right) + \lambda_n \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{t}^{\alpha} \quad =_{p+1} \tag{5.7}$$
$$\gamma_n^p\left(P^p\left(\vec{t}\right)\right) + \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{\lambda}^{\alpha}\vec{t}^{\alpha} + \lambda_n\eta_n \sum_{|\alpha|=p+1} b_{n,\alpha}\vec{t}^{\alpha}$$

for $1 \leqslant n \leqslant k$, and

$$f_n\left(\gamma^p\left(\vec{t}\right)\right) + \mu_n \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{t}^{\alpha} =_{p+1} \gamma_n^p\left(P^p\left(\vec{t}\right)\right) + \sum_{|\alpha|=p+1} a_{n,\alpha}\vec{\lambda}^{\alpha}\vec{t}^{\alpha} \tag{5.8}$$

for $k < n \leqslant m$.

By the induction assumption all terms up to order p in (5.7) and (5.8) cancel. Thus, in order to satisfy condition (5.2), $a_{n,\alpha}$ and $b_{n,\alpha}$ must be chosen such that all terms of order $p+1$ in in (5.7) and (5.8) cancel.

Let $c_{n,\alpha}$ be the coefficient corresponding to the power $\vec{t}^{\alpha}$, $|\alpha| = p+1$, in the expression

$$f_n\left(\gamma^p\left(\vec{t}\right)\right) - \gamma_n^p\left(P^p\left(\vec{t}\right)\right).$$

Those are well defined as this expression only involves already known quantities.

Then (5.7) and (5.8) can be rewritten into

$$\sum_{|\alpha|=p+1} c_{n,\alpha}\vec{t}^{\alpha} = \sum_{|\alpha|=p+1} \left(\left(\vec{\lambda}^{\alpha} - \lambda_n\right) a_{n,\alpha} + \eta_n b_{n,\alpha}\right)\vec{t}^{\alpha} \tag{5.9}$$

for $1 \leqslant n \leqslant k$, and

$$\sum_{|\alpha|=p+1} c_{n,\alpha} \vec{t}^{\alpha} = \sum_{|\alpha|=p+1} \left(\vec{\lambda}^{\alpha} - \mu_n\right) a_{n,\alpha} \vec{t}^{\alpha} \tag{5.10}$$

for $k < n \leqslant m$ respectively.

Performing term-by-term matching on both sides, values for $a_{n,\alpha}$ and $b_{n,\alpha}$ can now be determined. Consider first the case $k < n \leqslant m$. From (5.10) it follows directly that

$$a_{n,\alpha} = \frac{c_{n,\alpha}}{\vec{\lambda}^{\alpha} - \mu_n}.$$

Note that since all $|\lambda_i| > 1$ we have $|\vec{\lambda}^{\alpha}| > 1$ and thus $\vec{\lambda}^{\alpha} - \mu_n \neq 0$ as $|\mu_n| < 1$. Thus the value of $a_{n,\alpha}$ is well defined.

Next, consider the case $1 \leqslant n \leqslant k$. Again performing term-by-term matching in (5.9) the following condition is derived

$$\left(\vec{\lambda}^{\alpha} - \lambda_n\right) a_{n,\alpha} + \eta_n b_{n,\alpha} = c_{n,\alpha}. \tag{5.11}$$

Clearly, the choice of $a_{n,\alpha}$ and $b_{n,\alpha}$ is not unique in this case. In the following, two of the possible choices are discussed in more detail.

5.2.3.1 Gamma-Convention

Obviously, unlike in the previous case, now it is not possible to chose $a_{n,\alpha}$ such that condition (5.11) is aways satisfied. This is contingent upon the resonance condition

$$\vec{\lambda}^{\alpha} = \lambda_n. \tag{5.12}$$

Consider the case where $\vec{\lambda}^{\alpha} - \lambda_n \neq 0$, i.e. there is no resonance for α and n. Then let

$$b_{n,\alpha} = 0$$

and

$$a_{n,\alpha} = \frac{c_{n,\alpha}}{\vec{\lambda}^{\alpha} - \lambda_n}.$$

On the other hand, if $\vec{\lambda}^{\alpha} - \lambda_n = 0$, i.e. there is a resonance, let

$$b_{n,\alpha} = \frac{c_{n,\alpha}}{\eta_n}$$

and

$$a_{n,\alpha} = 0.$$

In both cases the values are well defined.

Remark 5.13. It follows directly from this construction that there exists a $\widehat{p}$ such that $P^{\widehat{p}} = P^{p}$ for all $p > \widehat{p}$. That is, resonances can only occur up to finite order.

This is because for $p = |\alpha| > \widehat{p}$ we have

$$|\vec{\lambda}^{\alpha}| \geqslant \min_i(|\lambda_i|)^p > \min_i(|\lambda_i|)^{\widehat{p}} > \max_i(|\lambda_i|)$$

for large enough $\widehat{p}$ and thus $\vec{\lambda}^{\alpha} - \lambda_n \neq 0$ for all $1 \leqslant n \leqslant k$, $|\alpha| > \widehat{p}$.

Remark 5.14. Furthermore, in the two dimensional case, this resonance condition can never occur as there is only one expanding and one contracting eigenvalue. Yet even in higher dimensions, since there are only finitely many resonance conditions, in typical cases these resonances do not occur. Consequently, with this construction P^p is kept “very close” to P^1.

5.2.3.2 P-Convention

An alternative choice is to follow the opposite path than the Gamma-Convention: Instead of keeping $b_{n,\alpha}$ zero whenever possible, one can instead always make the choice

$$b_{n,\alpha} = \frac{c_{n,\alpha}}{\eta_n}$$

and

$$a_{n,\alpha} = 0.$$

By this construction, the first k components of $\gamma\left(\vec{t}\right)$ always stay the identity, while all the information about manifold propagation is contained in $P\left(\vec{t}\right)$.

Remark 5.15. The resulting manifold parametrization is identical to the one obtained by the graph transform method. In particular, for $n = 1, \ldots, k$

$$P_n^p(\vec{t}) =_p f_n(\vec{t}).$$

However, this construction not only provides a polynomial approximation of the graph, as the graph transform does, but also provides the polynomial expression for propagation on that manifold in the P_n^p, $n = k+1, \ldots, m$.

5.2.4 Stable manifold

To construct the stable manifold approximation, one possibility is to apply the above construction to the inverse map ϕ^{-1}. In practice, however, it is usually easier to perform the same construction just described while exchanging the roles of μ and λ. The required changes to the algorithm are straight forward, and will not be discussed here.

5.3 Invariant Manifolds of Autonomous ODEs

We will now describe a method to construct and verify the invariant manifolds of an autonomous, i.e. not explicitly time dependent, ordinary differential equation.

In the following, Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be a C^r vector field satisfying the conditions described in Section 5.1.2. That is, let $F(0) = 0$, and the eigenvalues $\lambda_i, i = 1, \ldots, N_u$ and $\mu_j, j = 1, \ldots, N_s$ of $DF|_0$ be real and satisfy

$$\begin{aligned} \lambda_i &> 0 \ \forall i = 1, \ldots, N_u, \\ \mu_j &< 0 \ \forall j = 1, \ldots, N_s. \end{aligned}$$

with the eigenvectors corresponding to λ_i be aligned with the x_i-axis and the eigenvectors corresponding to the μ_j be aligned with the x_{N_u+j}-axis.

We then are interested in first generating a polynomial approximation of the stable and unstable manifolds of the critical point at the origin. Given that approximation, we then add a small thickening to it and verify that the result represents a uniformly verified C^0 enclosure of the manifold over a certain domain.

5.3.1 Manifold Generation

Consider a Taylor expansion to order r of the vector field F around the origin of the form

$$\overline{F}(x) = \begin{pmatrix} \Lambda & 0 \\ 0 & M \end{pmatrix} \cdot x + \widetilde{F}(x)$$

where Λ is a diagonal matrix with the eigenvalues λ_i in the diagonal and M is a diagonal matrix with the eigenvalues μ_j in the diagonal entries. Furthermore, $\widetilde{F}$ only contains terms of order 2 or higher up to some order r.

We now proceed to compute a polynomial approximation of the manifold for this approximate vector field $\overline{F}$. Due to corollary 5.8 and theorem 5.7, the manifold of the vector field $\overline{F}$ is identical to the manifold for any of the discrete time t maps Φ_t induced by $\overline{F}$ for any value $t > 0$. It is thus sufficient to compute an approximation of the manifolds of some time t_0 map Φ_{t_0} with $t_0 > 0$ for $\overline{F}$ and then apply the construction algorithm for discrete maps described as described in Section 5.2 to Φ_{t_0}.

In order to compute the Φ_{t_0} map, a Taylor expansion of the flow of $\overline{F}$ around the origin is computed. This can be done particularly nicely using a super-convergent fixed point iteration in differential algebra arithmetic[8]. Without going into the detail of this method here, suffice it to say here that it results in a Taylor expansion of $\Phi_t(x)$ in both t and x. To obtain the map Φ_{t_0}, all that is required is to evaluate this Taylor expansion for the value $t = t_0$, resulting in a polynomial only in x representing $\Phi_{t_0}(x)$.

Note that the choice of the value of $t_0 > 0$ is in principle arbitrary. However, one has to keep in mind that the Taylor expansion of the flow is only performed to a certain finite order. By inserting a specific value for the time t, care must be taken to ensure that the expansion for the given time t_0 still converges well enough. The larger the error in the approximation of the time t_0 map Φ_{t_0} of the system, the worse the final approximation of the manifold will be.

On the other hand, the eigenvalues of the derivative of Φ_{t_0} at the origin are given by expressions of the form $\exp(\lambda_i t_0)$ and $\exp(\mu_j t_0)$. For the construction to succeed numerically without resulting in divisions by values very close to zero, these eigenvalues of Φ_{t_0} must be separated sufficiently far away from 1. Thus t_0 must not be chosen too small, in order to still satisfy this requirement.

While somewhat problem dependent, we found that time steps of size 10^{-1} or 10^{-2} in our cases seem to strike a good balance between the two requirements, yielding very accurate enclosures while maintaining reasonably large eigenvalues.

This construction yields a polynomial approximation $\gamma_u : \mathbb{R}^{N_u} \to \mathbb{R}^n$ of the unstable manifold, and similarly $\gamma_s : \mathbb{R}^{N_s} \to \mathbb{R}^n$ of the stable manifold. Both by construction satisfy the properties

$$\gamma_u(0) = \gamma_s(0) = 0$$

and

$$D\gamma_u(0) = \begin{pmatrix} d_{u,1} & & 0 \\ & \ddots & \\ 0 & & d_{u,N_u} \\ & & \\ & 0 & \\ & & \end{pmatrix} \text{ and } D\gamma_s(0) = \begin{pmatrix} & & \\ & 0 & \\ & & \\ d_{s,1} & & 0 \\ & \ddots & \\ 0 & & d_{s,N_s} \end{pmatrix}$$

where $d_{s,i}, d_{u,j} > 0$ for $i = 1, \ldots, N_s$ and $j = 1, \ldots, N_u$.

5.3.2 Manifold Verification

In this Section, we describe how the unstable manifold approximation generated by the algorithm described in the previous section is outfitted with a small remainder bound and verified to form a rigorous enclosure of the local unstable manifold. The verification for the stable manifold is exactly analogous, using the vector field $-F$ instead of F.

The verification presented here will cover the two cases of a one dimensional and a two dimensional manifold in three dimensions. The method described here does generalize to arbitrary dimensions, however the notation becomes quite cumbersome.

Throughout the following, we denote by $\vec{e}_x, \vec{e}_y, \vec{e}_z$ the unit vector along the corresponding axis.

5.3.2.1 Outline of the proof

The general concept of constructing a manifold enclosure is to perform the following steps:

1. Each point on the polynomial manifold approximation, over some bounded domain, is outfit with a uniform thickening in each stable direction to yield a candidate for the manifold enclosure. We want to show that for each point on the polynomial approximation there is at least one point of the manifold within its thickening.

2. By examining the vector field on the boundary of this thickening, it is shown that the local manifold cannot leave the enclosure through the sides created by this thickening, but only through the ends of the polynomial manifold approximation.

3. By bounding the radial component of the vector field over a suitably chosen set, it is shown that any point on the local unstable manifold away from the origin must leave the manifold enclosure.

4. From this we will conclude that for any point on the initial polynomial approximation, there is a point of the manifold that lies within the thickening around this point.

5.3.2.2 One Dimensional Manifold in Three Dimensions

Let $\gamma_u(x)$ be the polynomial unstable manifold approximation as computed in the previous section. For ease of notation, we will drop the index u and refer to γ_u simply as γ from now on. Additionally, we will assume that the polynomial approximation is given as the graph of a function over the x-axis, i.e.

$$\gamma : \begin{cases} \mathbb{R} \to & \mathbb{R}^3 \\ \gamma(x) = & \begin{pmatrix} x \\ y(x) \\ z(x) \end{pmatrix} \end{cases} \tag{5.13}$$

where $y, z : \mathbb{R} \to \mathbb{R}$ are polynomials without constant or linear part.

Remark 5.16. This requirement can always be satisfied since the manifold is tangent to the x-axis at the origin. Furthermore, such polynomials are easily obtained using the "P-convention" in the construction as described in Section 5.2.3.2.

If the polynomial is constructed with some scaling factor η_1 in the x-coordinate (see section 5.2), without loss of generality one can consider the coordinate transform $T(x, y, z) = (x/\eta_1, y, z)$ and the new vector field $F'(x, y, z) = T \circ F \circ T^{-1}$ as well as $\gamma' = T \circ \gamma$.

Definition 5.17. For some $\varepsilon > 0$ fixed, let $\Gamma : [-1,1] \times [-1,1]^2 \to \mathbb{R}^3$ be defined as

$$\begin{aligned} \Gamma(s,t_1,t_2) &= \gamma(s) + \varepsilon \cdot \big(t_1 \cdot \vec{e}_y + t_2 \cdot \vec{e}_z\big) & (5.14) \\ &= \begin{pmatrix} s \\ y(s) + \varepsilon t_1 \\ z(s) + \varepsilon t_2 \end{pmatrix} & (5.15) \end{aligned}$$

That is, Γ is a uniform thickening of the unstable manifold approximation along the two stable directions.

In the following, we will refer to Γ as both the function $\Gamma(s,t_1,t_2)$ as well as the set Γ representing the range of $\Gamma(s,t_1,t_2)$ over its domain. The exact meaning will be made clear by the context.

Clearly Γ contains the local unstable manifold W_u^{loc} as it contains a neighborhood of the origin since

$$D\Gamma(0,0,0) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{pmatrix} \text{ and } \Gamma(0,0,0) = 0.$$

We call the following objects the *sides* of Γ:

$$\begin{aligned} S_y^{\pm}\Gamma(s,t) &= \Gamma(s,\pm 1,t) \\ S_z^{\pm}\Gamma(s,t) &= \Gamma(s,t,\pm 1) \end{aligned}$$

where again $s,t \in [-1,1]$.

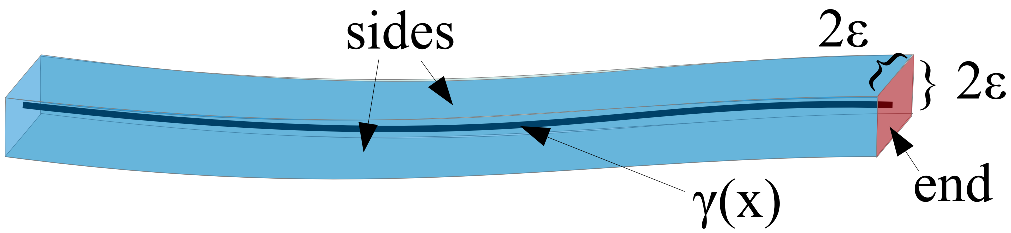


Figure 5.1: The enclosure Γ for a curve $\gamma(x)$ in $\mathbb{R}^3$ with its sides (blue) and ends (red) highlighted.

All sides of Γ are collectively referred to as $S\Gamma$ defined as

$$S\Gamma = S_y^+\Gamma \cup S_y^-\Gamma \cup S_z^+\Gamma \cup S_z^-\Gamma.$$

Similarly, the *ends* of Γ are given by

$$E_x^{\pm}\Gamma(t_1, t_2) = \Gamma(\pm 1, t_1, t_2),$$

where again $t_1, t_2 \in [-1, 1]$.

Both ends of Γ are collectively referred to as $E\Gamma$ defined as

$$E\Gamma = E_x^+\Gamma \cup E_x^-\Gamma.$$

Figure 5.1 illustrates the concepts of Γ and its sides and ends.

Lemma 5.18. *If the negative vector field F on $S\Gamma$ always points outwards of Γ, W_u^{loc} cannot not intersect $S\Gamma$.*

Furthermore, the negative vector field F on $S_y^{\pm}\Gamma$ points outwards Γ if for all $s, t \in [-1, 1]$

$$\pm\left(-\dot{y}(s) \cdot F_x\left(S_y^{\pm}\Gamma(s,t)\right) + F_y\left(S_y^{\pm}\Gamma(s,t)\right)\right) \;<\; 0$$

and similarly for $S_z^{\pm}\Gamma$

$$\pm\left(-\dot{z}(s) \cdot F_x\left(S_z^{\pm}\Gamma(s,t)\right) + F_z\left(S_z^{\pm}\Gamma(s,t)\right)\right) \;<\; 0.$$

Proof. If the negative vector field F on $S\Gamma$ always points outwards of Γ, the backward orbit $\vec{x}(t)$ of any point $\vec{x} \in S\Gamma$ leaves Γ. Hence W_u^{loc}, for which the entire backward orbit is contained in Γ, cannot intersect $S\Gamma$.

Consider now the case of $S_y^{+}\Gamma(s,t)$. The tangent vectors to this surface are given by

$$\begin{pmatrix} 1 \\ \dot{y}(s) \\ \dot{z}(s) \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \varepsilon \end{pmatrix}.$$

Thus the outward normal vector to the surface is given by $N(s,t) = \begin{pmatrix} -\dot{y}(s) \\ 1 \\ 0 \end{pmatrix}$. This vector indeed points outward of Γ because it points in the positive y direction, and by definition of Γ $S_y^{+}\Gamma(s,t)$ is the boundary of Γ with the largest y value for fixed s, t.

Thus the vector $-F\left(S_y^{+}\Gamma(s,t)\right)$ points outward if $N(s,t) \cdot F\left(S_y^{+}\Gamma(s,t)\right) < 0$ which is equivalent to

$$-\dot{y}(s) \cdot F_x\left(S_y^{+}\Gamma(s,t)\right) + F_y\left(S_y^{+}\Gamma(s,t)\right) < 0.$$

The same analysis performed for the other three sides yields the remaining conditions. □

We now introduce another set C as follows:

Definition 5.19. For some $\vartheta > 0$ fixed, let $C : [-1,1] \times [-1,1]^2 \to \mathbb{R}^3$ be defined as

$$C(s, t_1, t_2) = s \cdot \begin{pmatrix} 1 \\ \vartheta t_1 \\ \vartheta t_2 \end{pmatrix}.$$

That is, C is a thin rectangular cone around the x-axis.

In the following, we will refer to C as both the function $C(s, t_1, t_2)$ as well as the set C representing the range of $C(s, t_1, t_2)$ over its domain. The exact meaning will be made clear by the context.

We call the following objects the *sides* of C:

$$\begin{aligned} S_y^{\pm} C(s,t) &= C(s, \pm 1, t) \\ S_z^{\pm} C(s,t) &= C(s, t, \pm 1) \end{aligned}$$

where again $s, t \in [-1, 1]$.

All sides of C are collectively referred to as SC defined as

$$SC = S_y^{+} C \cup S_y^{-} C \cup S_z^{+} C \cup S_z^{-} C.$$

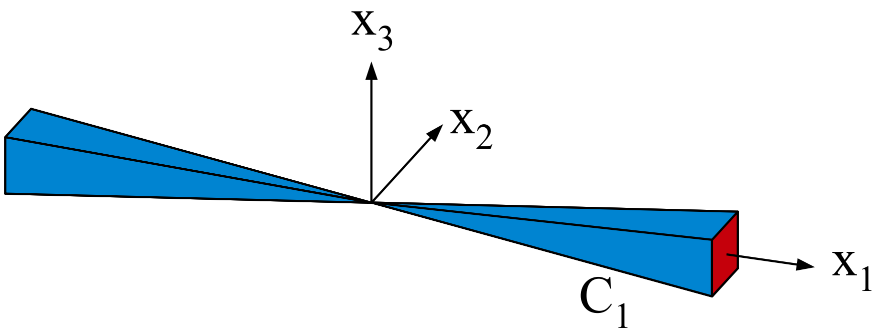


Figure 5.2: The set C with its ends EC (red) and sides SC (blue) highlighted.

Similarly, the *ends* of C are given by

$$E_x^{\pm}C(t_1, t_2) = C(\pm 1, t_1, t_2),$$

where again $t_1, t_2 \in [-1, 1]$.

Both ends of C are collectively referred to as EC defined as

$$EC = E_x^{+}C \cup E_x^{-}C.$$

To illustrate these concepts, figure 5.2 shows an example of such a set C.

Remark 5.20. Clearly C contains some open neighborhood of the local unstable manifold around the origin because C contains the origin and the manifold at the origin is tangent to the x-axis, while all sides of C have a non-zero slope in the x direction.

Lemma 5.21. *If the negative vector field F on $SC\backslash\{0\}$ always points outwards of C, W_u^{loc} cannot not intersect $SC\backslash\{0\}$.*

Furthermore, the negative vector field F on $S_y^{\pm}C\backslash\{0\}$ points outwards C if for all $s \in$

$[-1,1]\backslash\{0\}$ *and* $t \in [-1,1]$

$$\pm\frac{1}{s}\left(-\vartheta \cdot F_x\left(S_y^{\pm}C(s,t)\right) + F_y\left(S_y^{\pm}C(s,t)\right)\right) \;<\; 0$$

and similarly for $S_z^{\pm}C\backslash\{0\}$

$$\pm\frac{1}{s}\left(-\vartheta \cdot F_x\left(S_z^{\pm}C(s,t)\right) + F_y\left(S_z^{\pm}C(s,t)\right)\right) \;<\; 0.$$

Proof. If the negative vector field F on $SC\backslash\{0\}$ always points outwards of C, the backward orbit $\vec{x}(t)$ of any point $\vec{x} \in SC\backslash\{0\}$ leaves C. Hence W_u^{loc}, for which the entire backward orbit is contained in C, cannot intersect SC.

Consider now the case of $S_y^{+}C(s,t)$. The tangent vectors to this surface for $s \in [-1,1]\backslash\{0\}$ and $t \in [-1,1]$ are given by

$$\begin{pmatrix} 1 \\ \vartheta \\ \vartheta t \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \vartheta s \end{pmatrix}.$$

Thus the outward normal vector to the surface is given by $N = \begin{pmatrix} -\vartheta \\ 1 \\ 0 \end{pmatrix}$ for $s > 0$. This vector indeed points outward of C because it points in the positive y direction, and by definition of C $S_y^{+}C(s,t)$ is the boundary of C with the largest y value for fixed $s > 0$, t. Similarly, for $s < 0$ $S_y^{+}C(s,t)$ switches sides, and the outward normal vector here is given by $N = -\begin{pmatrix} -\vartheta \\ 1 \\ 0 \end{pmatrix}$.

Thus the vector $-F\left(S_y^+ C(s,t)\right)$ points outward if $N(s,t) \cdot F\left(S_y^+ C(s,t)\right) < 0$ which is equivalent to

$$\begin{cases} -\vartheta \cdot F_x\left(S_y^+ C(s,t)\right) + F_y\left(S_y^+ C(s,t)\right) < 0 & \text{for } s > 0, \\ -\vartheta \cdot F_x\left(S_y^+ C(s,t)\right) + F_y\left(S_y^+ C(s,t)\right) > 0 & \text{for } s < 0. \end{cases}$$

Since $s \neq 0$, we can divide by s and unify this condition to read

$$\frac{1}{s}\left(-\vartheta \cdot F_x\left(S_y^+ C(s,t)\right) + F_y\left(S_y^+ C(s,t)\right)\right) \quad < \quad 0.$$

This expression will later turn out to be easier to treat numerically, as well.

The same analysis performed for the other three sides yields the remaining conditions. □

Lemma 5.22. *If the following condition holds, the forward orbit of every point $\vec{x} \in C\backslash\{0\}$ must leave C within finite time.*

$$F_x(\vec{x}) \cdot x > 0 \; \forall \vec{x} = (x_x, x_y, x_z) \in C\backslash\{0\} \tag{5.16}$$

Proof. Let $\vec{x} = (x_x, x_y, x_z) \in C\backslash\{0\}$. Denote by $\vec{x}(t)$ the forward orbit in the vector field F

such that $\vec{x}(0) = \vec{x}$. Then consider the expression

$$\begin{aligned}\frac{d}{dt}\left|x_x(t)\right|^2 &= \frac{d}{dt}x_x(t)^2 \\ &= \dot{x}_x(t) \cdot x_x(t) \\ &= F_x(\vec{x}(t)) \cdot x_x(t).\end{aligned}$$

Thus while $\vec{x}(t) \in C$, $|x_x(t)|^2$ is monotonously increasing. But since F is continuous and by virtue of condition 5.16, the expression $F_x(\vec{x}) \cdot x_x$ assumes a strictly positive minimum on every set $C \backslash B_\delta(0)$ for any $\delta > 0$. Since $\vec{x} \neq 0$, we have that $\vec{x} \in C \backslash B_\delta(0)$ for some $\delta > 0$.

Thus $\frac{d}{dt}\left|x_x(t)\right|^2$ is bounded from below by some positive constant $\kappa > 0$. Hence $|x_x(t)|^2 > \kappa t$ for any t such that $\forall \tau \in [0, t]$ $\vec{x}(\tau) \in C$. Thus the orbit must leave C at some finite time, as in C by construction $|x_x| \leqslant 1$. □

Remark 5.23. For this argument to work, it is necessary to consider some set like C which includes the origin, but does not include a full neighborhood of it. This is because in any neighborhood B of the origin, this condition can never be satisfied, as B would include the local stable manifold, which by definition contains points that stay within B for all forward time. Thus a set has to be chosen that does not include any points of the stable manifold.

This automatically precludes testing for these conditions in Γ and requires the introduction of C instead.

Theorem 5.24 (Manifold Enclosure Theorem (1D))**.** *If there exist* $\varepsilon, \vartheta > 0$ *such that the conditions of the above lemmata 5.18 through 5.22 are satisfied, then also* $\forall s \in [-1, 1]$ $\exists t_1, t_2 \in [-1, 1]$ *such that* $\Gamma(s, t_1, t_2) \in W_u^{loc}$ *is satisfied. We call* Γ *a* manifold enclosure.

Proof. Claim: There exists some $\delta > 0$ such that

1. the local unstable manifold can be written as a graph $(x, \hat{y}(x), \hat{z}(x))$ for $x \in [-\delta, \delta]$,
2. the entire graph is contained in C,
3. the entire graph is contained in Γ.

This is true because

1. of the Hadamard-Perron theorem,
2. the manifold is tangent to the x-axis at zero while the boundaries of C form a finite angle with the x-axis,
3. Γ contains an open neighborhood of the origin.

Then consider the end point $\vec{x} = (\delta, \hat{y}(\delta), \hat{z}(\delta))$ of this graph. By lemma 5.22, the forward orbit $\vec{x}(t)$ of this point must leave C at some time T_0. By lemma 5.21 it cannot leave C through its sides, and thus must leave through the end where its x-coordinate is 1.

Since the x-coordinate of the orbit is continuous, by the intermediate value theorem for every $\delta \leqslant s \leqslant 1$ there is a $0 \leqslant t^* \leqslant T_0$ such that $x_x(t^*) = s$. By lemma 5.18 the orbit is contained in Γ for all times $t \leqslant T_0$, when it leaves through the end of Γ.

The same argument can be made for the point $\vec{x} = (-\delta, \hat{y}(-\delta), \hat{z}(-\delta))$ to complete the proof. □

5.3.2.3 Two Dimensional Manifold in Three Dimensions

Let $\gamma_u(x, y)$ be the polynomial unstable manifold approximation as computed in the previous section. For ease of notation, we will drop the index u and refer to γ_u simply as γ from now on. Additionally, we will assume that the polynomial approximation is given as the graph of a function over the $x - y$ plane, i.e.

$$\gamma : \begin{cases} \mathbb{R}^2 \to & \mathbb{R}^3 \\ \gamma(x,y) = & \begin{pmatrix} x \\ y \\ z(x,y) \end{pmatrix} \end{cases} \tag{5.17}$$

where $z : \mathbb{R}^2 \to \mathbb{R}$ is a polynomial without constant or linear part.

Remark 5.25. This requirement can always be satisfied since the manifold is tangent to the x-axis at the origin. Furthermore, such polynomials are easily obtained using the "P-convention" in the construction as described in Section 5.2.3.2.

If the polynomial is constructed with some scaling factors η_1, η_2 in the x and y coordinate (see section 5.2), without loss of generality one can consider the coordinate transform $T(x, y, z) = (x/\eta_1, y/\eta_2, z)$ and the new vector field $F'(x, y, z) = T \circ F \circ T^{-1}$ as well as $\gamma' = T \circ \gamma$.

Definition 5.26. For some $\varepsilon > 0$ fixed, let $\Gamma : [-1, 1]^2 \times [-1, 1] \to \mathbb{R}^3$ be defined as

$$\begin{aligned} \Gamma(s_1, s_2, t) &= \gamma(s_1, s_2) + \varepsilon \cdot t \cdot \vec{e}_z \\ &= \begin{pmatrix} s_1 \\ s_2 \\ z(s_1, s_2) + \varepsilon t \end{pmatrix} \end{aligned} \tag{5.18}$$

That is, Γ is a uniform thickening of the unstable manifold approximation along the stable direction.

In the following, we will refer to Γ as both the function $\Gamma(s_1, s_2, t)$ as well as the set Γ representing the range of $\Gamma(s_1, s_2, t)$ over its domain. The exact meaning will be made clear by the context.

Clearly Γ contains the local unstable manifold W_u^{loc} as it contains a neighborhood of the origin since

$$D\Gamma(0,0,0) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \varepsilon \end{pmatrix} \text{ and } \Gamma(0,0,0) = 0.$$

We call the following objects the *sides* of Γ:

$$S_z^{\pm}\Gamma(s_1, s_2) \quad = \quad \Gamma(s_1, s_2, \pm 1)$$

where again $s_1, s_2 \in [-1, 1]$. The union of both sides $S_z^{+}\Gamma$ and $S_z^{-}\Gamma$ is referred to as $S\Gamma$.

Similarly, the *ends* of Γ are given by

$$\begin{aligned} E_x^{\pm}\Gamma(s,t) &= \Gamma(\pm 1, s, t), \\ E_y^{\pm}\Gamma(s,t) &= \Gamma(s, \pm 1, t) \end{aligned}$$

where again $s, t \in [-1, 1]$.

All ends of Γ are collectively referred to as $E\Gamma$ defined as

$$E\Gamma = E_x^{+}\Gamma \cup E_x^{-}\Gamma \cup E_y^{+}\Gamma \cup E_y^{-}\Gamma.$$

Figure 5.3 illustrates the concepts of Γ and its sides and ends.

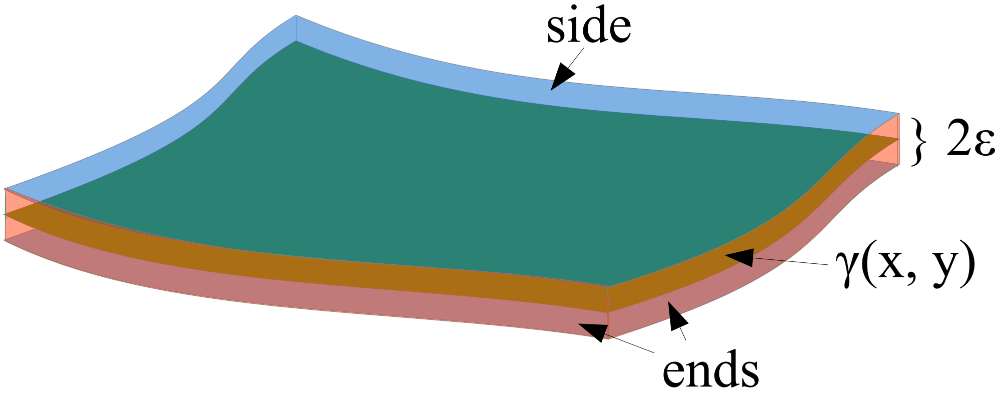


Figure 5.3: The enclosure Γ for a curve $\gamma(x,y)$ in $\mathbb{R}^3$ with its sides (blue) and ends (red) highlighted.

Lemma 5.27. *If the negative vector field F on $S\Gamma$ always points outwards of Γ, the forward orbit of any point in Γ does not leave Γ through $S\Gamma$.*

Furthermore, the negative vector field F on $S_z^{\pm}\Gamma$ points outwards Γ if for all $s_1, s_2 \in [-1,1]$

$$\pm\left(-\frac{\partial z}{\partial s_1}z(s_1,s_2)\cdot F_x\left(S_z^{\pm}\Gamma(s_1,s_2)\right)-\frac{\partial z}{\partial s_2}z(s_1,s_2)\cdot F_y\left(S_z^{\pm}\Gamma(s_1,s_2)\right)+F_z\left(S_z^{\pm}\Gamma(s_1,s_2)\right)\right)<0.$$

Proof. If the negative vector field F on $S\Gamma$ always points outwards of Γ, the backward orbit $\vec{x}(t)$ of any point $\vec{x} \in S\Gamma$ leaves Γ. Hence W_u^{loc}, for which the entire backward orbit is contained in Γ, cannot intersect $S\Gamma$.

Consider now $S_z^{+}\Gamma(s_1,s_2)$. The tangent vectors to this surface are given by

$$\begin{pmatrix} 1 \\ 0 \\ \frac{\partial z}{\partial s_1}z(s_1,s_2) \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ \frac{\partial z}{\partial s_2}z(s_1,s_2) \end{pmatrix}.$$

Thus the outward normal vector to the surface is given by $N(s,t) = \begin{pmatrix} -\frac{\partial z}{\partial s_1} z(s_1, s_2) \\ -\frac{\partial z}{\partial s_2} z(s_1, s_2) \\ 1 \end{pmatrix}$.

This vector indeed points outward of Γ because it points in the positive z direction, and by definition of Γ $S_z^+\Gamma(s_1, s_2)$ is the top boundary of Γ with the largest z value for fixed s_1, s_2.

Thus the vector $-F\left(S_z^+\Gamma(s_1, s_2)\right)$ points outward if $N(s,t) \cdot F\left(S_z^+\Gamma(s_1, s_2)\right) < 0$ which is equivalent to

$$-\frac{\partial z}{\partial s_1} z(s_1, s_2) \cdot F_x\left(S_z^+\Gamma(s_1, s_2)\right) - \frac{\partial z}{\partial s_2} z(s_1, s_2) \cdot F_y\left(S_z^+\Gamma(s_1, s_2)\right) + F_z\left(S_z^+\Gamma(s_1, s_2)\right) < 0.$$

The same analysis performed for the other side yields the remaining condition with the normal vector pointing in the opposite direction and thus the sign of the condition flipped. □

We now introduce another set $C = C_x \cup C_y$ as follows:

Definition 5.28. For some $\vartheta > 0$ fixed, let $C_x, C_y : [-1,1] \times [-1,1]^2 \to \mathbb{R}^3$ be defined as

$$\begin{aligned} C_x(s, t_1, t_2) &= s \cdot \begin{pmatrix} 1 \\ t_1 \\ \vartheta t_2 \end{pmatrix}, \\ C_y(s, t_1, t_2) &= s \cdot \begin{pmatrix} t_1 \\ 1 \\ \vartheta t_2 \end{pmatrix}. \end{aligned}$$

That is, C_x, C_y are pyramids around the x and y axes, respectively, with a small opening angle in the z direction, and a 90 degree opening angle in the $x-y$ plane.

We call the following objects the *sides* of C:

$$\begin{aligned} S_x^{\pm}C(s,t) &= C_x(s,t,\pm 1) \\ S_y^{\pm}C(s,t) &= C_y(s,t,\pm 1) \end{aligned}$$

where again $s,t \in [-1,1]$.

All sides of C are collectively referred to as SC defined as

$$SC = S_y^+ C \cup S_y^- C \cup S_z^+ C \cup S_z^- C.$$

Similarly, the *ends* of C are given by

$$\begin{aligned} E_x^{\pm}C(t_1,t_2) &= C_x(\pm 1, t_1, t_2), \\ E_y^{\pm}C(t_1,t_2) &= C_y(\pm 1, t_1, t_2), \end{aligned}$$

where again $t_1, t_2 \in [-1,1]$.

All ends of C are collectively referred to as EC defined as

$$EC = E_x^+ C \cup E_x^- C \cup E_y^+ C \cup E_y^- C.$$

To illustrate these concepts, figure 5.4 shows an example of such a set C.

Remark 5.29. The boundary ∂C of C consists of $\partial C = EC \cup SC$. This is because the remaining boundaries of C_x and C_y are identical and thus do not form boundaries of the

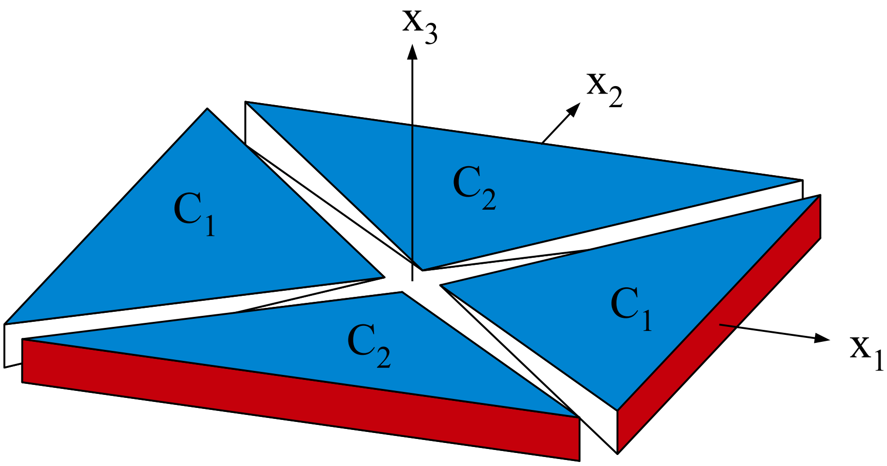


Figure 5.4: The set C with its ends EC (red) and sides SC (blue) highlighted. Each C_x and C_y consist of two pyramids around the x and y axis respectively.

combined C.

Remark 5.30. Clearly C contains some open neighborhood of the local unstable manifold around the origin because C contains the origin and the manifold at the origin is tangent to the $x-y$ plane, while all sides of C have a non-zero slope in the x and y directions.

Lemma 5.31. *If the negative vector field F on $SC\backslash\{0\}$ always points outwards of C, the forward orbit of any point in C does not leave C through SC.*

Furthermore, the negative vector field F on $S_x^{\pm}C\backslash\{0\}$ points outwards C if for all $s \in [-1,1]\backslash\{0\}$ and $t \in [-1,1]$

$$\pm\frac{1}{s}\left(-\vartheta\cdot F_x\left(S_x^{\pm}C(s,t)\right)+F_z\left(S_x^{\pm}C(s,t)\right)\right) \quad < \quad 0$$

and similarly for $S_y^{\pm}C\backslash\{0\}$

$$\pm\frac{1}{s}\left(-\vartheta\cdot F_y\left(S_y^{\pm}C(s,t)\right)+F_z\left(S_y^{\pm}C(s,t)\right)\right) \quad < \quad 0.$$

Proof. If the negative vector field F on $SC\backslash\{0\}$ always points outwards of C, the backward orbit $\vec{x}(t)$ of any point $\vec{x} \in SC\backslash\{0\}$ leaves C. Hence no orbit from within C can intersect SC. Furthermore, 0 is a fixed point, and thus no orbit leaves C through 0 either.

Consider now the case of $S_x^+C(s,t)$. The tangent vectors to this surface for $s \in [-1,1]\backslash\{0\}$ and $t \in [-1,1]$ are given by

$$\begin{pmatrix} 1 \\ \vartheta t \\ \vartheta \end{pmatrix}, \begin{pmatrix} 0 \\ s \\ 0 \end{pmatrix}.$$

Thus the outward normal vector to the surface is given by $N = \begin{pmatrix} -\vartheta \\ 0 \\ 1 \end{pmatrix}$ for $s > 0$. This vector indeed points outward of C because it points in the positive z direction, and by definition of C $S_x^+C(s,t)$ is the top boundary of C with the largest z value for fixed $s > 0$, t. Similarly, for $s < 0$ $S_x^+C(s,t)$ switches from top to bottom boundary, and the outward normal vector here is given by $N = -\begin{pmatrix} -\vartheta \\ 0 \\ 1 \end{pmatrix}$.

Thus the vector $-F\left(S_x^+C(s,t)\right)$ points outward if $N(s,t) \cdot F\left(S_x^+C(s,t)\right) < 0$ which is equivalent to

$$\begin{cases} -\vartheta \cdot F_x\left(S_x^+C(s,t)\right) + F_z\left(S_x^+C(s,t)\right) < 0 & \text{for } s > 0, \\ -\vartheta \cdot F_x\left(S_x^+C(s,t)\right) + F_z\left(S_x^+C(s,t)\right) > 0 & \text{for } s < 0. \end{cases}$$

Since $s \neq 0$, we can divide by s and unify this condition to read

$$\frac{1}{s}\left(-\vartheta \cdot F_x\left(S_x^+ C(s,t)\right) + F_z\left(S_x^+ C(s,t)\right)\right) \;<\; 0.$$

This expression will later turn out to be easier to treat numerically, as well. □

Lemma 5.32. *If the vector field F on EC always points outwards of C, the backward orbit of any point in C does not leave C through EC.*

Furthermore, the vector field F on EC points outwards of C if for all $t_1, t_2 \in [-1,1]$

$$\pm F_x\left(E_x^\pm C(t_1,t_2)\right) \;>\; 0$$

and similarly for $E_y^\pm C$

$$\pm F_y\left(E_y^\pm C(t_1,t_2)\right) \;>\; 0.$$

Proof. If the vector field F on EC always points outwards of C, the forward orbit $\vec{x}(t)$ of any point $\vec{x} \in EC$ leaves C. Hence no orbit from within C can intersect EC.

Consider now the case of $E_x^+ C(t_1,t_2)$. The tangent vectors to this surface for $t_1, t_2 \in [-1,1]$ are given by

$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \vartheta \end{pmatrix}.$$

Thus the outward normal vector to the surface is given by $N = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$. This vector indeed points outward of C because it points in the positive x direction, and by definition of C $E_x^+ C$ is the right boundary of C with the largest x value.

Thus the vector $F\left(E_x^+ C(t_1, t_2)\right)$ points outward if $N(s,t) \cdot F\left(E_x^+ C(t_1, t_2)\right) > 0$ which is equivalent to

$$F_x\left(E_x^+ C(t_1, t_2)\right) \quad > \quad 0.$$

The same analysis performed for the other three sides yields the remaining conditions. □

Lemma 5.33. *If the following condition holds, then for every $\delta > 0$ there is a $T > 0$ such that for any orbit $\vec{x}(t) = (x_x(t), x_y(t), x_z(t))$ that is contained in $C \backslash \{0\}$ for all $t \in [-T, 0]$ the expression $|x_x(-T)|^2 + |x_y(-T)|^2 < \delta$ holds:*

$$F_x(\vec{x}) \cdot x_x + F_y(\vec{x}) \cdot x_y > 0 \;\forall \vec{x} = (x_x, x_y, x_z) \in C \backslash \{0\} \tag{5.19}$$

Proof. Consider the expression

$$\begin{aligned} \frac{d}{dt}\left(|x_x(t)|^2 + |x_y(t)|^2\right) &= \frac{d}{dt}\left(x_x(t)^2 + x_y(t)^2\right) \\ &= \dot{x}_x(t) \cdot x_x(t) + \dot{x}_y(t) \cdot x_y(t) \\ &= F_x(\vec{x}(t)) \cdot x_x(t) + F_y(\vec{x}(t)) \cdot x_y(t). \end{aligned}$$

Since F is continuous and by virtue of condition 5.19, this expression assumes a strictly positive minimum M_δ on every compact set $C\backslash B_\delta(0)$. Thus for $T = \frac{\delta}{M_\delta}$ and any orbit $\vec{x}(t)$ contained in $C\backslash\{0\}$ for all $t \in [-T, 0]$ we have that

$$|x_x(-T)|^2 + |x_y(-T)|^2 < M_\delta \cdot T < \delta.$$

□

Remark 5.34. As was the case with the one dimensional manifold, for this argument to work, it is necessary to consider some set like C which includes the origin, but does not include a full neighborhood of it. This is because in any neighborhood B of the origin, this condition can never be satisfied, as B would include the local stable manifold. Thus a set has to be chosen that does not include any points of the stable manifold.

Again this automatically precludes testing for these conditions in Γ and requires the introduction of C instead.

Theorem 5.35 (Manifold Enclosure Theorem (2D))**.** *If there exist $\varepsilon, \vartheta > 0$ such that the conditions of the above lemmata 5.27 through 5.33 are satisfied, then also $\forall s_1, s_2 \in [-1, 1]$ $\exists t_1 \in [-1, 1]$ such that $\Gamma(s, t_1, t_2) \in W_u^{loc}$ is satisfied.*

Proof. Let $\zeta(r)$, $r \in [0, 1]$ be a continuous curve within $C\backslash\{0\}$ connecting some point $\zeta(0)$ on the top side to some point $\zeta(1)$ on the bottom side of C. Let $\tau(r) \leqslant 0$ be the largest time such that $\Phi_{\tau(r)}(\zeta(r)) \in SC$, or let $\tau(r) = -\infty$ if no such time exists. By lemma 5.31 the function $\tau(r)$ is continuous wherever it is finite, since the vector field on the top boundary is never tangent to the boundary and thus each point on SC leaves C together with a small neighborhood around it. Furthermore, by lemma 5.32 for any time $0 > t > \tau(r)$ the orbit $\Phi_T(\zeta(r))$ is within C.

Assume now every point on $\zeta(r)$ intersects SC at some time $-\infty < T < \tau(r)$. Then consider the continuous function $\Phi_{\tau(r)}(\zeta(r))$. We have that $\Phi_{\tau(r)}(\zeta(0))$ is in the top side of C, and $\Phi_{\tau(r)}(\zeta(1))$ is in the bottom side of C. However, the only point connecting the top and bottom sides of C is the origin, and thus for some $s^* \in [0,1]$ we have that $\Phi_{\tau(r)}(\zeta(s^*)) = 0$. That is impossible since no orbit starting away from the origin goes through the origin in finite time.

Thus for some $r^* \in [0,1]$ we must have that $\tau(r^*) = -\infty$, and then by lemma 5.33 $\lim_{t\to-\infty} \Phi_t(\zeta(r^*)) = 0$. Thus $\zeta(r^*) \in W_u^{loc}$.

Assume now that $\exists\, s, t_1 \in [-1,1], s \neq 0$ such that $\forall\, t_2 \in [-1,1]$ we have $C_x(s, t_1, t_2) \notin W_u^{loc}$. Then let ζ be the curve connecting $C_x(s, t_1, 0)$ and $C_x(s, t_1, 1)$. Certainly ζ meets the above criteria, and thus must contain a point $p \in \zeta$ such that $p \in W_u^{loc}$, contradicting the assumption. The same holds for C_y.

Since Γ contains a neighborhood of the origin, $\exists\, T > 0$ such that $\Phi_{-T}(p) \in \Gamma$. Then also $p \in \Gamma$, as by lemma 5.27 the forward orbit of $\Phi_{-T}(p)$ cannot leave Γ except through the ends. Thus $\forall\, s_1, s_2 \in [-1,1]\ \exists\, t \in [-1,1]$ such that $\Gamma(s_1, s_2, t) \in W_u^{loc}$. □

Remark 5.36. This proof is much more involved than its one dimensional counterpart in theorem 5.24. This is mostly due to the topological aspects of this proof. We hope that further research on the topology of this problem will allow a significant simplification of this proof.

5.3.3 Numerical Verification

In the following, we will describe the numerical verification of the above conditions. Computations will be performed in Taylor Model arithmetic, and we first recall a few facts about

Taylor Model arithmetic.

Taylor Model Derivative

Recall that in Taylor Model arithmetic in general it is not possible to obtain an enclosure of the derivatives of a function from a Taylor Model enclosure of the function itself. While it is possible to differentiate the polynomial part of the Taylor Model, we have no information about the derivative of the remainder term. This is because it represents a purely C^0 bound of the truncation and numerical round-off errors.

However, if the function itself is a polynomial of order less than the current computation order to begin with, it is simple to compute its derivative. We will use this fact to compute the derivatives of the manifold approximation γ_u (but not the manifold itself) rigorously.

Integral-Division Operator

In general, in Taylor Model arithmetic it is also not possible to divide by an independent variable x_i. This is because in the differential algebra x_i does not have an inverse element. Yet even if every coefficient in the polynomial part $P(x)$ of a Taylor Model depends on x_i, and thus $Q(x) = P(x)/x_i$ is a polynomial, usually this division cannot be performed rigorously. This is because the error bound R of the Taylor Model in general does not scale with x_i, so that the Taylor Model $P(x) \pm R$, when divided by x_i would be of the form $Q(x) + B(R/x_i)$ where B is the bounding operation over the Taylor Model domain $[-1, 1]$. Obviously, the expression for the error bound does not have a bound on that domain unless $R = 0$.

However, under certain very specific circumstances it is possible to rigorously divide a Taylor Model by an independent variable x_i. For this to be possible, the remainder error R

of the Taylor Model must a priori be known to scale at least linearly with x_i and, of course, every coefficient of the polynomial part must depend on x_i. Then such a division can be performed by simply leaving the remainder term untouched and the resulting Taylor Model is indeed a valid enclosure of the desired expression valid for all $x_i \neq 0$ on the Taylor Model domain.

In particular, consider the operator $\frac{1}{x_i}\int_0^{x_i} \cdot \, dx_i$. This operator should be read as one single operator, not as an integration followed by a division. When applied to a Taylor Model $T = P(x) \pm R$ the result is

$$\frac{1}{x_i}\int_0^{x_i} T \, dx_i = \frac{1}{x_i}\int_0^{x_i} P(x) \, dx_i \pm \frac{1}{x_i}\int_0^{x_i} R \, dx_i.$$

The polynomial part $P(x)$ can easily be integrated, as can be R, which is simply $R \cdot x_i$. The following division by x_i can thus be performed on both the integrated polynomial part as well as the integrated error bound, which right after integration scales linearly with x_i. The resulting Taylor Model is a rigorous enclosure of the operator $\frac{1}{x_i}\int_0^{x_i} \cdot \, dx_i$, which is well defined for all $x_i \neq 0$, applied to T which is valid over the entire Taylor Model domain $[-1, 1]$ except at $x_i = 0$.

We will apply this operator in the following form. Consider the vector field F evaluated over a set of the form

$$K(s, \vec{t}) = s \cdot \vec{v}(s, \vec{t})$$

where $s \in [-1, 1] \backslash \{0\}$ and $\vec{v} \in \mathbb{R}^3$. The vector $\vec{v}$ may additionally depend on any number of other parameters $\vec{t} \in \mathbb{R}^k$.

Note that since $F(0) = 0$, we have that

$$\begin{aligned} F\left(s \cdot \vec{v}(s,\vec{t})\right) &= \int_0^s \frac{d}{ds} F\left(s \cdot \vec{v}(s,\vec{t})\right)\, ds + \underbrace{F(0 \cdot \vec{v}(0,\vec{t}))}_{=0} \\ &= \int_0^s DF\left(s \cdot \vec{v}(s,\vec{t})\right) \cdot \vec{v}(s,\vec{t})\, ds. \end{aligned}$$

If we are only interested in what happens for $s \neq 0$, we can divide by s and obtain

$$\frac{1}{s} F\left(s \cdot \vec{v}(s,\vec{t})\right) = \frac{1}{s} \int_0^s DF\left(s \cdot \vec{v}(s,\vec{t})\right) \cdot \vec{v}(s,\vec{t})\, ds \tag{5.20}$$

which is well defined for all $s \in [-1, 1] \backslash \{0\}$.

While the expression on the right looks complicated, it can be evaluated rather easily in Taylor Model arithmetic using the combined integral-division operator introduced above, as long as the derivative DF of F is known analytically.

The advantage of this form is that near zero, this expression is bounded away from zero. Note that for fixed $0 \neq \vec{x} \in \mathbb{R}^n$ and F diagonal

$$F(s \cdot \vec{x}) = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \cdot s \cdot \vec{x} + \mathcal{O}\left(s^2\right).$$

Thus the expression

$$\frac{1}{s} F\left(s \cdot \vec{v}(s,\vec{t})\right) = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \cdot \vec{v}(s,\vec{t}) + \mathcal{O}\left(s\right)$$

has a non-zero limit as $s \to 0$ as long as $\lim_{s\to 0} \vec{v}(s,\vec{t}) \neq 0$.

5.3.3.1 Verification using Taylor Models

The following steps are performed to verify the various conditions for the manifold enclosure. For all of these computations, note that the enclosure candidate Γ can easily be written as a a Taylor Model. The choice of ε can be automated by testing increasingly larger values until all conditions are met, or a value can simply be chosen. For double precision computations, good starting points are values on the order of 10^{-13}.

The same is true for the set C and the value ϑ used in its construction. Since here the enclosure does not need to be very sharp, the value for ϑ can be chosen rather large. We found that using $\vartheta = 0.1$ works quite well.

Verification of Lemmata 5.18, 5.27 and 5.32

Straight forward evaluation of each separate condition, with subsequent bounding of the result allows the verification of these conditions. Note that since γ is a polynomial, its derivatives are readily computed and Taylor Model enclosures of the derivatives used in these expressions can be obtained.

Moreover, since all of these conditions are evaluated over sets not containing zero, the vector field F does not vanish. Thus the direct evaluation and bounding of F does not pose a problem if ε is chosen big enough.

Verification of Lemmata 5.21, 5.22

Note that the expressions in lemmata 5.21 and 5.31 are of the form

$$\begin{aligned} & \frac{1}{s}\left(-\vartheta \cdot F_i\left(s \cdot \vec{v}\right) + F_j\left(s \cdot \vec{v}\right)\right) \\ = \quad & -\vartheta \cdot \frac{1}{s} F_i\left(s \cdot \vec{v}\right) + \frac{1}{s} F_j\left(s \cdot \vec{v}\right) \end{aligned}$$

where $i, j = x, y, z,\ s \in [-1, 1]\backslash\{0\}$ and $\vec{v} \in \mathbb{R}^3$ is some vector depending on additional parameters $\vec{t}$.

This is precisely the form of expression used in equation 5.20, and thus the $\frac{1}{s}F_i$, $i = 1, 2, 3$ used in this expression can be computed using Taylor Models. Bounding the entire expression in Taylor Model arithmetic allows for easy verification of this condition.

Verification of Lemmata 5.31 and 5.33

Consider the expressions

$$F_x(\vec{x}) \cdot x_x,$$

$$F_x(\vec{x}) \cdot x_x + F_y(\vec{x}) \cdot x_y$$

in lemmata 5.22 and 5.33 respectively. Both of them are evaluated for $\vec{x} = (x_x, x_y, x_z)$ of the form

$$\vec{x} = s \cdot \vec{v}$$

where $s \in [-1, 1]\backslash\{0\}$ and $\vec{v} = (v_x, v_y, v_z) \in \mathbb{R}^3$ is some vector depending on additional parameters $\vec{t}$.

Inserting this into the above expression yields

$$F_x(s\vec{v}) \cdot sv_x,$$

$$F_x(s\vec{v}) \cdot sv_x + F_y(s\vec{v}) \cdot sv_y.$$

Since $s \neq 0$, this can be rewritten as

$$s^2 \cdot \left(\frac{1}{s} F_x(s\vec{v}) \cdot v_x \right),$$

$$s^2 \cdot \left(\frac{1}{s} F_x(s\vec{v}) \cdot v_x + \frac{1}{s} F_y(s\vec{v}) \cdot v_y \right).$$

Now by again applying equation 5.20, the expressions $\frac{1}{s}F_i(s\vec{v})$ can easily be computed in Taylor Model arithmetic. Thus the entire expression in parenthesis can be computed as a Taylor Model and subsequently bounded. Since s^2 is always positive for $s \neq 0$, if this expression is always positive, then

$$F_x(\vec{x}) \cdot x_x > 0,$$

$$F_x(\vec{x}) \cdot x_x + F_y(\vec{x}) \cdot x_y > 0.$$

5.3.4 Manifold Propagation

After verifying an initial enclosure of the manifolds, these enclosures can be propagated to a much larger size using verified integration. In particular, since the initial enclosure is available directly as a Taylor Model, this step can be performed using the COSY VI verified integrator, which is described in detail in [34, 10, 35]. Using this tool, it is possible to rigorously transport an initial Taylor Model through a flow resulting in a verified Taylor Model enclosure of the final Taylor Model.

At the core of this integrator lies the computation of a rigorous enclosure of the flow of the ODE for some suitably chosen time step for the given initial condition. That is, in each step the verified integrator computes a rigorous enclosure of $\Phi_t(x_i)$ valid for some initial condition x_i, which itself is a Taylor Model, and any time $t \in [0, \Delta t]$ for some time step Δt[35]. There are various additional operations performed during the integration in order to keep the growth of the error bound of the Taylor Model under control. These are described in detail in [10, 34] and we will not discuss those methods here. Furthermore, the verified integrator automatically splits the initial condition into smaller pieces if it considers such splits necessary to maintain optimal precision. The integration is then performed on each piece separately, resulting in a list of final Taylor Models covering the final result.

Using such a verified integrator, it is possible to "grow" the invariant manifolds by integrating the unstable manifold forward and the stable manifold backward in time. Such an integration would produce a new enclosure of part of the manifold, covered by a collection of Taylor Models. While in principle this is a possible way to integrate the entire verified manifold enclosure, it is not ideal to "grow" the manifold.

5.3.4.1 Initial Condition

Recall that with any point x on the manifold, also automatically its entire orbit $x(t)$ is part of the manifold. Thus in order to propagate the invariant manifold enclosure, it is sufficient to integrate a small ball around the origin within the manifold. In fact, the flow of any initial condition I_0 within the manifold that is transversal to the vector field F will trace out part of the manifold.

The advantage in choosing such an initial condition is that it has one less dimension than the entire manifold, making the verified integration faster and introducing fewer errors during the integration thus increasing the accuracy.

Given a verified Taylor Model enclosure $\Gamma(\vec{x}, \vec{t})$ of the manifold, where $\vec{t}$ are the error variables representing the slight thickening, we typically simply chose the initial enclosure to be the ball of radius 1 for $\vec{x}$ as the initial condition, i.e. $I_0 = \Gamma(\vec{x}, \vec{t})$ for $|\vec{x}| = 1$. In practice, of course, this initial condition will not be represented by one single Taylor Model, but instead be split up into several smaller Taylor Models which together cover the entire ball $|\vec{x}| = 1$. This also allows to only integrate "interesting" initial conditions, that lead to manifold enclosures of the the relevant parts of the manifold.

Since the Taylor Model integrator COSY VI computes verified rigorous enclosures of the flow $\Phi_t(x)$ in each integration step, a Taylor Model representation of the flow is readily available with this method. By storing each of these flow expansions, a covering of the manifold by these flows is obtained.

5.3.4.2 Manifold Growth

Either way, one problem with the propagation of the manifold in this way is the rapid growth of the manifold. Locally, the growth of the manifold is given by the eigenvalues λ_i of the Jacobian $DF(0)$ at the origin. For the each manifold, the linear growth in the direction of the eigenvectors is exponential in time, given by

$$x_i(t) = x_i(0) \cdot \exp(\left|\lambda_i\right| t)$$

where x_i is the distance along the eigenvector. The larger the absolute value of the eigenvalues, the faster the manifold grows.

A problem appears if the different eigenvalues of a higher dimensional invariant manifold are significantly different. If that is the case, the manifold will grow very quickly along the larger eigenvalue, while it will grow relatively slowly along the smaller eigenvalue. This poses various problems for the verified integration.

First, the manifold in the direction of the major expanding eigenvalue may not be the relevant part of the manifold for a given problem. Thus, integrating the entire manifold in this direction is a waste of computation time.

There are several strategies to counter this undesirable growth of the manifold under integration. Two easily implemented techniques are described here, which one is more efficient to use depends on the problem at hand.

Since the COSY VI integrator supports the splitting of initial conditions, it is relatively easy to limit the size of each initial condition in each integration step. If an initial condition exceeds a certain predefined size, it is automatically split along the longest side. Thus the propagation moves a growing number of Taylor Models through the ODE. To restrict the growth, one can automatically discard such pieces that have left a predefined area of interest completely. This effectively takes care of pieces that quickly grow along the direction of the largest eigenvalue. The enforced splitting above a certain size ensures that more pieces actually leave the area of interest faster, instead of having pieces that are mostly outside the area of interest but still have some small overlap. Clearly, the maximum allowed size for each piece should be chosen smaller than the area of interest.

Another technique is not to wait until a given piece of the initial condition has left the

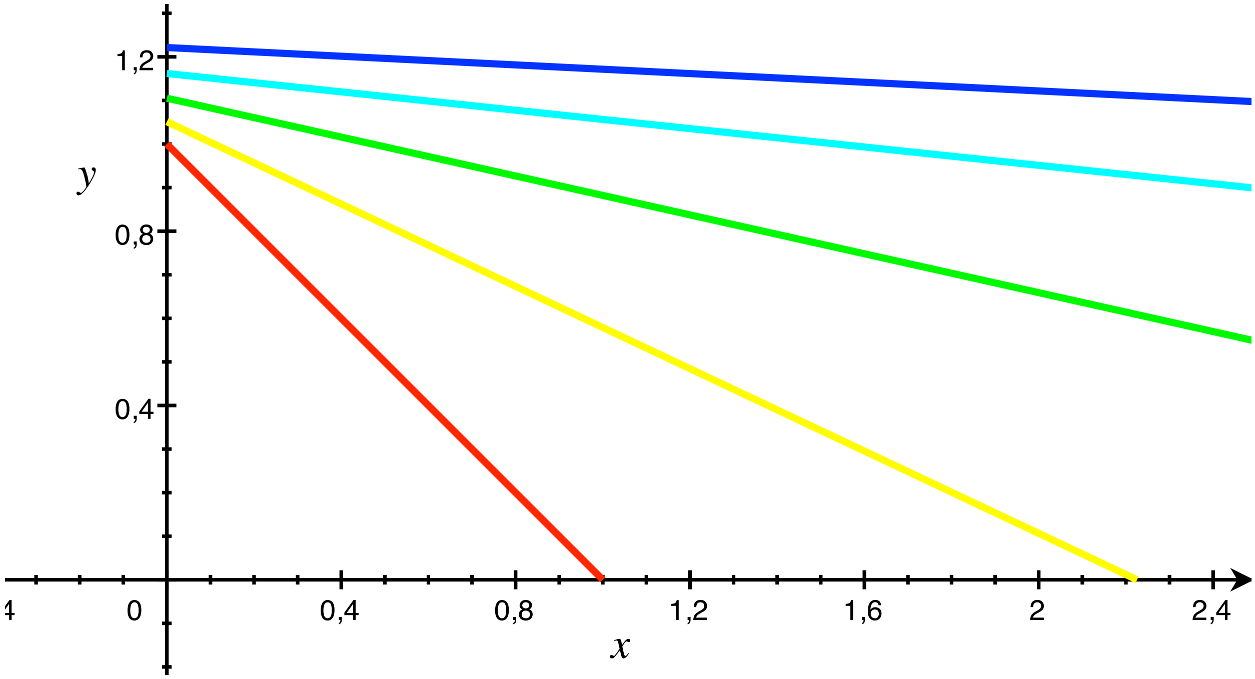


Figure 5.5: Alignment along the strongest expanding eigenvector for a simple linear ODE. The initial condition is shown in red, and then integrated forward for times $T = 0.1$ (yellow), $T = 0.3$ (green), $T = 0.3$ (light blue), $T = 0.4$ (dark blue).

box of interest and then discard it, but discarding parts of the initial condition in every integration step. This is particularly useful if the manifold is only needed in a neighborhood of some known point x_0 on the initial manifold enclosure.

Consider an initial condition $I(s)$, $s \in [-1, 1]$ such that $I(0) = x_0$. In each integration step a rigorous enclosure of $\Phi_t(I(s))$ is computed for some $t > 0$. This flow is then only evaluated over some smaller domain such as $s \in [-1, 1]/\exp(\lambda t)$ and rescaled to the domain $[-1, 1]$. The initial condition for the next integration step is thus given by $I_1 = \Phi_t(I(s/\exp(\lambda t)))$, $s \in [-1, 1]$. Clearly in each step a neighborhood of $\Phi_t(x_0)$ is retained, while the “ends” of the initial condition are discarded. If λ is chosen correctly in each step, this approach allows the computation of a band of relatively constant size on the manifold. Typically, the largest eigenvalue λ_{max} is a good choice for λ.

Another more subtle problem arising from very different eigenvalues is that the strong expansion in one direction causes the initial condition to align with that expanding direction after longer integration times. This alignment with the strongest expanding eigenvector also means that the initial condition becomes more and more parallel to the vector field, thus causing the area each flow expansion covers to become smaller and smaller. That is particularly noticeable if parts of the initial condition leaving a given bounding box are discarded.

Consider, for example, the two dimensional case of a linear ODE, with eigenvalue $\lambda_1 = 8$ along the x-axis and $\lambda_2 = 0.5$ along the y-axis. Then the initial condition $I(s) = (1, 0) + s \cdot (-1, 1)$, $s \in [0, 1]$ forming a line from $(1, 0)$ to $(0, 1)$ is stretched out significantly more along the x-axis than along the y-axis as it is integrated over time, resulting in an object almost parallel to the x-axis after long time. This effect is shown in figure 5.5, where the initial condition I (red) is integrated for times $T = 0.1$ (yellow), $T = 0.3$ (green), $T = 0.3$ (light blue), and $T = 0.4$ (dark blue). Note how the area the rays sweep out between two subsequent time steps becomes smaller and smaller in the shown bounding box.

This problem is, of course, intrinsic to the ODE, and thus cannot be circumvented. However, there are possible techniques to prevent the stretching of most pieces of the initial condition. In the above, integration was always performed using fixed time steps of a given size (e.g. 0.1 in the example) for every point in the initial condition. However, since the Taylor Model integrator computes a validated flow expansion before inserting a time, there is nothing preventing us from inserting a polynomial expression for the time step depending on the parametrization of the initial condition. Mathematically speaking, instead of computing $\Phi_{t_1}(I(s))$ for some fixed $t_1 \in \mathbb{R}$ and some initial condition $I(s)$ parametrized by $s \in D \subset \mathbb{R}^n$, the expression $\Phi_{t(s)}(I(s))$ for some suitably chosen $t(s)$ is thus computed.

In particular, one can attempt to chose $t(x)$ in such a way, that points growing faster are delayed by using smaller time steps, while points growing slower are sped up by using larger time steps. Consider the above example, and some subset $I_1(s)$, $s \in [0, 1]$ of the initial condition I not including the point $(0, 1)$. Then every point in I_1 moves to the right. If the time step is chosen as $t(s) = \ln\left(\frac{x_f}{x(s)}\right)/\lambda_1$, where $x(s)$ indicates the x coordinate of a point in I_1 and $x_f \in \mathbb{R}^+$, then the map $\Phi_{t(s)}(I_1(s))$ takes I_1 to a vertical line with $x(s) = x_f$. If this is performed in each integration step for some suitably chosen x_f, the result of each step is always almost perpendicular to the flow, thus providing the best possible coverage while controlling the expansion at the same time.

Lastly, there is an inherent numerical problem due to the stretching of the initial condition happening during the integration. Not only the length of the initial condition, but also the size of the unavoidable integration errors grows exponentially. While points on the eigenvector of the smaller eigenvalue move relatively slowly, locally all points slightly off are moving away exponentially with the stronger eigenvector. This leads to rapid growth of the error bound, which presents the integrator with significant problems and limits the maximum possible integration time.

This problem can be addressed by either using High Precision Taylor Models to perform the integration, or to some extend by again using position dependent, non-uniform time steps for each integration, thus controlling the growth of the manifold and thus the errors.

Unfortunately, while the first technique of discarding pieces of the initial condition, or rescaling the initial condition in each step is relatively straight forward in their implementation, the latter two techniques are more complicated to implement. Both require, for example, various non-trivial changes to the COSY VI integrator. It is beyond the scope of this dissertation to implement and explore the possibilities of these approaches. Further research on how to modify COSY VI to construct an optimal verified manifold propagator is ongoing.

5.3.5 Application to the Lorenz System

To demonstrate the verified manifold enclosures we consider the well known Lorenz equations in three dimensions. This system was introduced by Lorenz in 1963[27], and it has been the focus of extended studies in the dynamical systems community ever since.

The standard Lorenz system of differential equations is the three dimensional system of ordinary differential equations given by

$$\begin{aligned} \dot{x}_1 &= \sigma(x_2 - x_1) \\ \dot{x}_2 &= (\rho - x_3)x_1 - x_2 \\ \dot{x}_3 &= x_1 x_2 - \beta x_3 \end{aligned} \tag{5.21}$$

where the real numbers σ, ρ, β are external parameters. The canonical values used are $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. We denote the associated vector field by F. In the following, we refer to the vector field with the canonical values for the parameters as the *Lorenz system.*

This system has a hyperbolic fixed point at the origin, with the Jacobian given by

$$JF|_0 = \begin{pmatrix} -\sigma & \sigma & 0 \\ \rho & -1 & 0 \\ 0 & 0 & -\beta \end{pmatrix}.$$

Thus the eigenvalues of the Jacobian at the origin are given by the analytical expressions

$$\begin{aligned} \lambda_1 &= \frac{-\sigma-1+\sqrt{4\rho\sigma+(1-\sigma)^2}}{2} && \approx 11.8 \\ \mu_1 &= \frac{-\sigma-1-\sqrt{4\rho\sigma+(1-\sigma)^2}}{2} && \approx -22.8 \\ \mu_2 &= -\beta && \approx -2.7 \end{aligned}$$

and the corresponding eigenvectors read

$$e_1 = \begin{pmatrix} 1 \\ \frac{\lambda_1}{\sigma}+1 \\ 0 \end{pmatrix}, e_2 = \begin{pmatrix} 1 \\ \frac{\mu_1}{\sigma}+1 \\ 0 \end{pmatrix}, e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Forming the 3×3 transformation matrix $M = (e_1, e_2, e_3)$ to transform from cartesian into eigen-coordinates, one readily computes the inverse matrix M^{-1} given by

$$M^{-1} = \begin{pmatrix} \frac{\mu_1+\sigma}{\mu_1-\lambda_1} & -\frac{\sigma}{\mu_1-\lambda_1} & 0 \\ -\frac{\lambda_1+\sigma}{\mu_1-\lambda_1} & \frac{\sigma}{\mu_1-\lambda_1} & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

With these expressions the diagonalized Lorenz vector field $\widehat{F} = M^{-1} \circ F \circ M$ can be written analytically. Similarly, the derivative of $\widehat{F}$ is given by $D\widehat{F} = M^{-1} \circ DF \circ M$.

Note that all of the analytical expressions in M and M^{-1} are readily computed using verified techniques, yielding a rigorous enclosure of the correctly diagonalized Lorenz map.

5.3.5.1 Initial Manifold Enclosure

We apply the construction algorithm described in the preceding section to this system. All of the following computations are performed using double precision Taylor Models of order 21.

First, the matrices M and M^{-1} are computed using Taylor Models containing only a constant part. This yields a rigorous representation of $\widehat{F}$ and $D\widehat{F}$. Note that the eigenvectors are ordered in such a way as to yield the same order of eigenvalues as described in Section 5.2. In the non-verified construction steps of the polynomial approximation, these rigorous enclosures are simply replaced by double precision approximations by dropping the error bound of the Taylor Models.

Next, the order by order construction of the polynomial approximation is started. Due to the overall size of the dynamics in the Lorenz system, the free parameter for the derivatives η_i of the polynomial manifold approximation at the origin can be chosen rather large without major loss in the accuracy of the resulting approximation.

We choose the values $\eta_1 = 1$ for the unstable manifold, and various values for η_2 and η_3 for the stable manifold. Since the verification is always performed over the domain $[-1, 1]^n$, the size of the η_i effectively determines the size of the manifold enclosure. Computing the enclosure for various values of η_3 will demonstrate the effect of changing this value on the size and sharpness of the manifold enclosure. In this particular ODE, it is advantageous to make the size of the initial manifold enclosure along the z-axis as large as possible. This is

$\eta_2 \backslash \eta_3$	1	5	10	15	20
1	$5 \cdot 10^{-14}$	$1 \cdot 10^{-13}$	$1 \cdot 10^{-12}$	$1 \cdot 10^{-9}$	$1 \cdot 10^{-7}$
0.5	$1 \cdot 10^{-14}$	$5 \cdot 10^{-14}$	$1 \cdot 10^{-13}$	$5 \cdot 10^{-11}$	$5 \cdot 10^{-8}$
0.2	$5 \cdot 10^{-15}$	$1 \cdot 10^{-14}$	$5 \cdot 10^{-14}$	$1 \cdot 10^{-11}$	$6 \cdot 10^{-9}$

Table 5.1: Thickening of the manifold enclosure (in diagonalized coordinates) for which verification is successful for various values of η_2 and η_3.

because in the later propagation of the manifold enclosure, the z-axis represents the slowest growing direction with its eigenvalue of absolute value $8/3 \approx 2.67$ compared to 22.8 along the y-axis. The effect of this will become painfully obvious during the manifold propagation.

In order to be able to conveniently perform the verification as described in Section 5.3.2, the "P-convention" is chosen for the manifold construction, yielding the manifold approximation as the graph over the x-axis (unstable) or the y-z plane (stable) respectively.

To test the quality of the computed approximations, the construction condition (5.2) is evaluated with the resulting polynomials. For all values of η_i, both sides of this condition indeed agree up to the computation order 21 with individual differences of less than 10^{-17} in the coefficients. This small disagreement is caused by floating point round-off errors during the computation and is to be expected in double precision floating point arithmetic.

For the verification of the one dimensional unstable manifold, the polynomial approximation curve is outfit with a small thickening of size $2 \cdot 10^{-14}$ in both y and z direction in the diagonalized system, which after transforming back into standard Lorenz coordinates translates into a thickness of less than $5.7 \cdot 10^{-14}$. For the two dimensional stable manifold, the initial surface is thickened by some value depending on the choices of η_2 and η_3. Table 5.1 shows the thickening chosen for various combinations of η_2 and η_3 for which verification of the enclosure was successful.

t	1	2	3	4	5	10	15	20	25	28
Approximate Length	126	144	166	189	214	390	710	1156	1592	1879
Maximum Error	$2 \cdot 10^{-11}$	$3 \cdot 10^{-11}$	$5 \cdot 10^{-11}$	$1 \cdot 10^{-10}$	$2 \cdot 10^{-10}$	$2 \cdot 10^{-9}$	$7 \cdot 10^{-8}$	$5 \cdot 10^{-5}$	$7 \cdot 10^{-4}$	$2 \cdot 10^{-2}$
# of Taylor Models	44	72	106	140	175	353	537	718	1091	9196

Table 5.2: Approximate length, maximum error, and number of Taylor Models covering the unstable manifold after propagating for the given time t.

After completing the computation of the verification conditions given in Section 5.3.2, this results in a rigorously verified enclosure of the one dimensional unstable and two dimensional stable manifolds of the origin in the Lorenz system. The computation and verification of all these manifold enclosures of order 21 takes only a couple seconds on a 2GHz Apple MacBook with 2GB RAM.

Note that the sizes of the manifold enclosures are roughly given by the size specified by the η_i during the construction. For the unstable manifold, the length is about $2 \cdot \eta_1$, while for the stable manifold the area is about $2 \cdot \eta_2 \cdot 2 \cdot \eta_3$.

Figure 5.6 shows the initial enclosures of the stable and unstable manifolds in the standard Lorenz coordinates for the case $\eta_1 = \eta_2 = 1$ and $\eta_3 = 10$. The thickness of the stable manifold enclosure shown is less than $2.3 \cdot 10^{-12}$, while the error in the unstable manifold is less than $5.7 \cdot 10^{-14}$. The twisting of the stable manifold around the z-axis is becoming quite visible, indicating that this enclosure is much better than just a linear approximation.

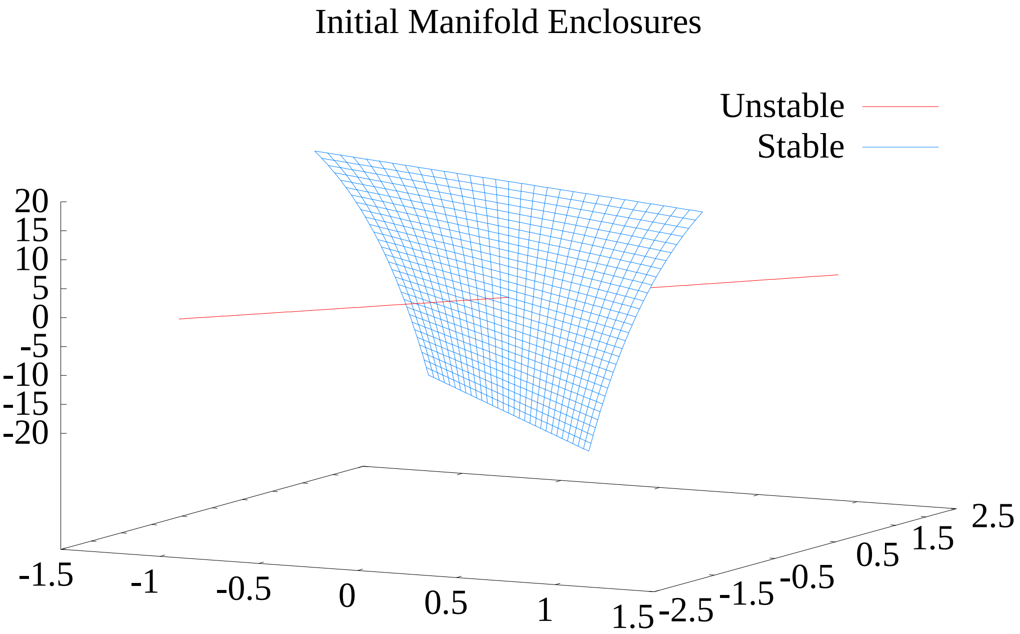


Figure 5.6: The initial manifold enclosures generated for the Lorenz system at the origin shown in standard Lorenz coordinates.

5.3.5.2 Manifold Propagation

The initial manifold enclosure is then integrated forward. Due to the symmetry of the Lorenz equations, for any orbit $(x(t), y(t), z(t))$ also $(-x(t), -y(t), z(t))$ is an orbit. Thus, if the point (x, y, z) is on one of the manifolds, so is $(-x, -y, z)$. The manifold are therefore symmetric, and it is sufficient to integrate an initial condition on the manifold for which $x > 0$, and the other half can be obtained by a simple reflection instead of a lengthy integration.

For the one dimensional unstable manifold this integration is relatively simple. The end point of the initial manifold enclosure, $\Gamma_u(1, \vec{t})$, is integrated forward in time using COSY VI. Since this is a 0 dimensional initial condition (except for the tiny error variables), there are no large problems with the propagation even for large times. COSY VI does not need to split this initial condition, as there are no significant directions to split along. This allows to integrate the unstable manifold forward through the Lorenz equations even for large times. The only limiting factor here is the growth of the remainder bound due to floating point round-off errors.

Figures 5.7 through 5.12 show the growth of the unstable manifold for various times between $t = 1$ and $t = 28$, while Table 5.2 shows the length of the unstable manifold, the number of Taylor Models covering it, and the maximum error for various times up to $t = 28$. Both in the pictures and the table, only one half of the manifold is considered, the other half is symmetric due to the underlying symmetry of the Lorenz equations and is not included in these figures.

To visualize the three dimensional structure of the manifolds, each polynomial flow was covered with a uniform polygon mesh which was then exported into the Meshlab[17] program for display. In the plots of the manifold, it was necessary to thicken the manifold to make it visible at all, the enclosures are much sharper than shown in the pictures.

As can be seen in Table 5.2, for times up to $t = 25$ the integration performs as it should. Once $t = 28$ is reached, however, this is near the end of the possible integration range. This can be seen by the huge number of resulting Taylor Models required to cover the manifold, about 9 times more than at time $t = 25$. This indicates that the time step size COSY VI can take after $t = 25$ drops significantly.

With the two dimensional stable manifold the difference of the eigenvalues of the stable manifold by almost a factor of 10, causes the problems described in Section 5.3.4.2 to appear with full force. Since our goal is to compute as much of the local stable manifold of the origin as possible, we use the cropping integration technique described earlier. Any manifold piece that after an integration step ends up outside the box $B = [-50, 50] \times [-50, 50] \times [-20, 90]$ is automatically discarded from further integration. This allows the computation of the entire local stable manifold up to some final integration time.

The next question is which initial condition to use for the integration. Due to the extreme difference of the growth of the stable manifold in backward time along the direction of the stable eigenvector belonging to the eigenvalue μ_1 compared to that along the z-axis, which is the eigenvector associated with eigenvalue μ_2, it proves to be very beneficial to select an initial condition as large as possible along the z-axis.

In the following, we use the boundary of the initial stable manifold enclosure generated with $\eta_2 = 0.2$ and $\eta_3 = 20$. The small size of this enclosure in the x-y plane is no problem, as this is aligned in the direction of the strongly expanding eigenvector. Even though this larger initial condition is less accurate than one of the smaller ones, by the time the smaller initial conditions reach its size, their error would be at least as large.

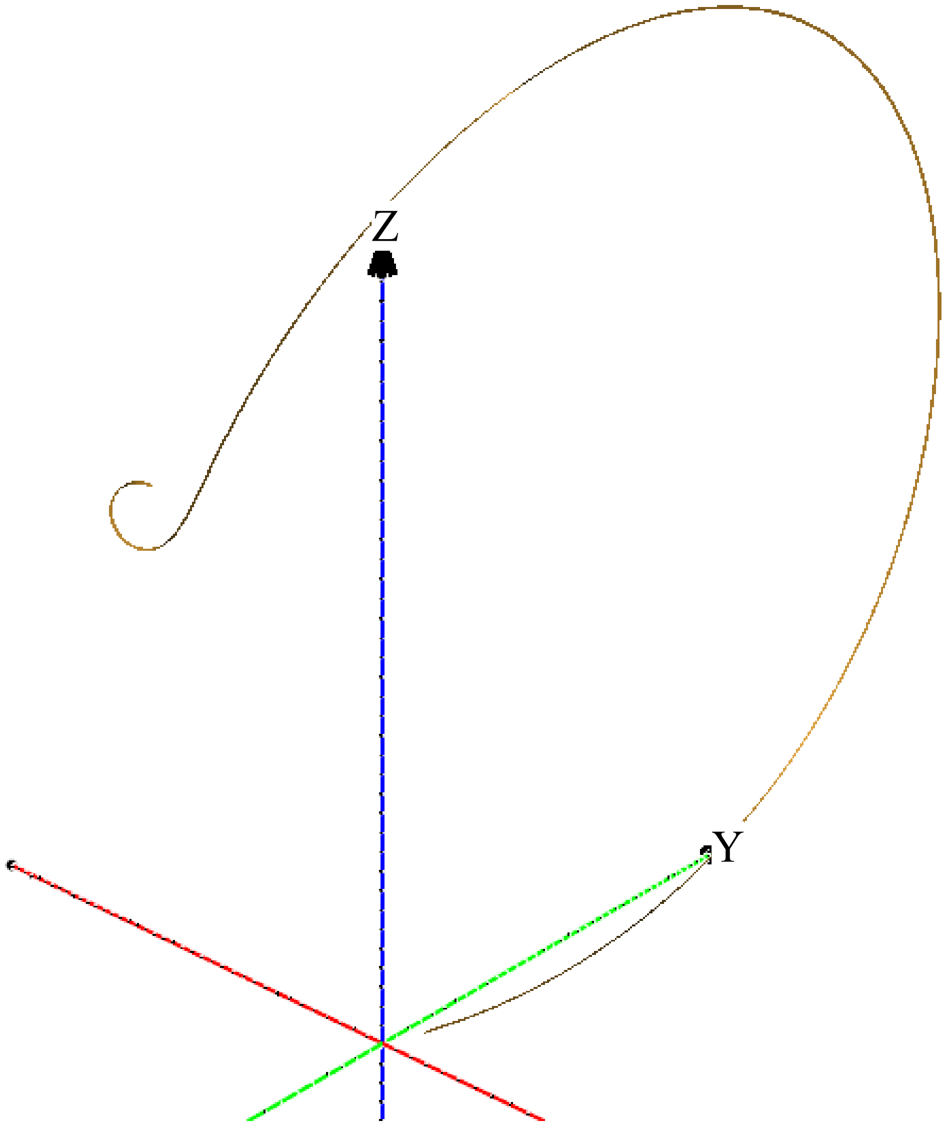


Figure 5.7: The local unstable manifold of the origin in the Lorenz system propagated for time $t = 1$. The error of the manifold shown is less than $2 \cdot 10^{-11}$.

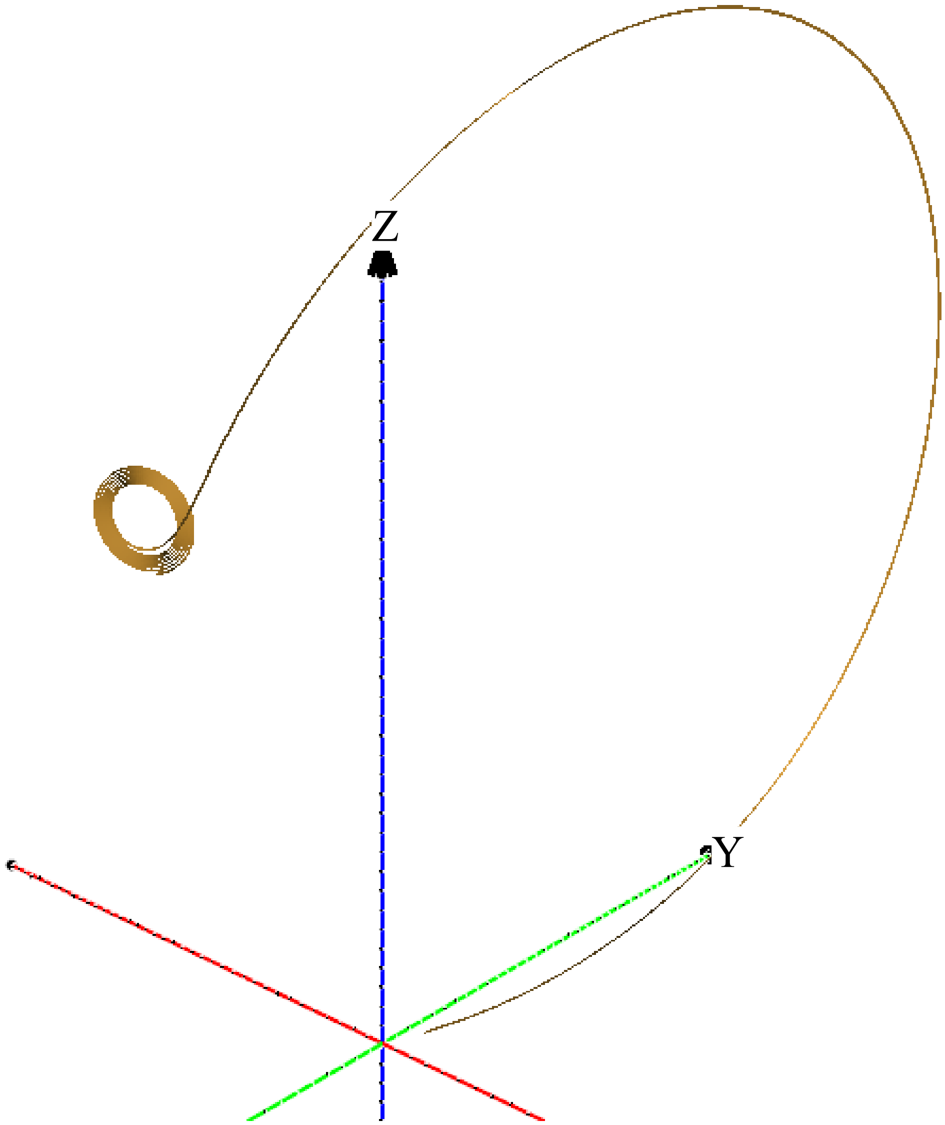


Figure 5.8: The local unstable manifold of the origin in the Lorenz system propagated for time $t = 5$. The error of the manifold shown is less than $2 \cdot 10^{-10}$.

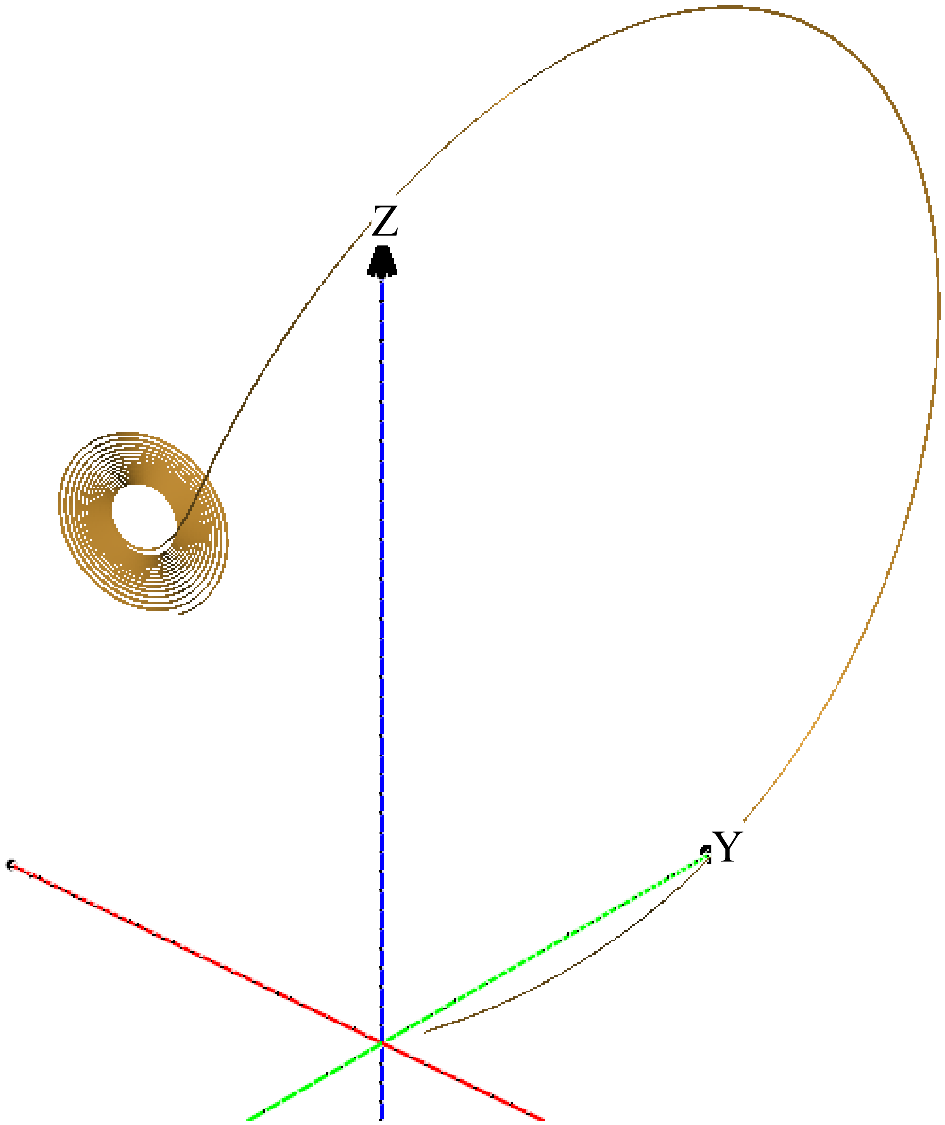


Figure 5.9: The local unstable manifold of the origin in the Lorenz system propagated for time $t = 10$. The error of the manifold shown is less than $2 \cdot 10^{-9}$.

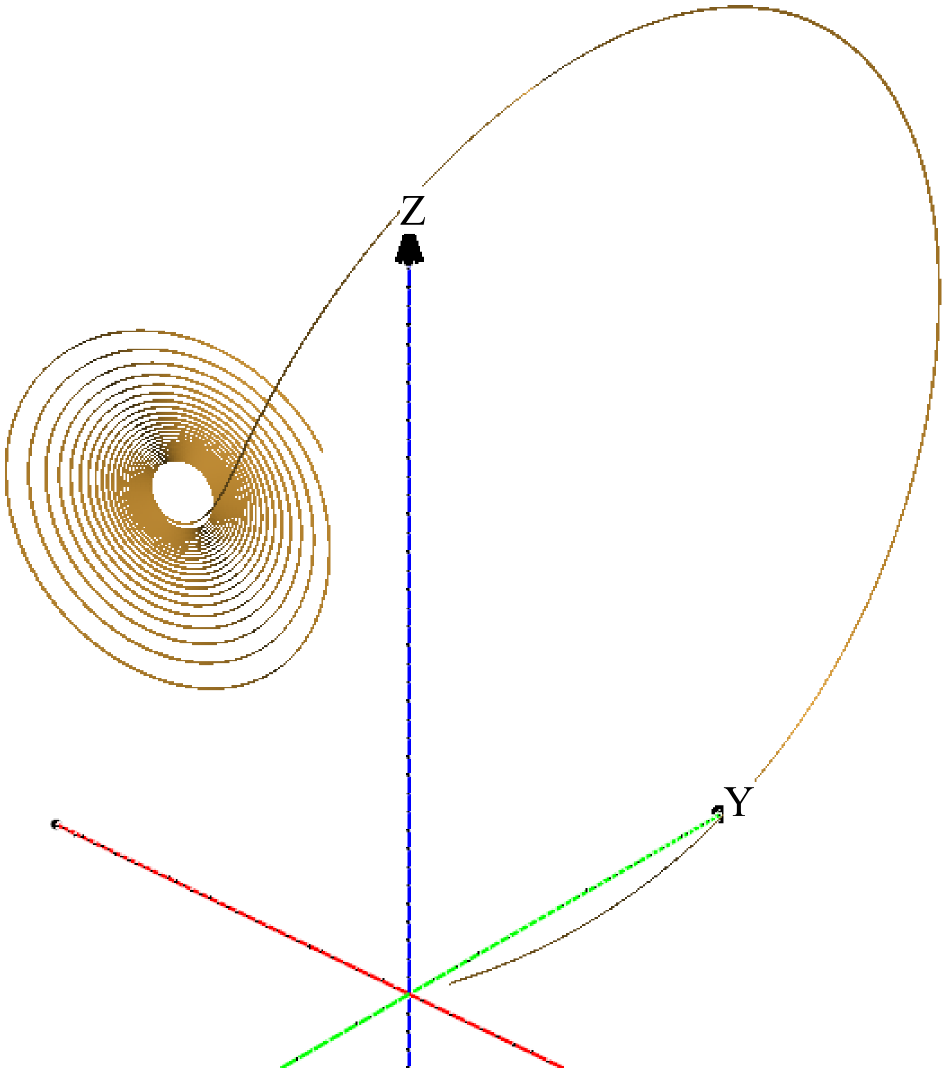


Figure 5.10: The local unstable manifold of the origin in the Lorenz system propagated for time $t = 15$. The error of the manifold shown is less than $7 \cdot 10^{-8}$.

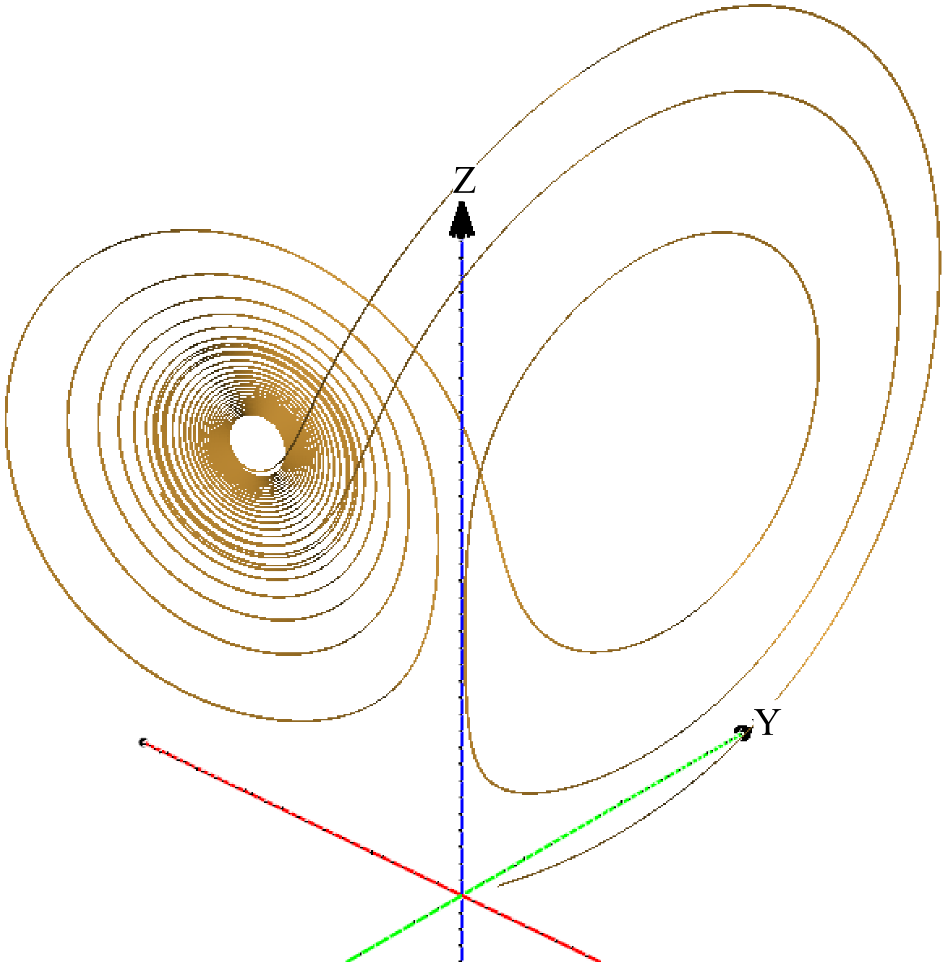


Figure 5.11: The local unstable manifold of the origin in the Lorenz system propagated for time $t = 20$. The error of the manifold shown is less than $5 \cdot 10^{-5}$.

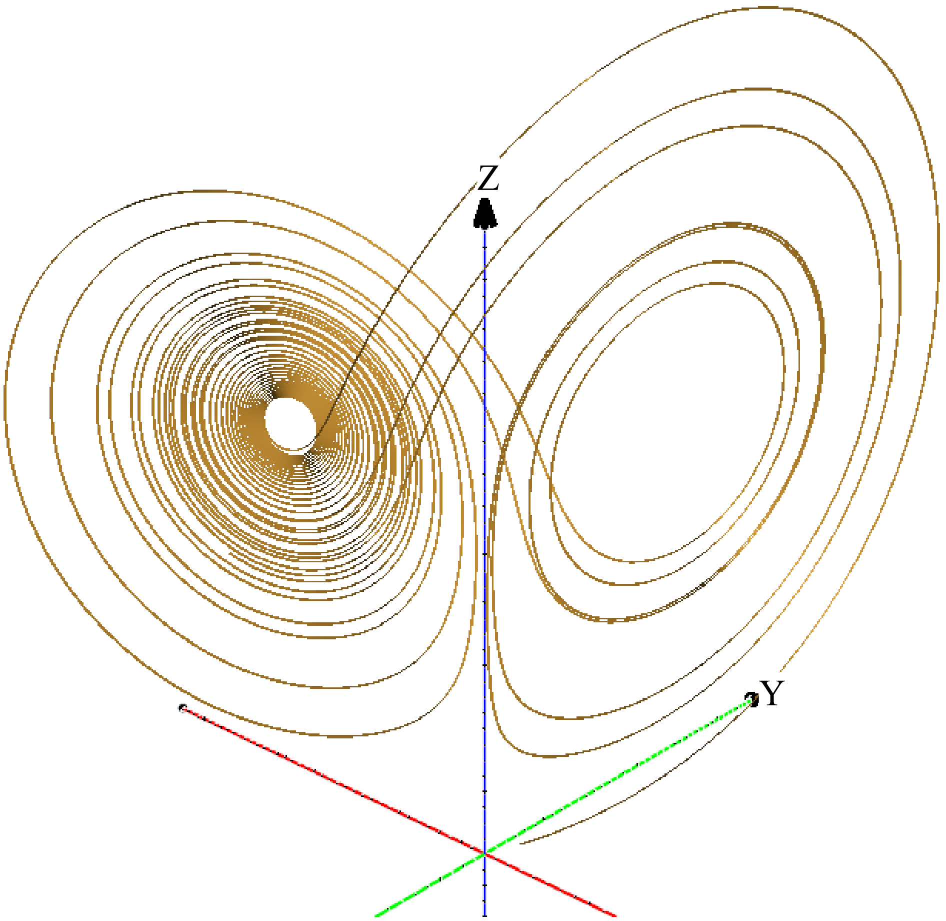


Figure 5.12: The local unstable manifold of the origin in the Lorenz system propagated for time $t = 28$. The error of the manifold shown is less than $2 \cdot 10^{-2}$.

t	0.2	0.3	0.4	0.5	0.6
Approximate Area A	2360	6307	8112	10192	14862
Maximum Error e_{max}	$3.7 \cdot 10^{-7}$	$3.7 \cdot 10^{-7}$	$3.7 \cdot 10^{-7}$	$1.5 \cdot 10^{-6}$	$7.9 \cdot 10^{-6}$
Volume V	$8.7 \cdot 10^{-4}$	$2.3 \cdot 10^{-3}$	$3.0 \cdot 10^{-3}$	$1.5 \cdot 10^{-2}$	$1.2 \cdot 10^{-1}$
# of Taylor Models	70	145	233	473	2469
# of Intervals	$1.7 \cdot 10^{16}$	$4.6 \cdot 10^{16}$	$5.9 \cdot 10^{16}$	$4.5 \cdot 10^{15}$	$2.4 \cdot 10^{14}$

Table 5.3: Approximate area, maximum error, volume, and number of Taylor Models covering the integrated half of the stable manifold within the box $B = [-50, 50] \times [-50, 50] \times [-20, 90]$ after propagating for the given time t backwards through the Lorenz.

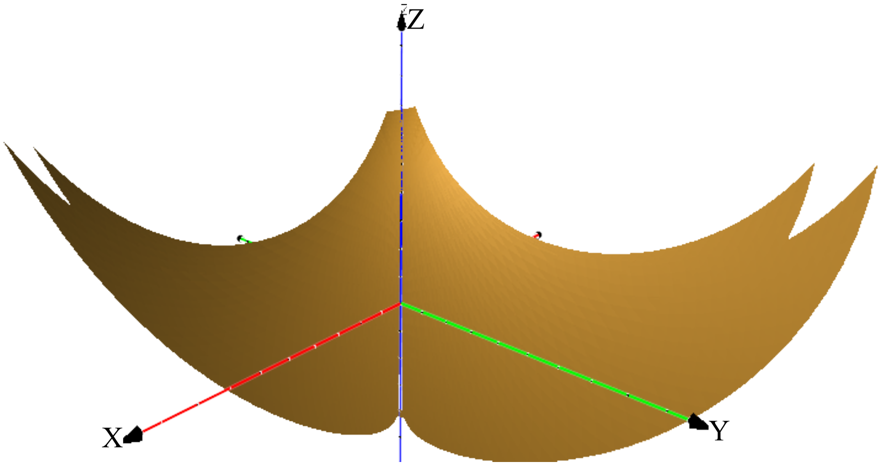


Figure 5.13: The local stable manifold of the origin in the Lorenz system within the box $[-50, 50] \times [-50, 50] \times [-20, 90]$ propagated for time $t = 0.2$. The error of the manifold shown is less than $4 \cdot 10^{-7}$.

Furthermore, due to the symmetry of the Lorenz equations, it is again sufficient to compute the manifold for only one half of the initial condition in the $x \geqslant 0$ quadrant, the other half is obtained by the simple reflection $R(x, y, z) = (-x - y, z)$.

Figures 5.13 through 5.17 show the entire manifold, including the mirror image of the computed pieces for the same time steps. The rugged, sawtooth like edges in these pictures are due to the way manifold pieces are discarded. Since the box B is not aligned with the direction of the eigenvectors, the direction of motion of each flow expansion happens at an

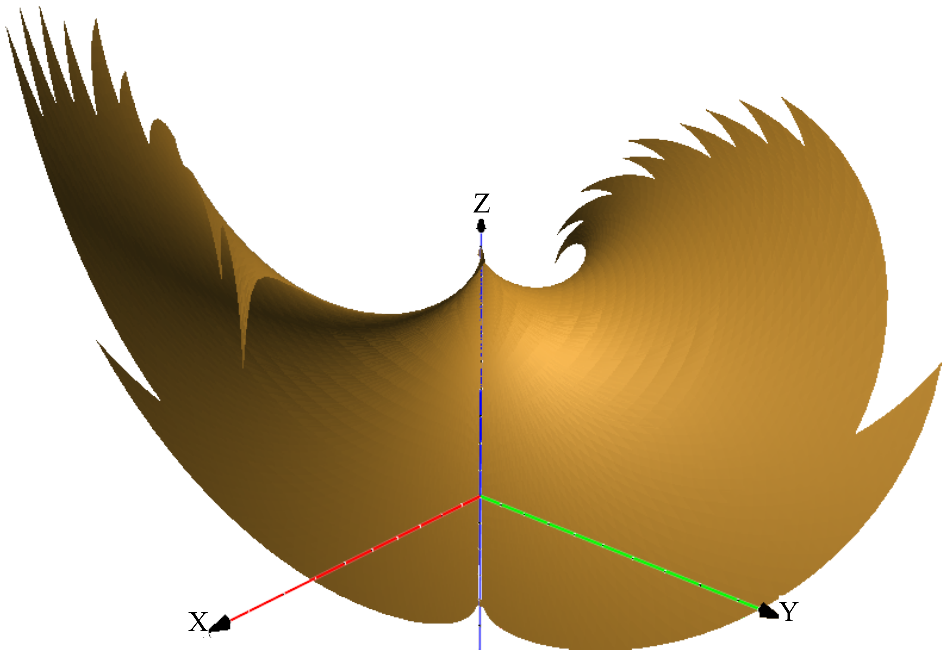


Figure 5.14: The local stable manifold of the origin in the Lorenz system within the box $[-50, 50] \times [-50, 50] \times [-20, 90]$ propagated for time $t = 0.3$. The error of the manifold shown is less than $4 \cdot 10^{-7}$.

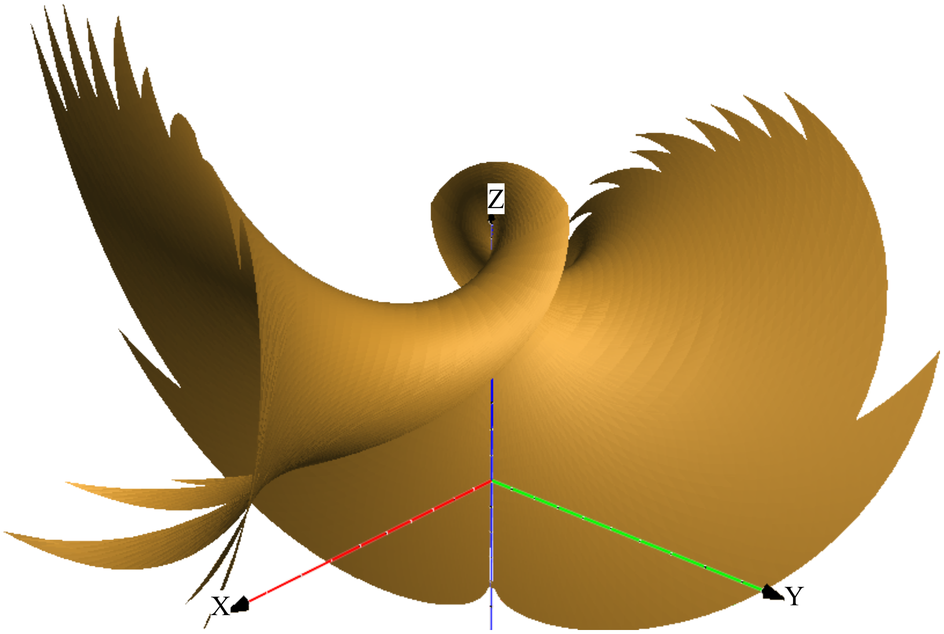


Figure 5.15: The local stable manifold of the origin in the Lorenz system within the box $[-50, 50] \times [-50, 50] \times [-20, 90]$ propagated for time $t = 0.4$. The error of the manifold shown is less than $4 \cdot 10^{-7}$.

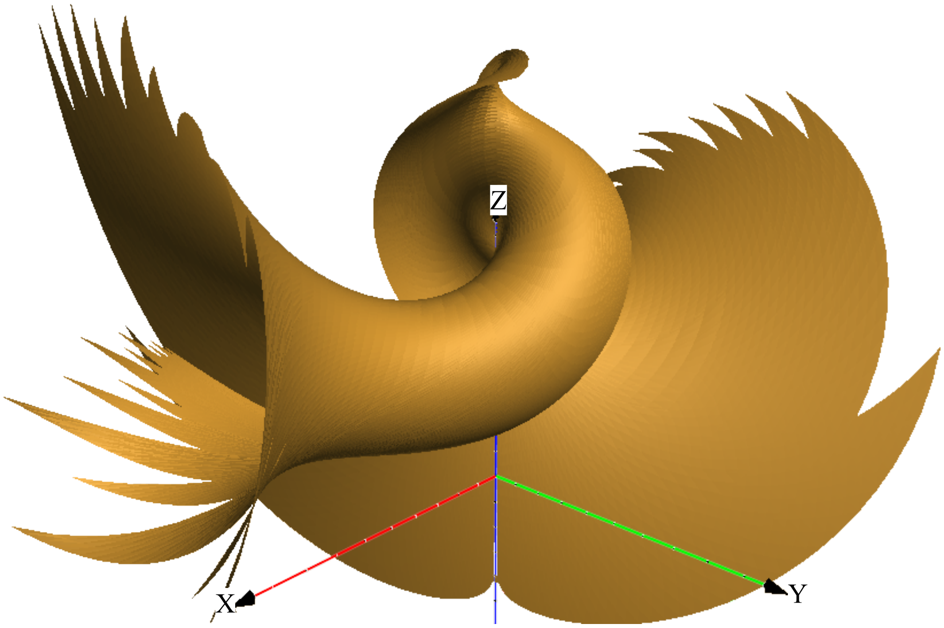


Figure 5.16: The local stable manifold of the origin in the Lorenz system within the box $[-50,50] \times [-50,50] \times [-20,90]$ propagated for time $t = 0.5$. The error of the manifold shown is less than $2 \cdot 10^{-6}$.

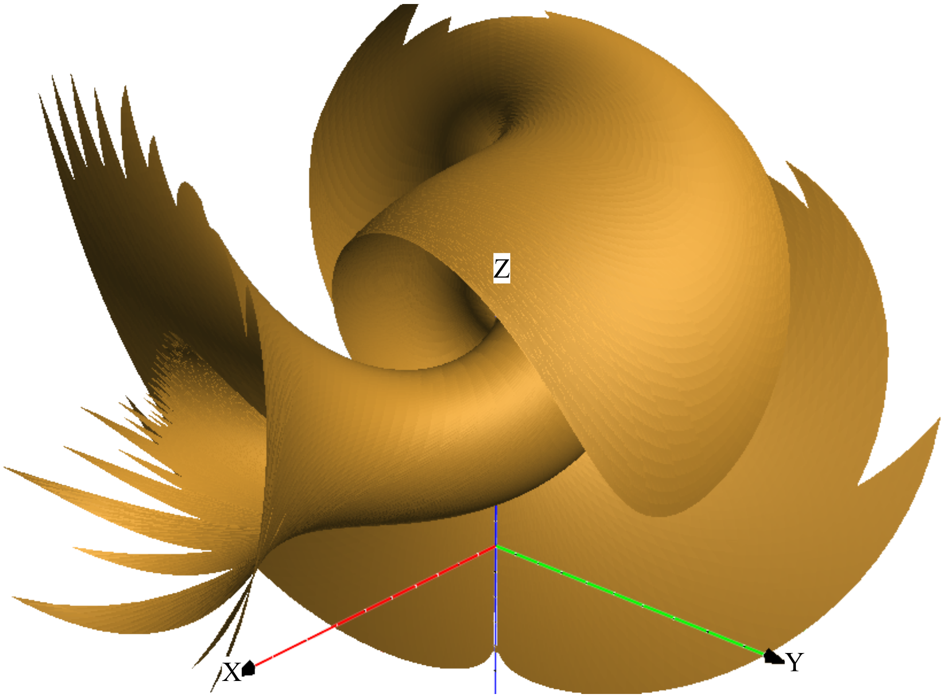


Figure 5.17: The local stable manifold of the origin in the Lorenz system within the box $[-50,50] \times [-50,50] \times [-20,90]$ propagated for time $t = 0.6$. The error of the manifold shown is less than $8 \cdot 10^{-6}$.

angle. Only flow expansions completely outside the box B are discarded, hence some pieces still have a corner inside B, while others are already completely outside.

Table 5.3 shows the numerical results of the integration of the boundary of the initial stable manifold enclosure for $\eta_2 = 0.2$ and $\eta_3 = 20$ backward in time for various times t. All statistics given in this table refer to the half of the manifold actually integrated, and do not include the mirror image making up the other half.

The maximum error e_{max} comprises both the error due to the integration, as well as the error due to the thickening of the initial manifold. The volume V given in this table is computed as $V = e_{max} \cdot A$. The number of Taylor Models indicates how many flow expansions were needed to cover the entire surface. Finally, the number of intervals is an estimate of how many interval boxes would be required to cover the same manifold with the accuracy of the Taylor Models. This number is estimated as the number of interval boxes of size e_{max} required to cover the volume V (i.e. $V/e_{max}^3 = A/e_{max}^2$).

The reason why the maximum error does not change for times 0.2 through 0.4 is because the maximum error comprises two components. One is the error accumulating during the numerical integration, the other is the error due to the thickening of the initial manifold. To the verified integrator, however, the thickening is not an error but just an extra dimension of the initial condition.

In the beginning, the error due to the thickening of the initial condition dominates the maximum error. This thickening grows differently in different places along the manifold, the fastest growth occurring where the manifold itself grows the fastest. Thus right in the beginning, this error grows quickly, but then stops as the manifold pieces with the largest error are discarded as they leave the box B. With those "offenders" discarded, the remaining pieces have much smaller thickness and thus do not contribute to the maximum error for quite a while. This shows that the maximum error given in this table is actually a rather pessimistic number, for the majority of the manifold the error is significantly less than this number. Eventually, the integration errors grow as the integration time increases and they become the dominating effect at time 0.5 and beyond, causing the maximum overall error to grow.

As can be seen, integration can be performed up to time $t = 0.5$ easily. The verified integrator is able to reach time $t = 0.6$. At this time, however, the current implementation of the automated domain decomposition has split the manifold into so many pieces that the number of Taylor Models covering the manifold has grown too large and further integration becomes impractical. Most likely this effect can be greatly reduced by an improved implementation of the automated domain decomposition, including the use of High Precision Taylor Models to make splitting decisions, allowing integration of the manifold beyond $t = 0.6$.

The comparison between intervals and Taylor Models is striking. As described in [26], one of the techniques to compute invariant manifolds that can be done in a verified setting is an interval based space pruning technique. However, in order to attain the same accuracy of representation as our Taylor Model covering, enormous numbers of interval boxes would be required just to store the representation. Considering that each interval box in three dimensions is made up of six double precision numbers of eight bytes each, just the total memory required to keep all these interval boxes is on the order of 10^{18} bytes or about 1 exabyte. Clearly this is practically impossible at this point in time, and this does not even consider yet the process to compute such an interval enclosure in the first place.

The Taylor Model representation, on the other hand, requires less than 2500 Taylor Models at most. Each Taylor Model as used by COSY VI has a maximum of 15714 coefficients, resulting in a maximum memory requirement of $15714 \cdot 2500 \cdot 8 \approx 314 \cdot 10^6$ bytes which is about 314 megabytes for the manifold representation. In practice, this number is even much lower, since the Taylor Models used are only sparsely populated.

To conclude our examination of the invariant manifolds of the Lorenz at the origin, figure 5.18 shows both the stable and unstable manifolds of the Lorenz system in the same picture. Unfortunately, we were unable to draw the stable and unstable manifold in different colors in this picture due to limitations in the Meshlab program used to display these pictures.

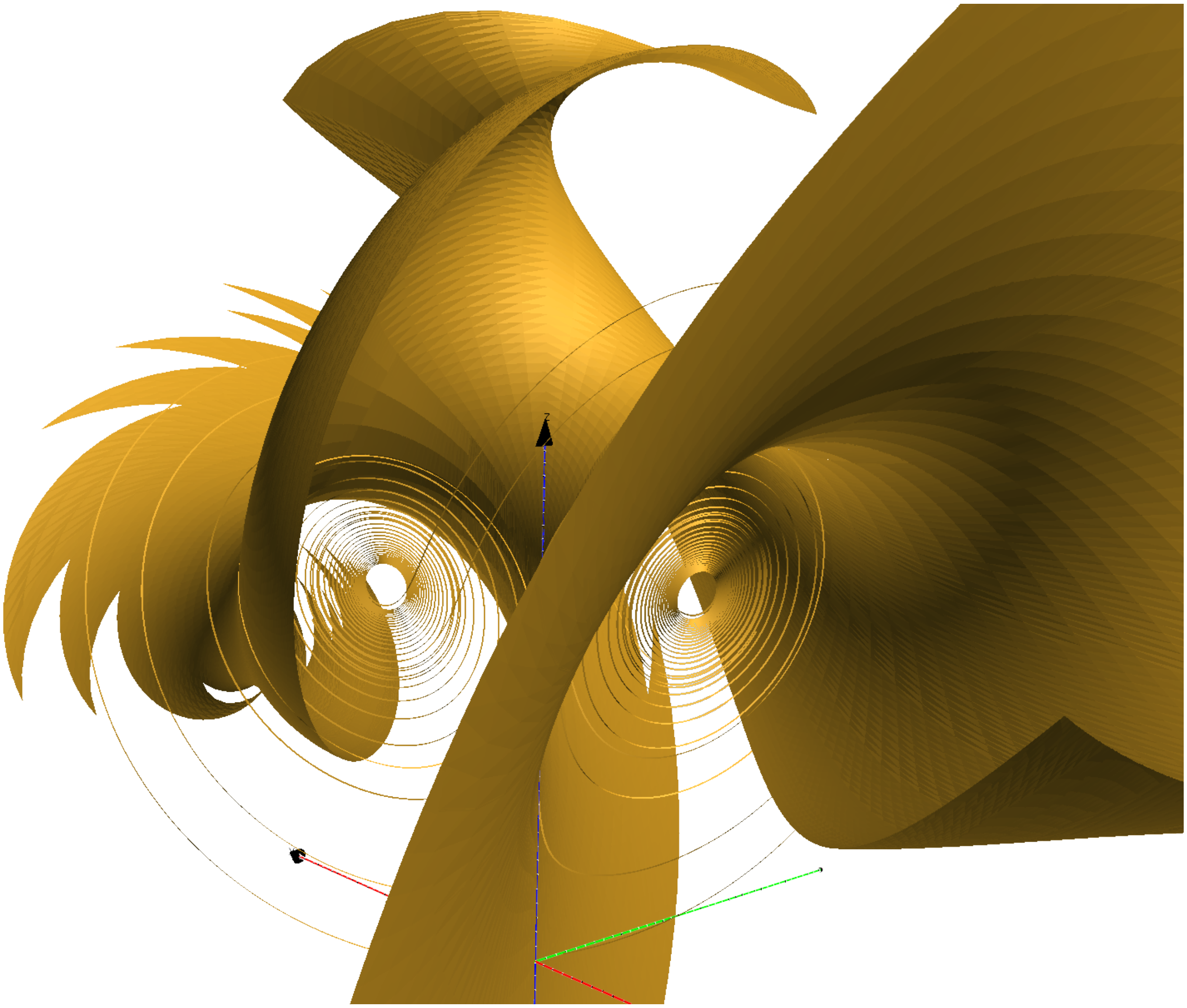


Figure 5.18: Both the local stable and unstable manifold of the origin in the Lorenz system within the box $[-50, 50] \times [-50, 50] \times [-20, 90]$ in the same picture. The error on both manifolds shown is less than $5 \cdot 10^{-5}$.

APPENDIX

APPENDIX

Compiler Options

The algorithms described in this dissertation, especially those in Chapter 3, rely heavily on correct implementation of the IEEE 754 floating point standard[2]. While modern computer hardware in general implements double precision operations according to this standard, care must be taken that the code written is compiled into the correct sequence of processor instructions.

Unfortunately, many compilers perform optimization on the code written and sometimes reorder operations. While often this results in faster execution time, reordering floating point operations in the Dekker operations breaks these operations, as they rely on subtle round-off effects of floating point numbers that are not taken into consideration by optimizing compilers.

Our implementation is written in FORTRAN, and we use the Intel Fortran Compiler version 12. While this compiler performs aggressive optimization, it provides a command line switch to turn off any floating point optimizations. The command line we use to compile the COSY INFINITY environment is “-fp-model strict -m32”.

This command line appears to be safe. A detailed analysis and comparison of various compilers performed by us can be found in [22].

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] *DoD supplement to ANSI X3.9-1978*, Tech. report, US Department of Defense, 1978.

[2] *IEEE standard for binary floating-point arithmetic.*, Tech. Report IEEE Std 754-1985, The Institute of Electrical and Electronics Engineers, 1985.

[3] S. Banach, *Sur les opérations dans les ensembles abstraits et leurs application aux équations intégrales*, Fund. Math. **3** (1922), 7–33.

[4] M. Berz, *Forward algorithms for high orders and many variables*, Automatic Differentiation of Algorithms: Theory, Implementation and Application **SIAM** (1991).

[5] ______, *High-order computation and normal form analysis of repetitive systems, in: M. month (ed), physics of particle accelerators*, vol. 249, p. 456, American Institute of Physics, New York, 1991.

[6] ______, *Differential algebraic formulation of normal form theory*, M. Berz, S. Martin and K. Ziegler (Eds.), Proc. Nonlinear Effects in Accelerators (London), IOP Publishing, 1992, p. 77.

[7] ______, *From Taylor series to Taylor models*, AIP CP **405** (1997), 1–20.

[8] ______, *Modern map methods in particle beam physics*, Academic Press, San Diego, 1999, Also available at http://bt.pa.msu.edu/pub.

[9] M. Berz and G. Hoffstätter, *Computation and application of Taylor polynomials with interval remainder bounds*, Reliable Computing **4** (1998), no. 1, 83–97.

[10] M. Berz and K. Makino, *Suppression of the wrapping effect by Taylor model- based verified integrators: Long-term stabilization by shrink wrapping*, International Journal of Differential Equations and Applications **10,4** (2005), 385–403.

[11] K. Braune and W. Kramer, *High-accuracy standard functions for intervals*, Computer Systems: Performance and Simulation (Manfred Ruschitzka, ed.), vol. 2, Elsevier Science Publishers B.V., 1986, pp. 341–347.

[12] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig, *Taschenbuch der Mathematik*, 5th edition ed., Verlag Harri Deutsch, 2001.

[13] L.E.J. Brouwer, *Über eineindeutige, stetige Transformationen von Flächen in sich*, Math. Ann. **69** (1910), 176–180.

[14] X. Cabré, E. Fontich, and R. de la Llave, *The parameterization method for invariant manifolds I: Manifolds associated to non-resonant subspaces*, Software for Railways Control and Protection Systems. EN 50128, 1995.

[15] J. B. Conway, *Functional analysis*, Springer, 1990.

[16] T.J. Dekker, *A floating-point technique for extending the available precision*, Numerische Mathematik **18** (1971/72), 224–242.

[17] Paolo Cignoni et al., *Meshlab*, Tech. report, Visual Computing Lab - ISTI - CNR, 2011.

[18] David Goldberg, *What every computer scientist should know about floating-point arithmetic*, ACM Comput. Surv. **23** (1991), no. 1, 5–48.

[19] Johannes Grote, *High-order computer-assisted estimates of topological entropy*, Ph.D. thesis, Michigan State University, East Lansing, Michigan, USA, 2009.

[20] M. Hazewinkel (ed.), *Encyclopaedia of mathematics*, Springer Verlag, 2002.

[21] M. Hénon, *A two-dimensional mapping with strange attractor*, Communications in Mathematical Physics **50** (1976), no. 1, 69–77.

[22] R. Jagasia and A. Wittig, *Survey of fortran compiler options and their impact on cosy infinity*, Tech. Report MSUHEP-090422, Michigan State University, 2009.

[23] W. Kahan, *Pracniques: further remarks on reducing truncation errors*, Commun. ACM **8** (1965), no. 1, 40.

[24] A. Katok and B. Hasselblatt, *Introduction to the modern theory of dynamical systems*, Cambridge University Press, Cambridge, 1996.

[25] D. E. Knuth, *The art of computer programming*, vol. I-III, Addison Wesley, Reading, MA, 1973.

[26] B Krauskopf, H M Osinga, E J Doedel, M E Henderson, J Guckenheimer, A Vladimirsky, M Dellnitz, and O Junge, *A survey of methods for computing (un)stable manifolds of vector fields*, International Journal of Bifurcation and Chaos **15** (2005), no. 3, 763–791.

[27] E. N. Lorenz, *Deterministic nonperiodic flow*, J. Atmos. Sci. **20** (1963), 130–141.

[28] K. Makino, *Rigorous analysis of nonlinear motion in particle accelerators*, Ph.D. thesis, Michigan State University, East Lansing, Michigan, USA, 1998, Also MSUCL-1093.

[29] K. Makino and M. Berz, *Remainder differential algebras and their applications*, Computational Differentiation: Techniques, Applications, and Tools (Philadelphia) (M. Berz, C. Bischof, G. Corliss, and A. Griewank, eds.), SIAM, 1996, pp. 63–74.

[30] ______, *Efficient control of the dependency problem based on Taylor model methods*, Reliable Computing **5** (1999), no. 1, 3–12.

[31] ______, *Taylor models and other validated functional inclusion methods*, International Journal of Pure and Applied Mathematics **6,3** (2003), 239–316.

[32] ______, *COSY INFINITY version 9*, Nuclear Instruments and Methods **558** (2005), 346–350.

[33] ______, *Range bounding for global optimization with Taylor models*, Transactions on Computers **4,11** (2005), 1611–1618.

[34] ______, *Suppression of the wrapping effect by Taylor model- based verified integrators: Long-term stabilization by preconditioning*, International Journal of Differential Equations and Applications **10,4** (2005), 353–384.

[35] ______, *Suppression of the wrapping effect by Taylor model- based verified integrators: The single step*, International Journal of Pure and Applied Mathematics **36,2** (2006), 175–197.

[36] Kyoko Makino, Private communication.

[37] R. E. Moore, *Automatic local coordinate transformation to reduce the growth of error bounds in interval computation of solutions of ordinary differentical equations*, Error in Digital Computation, Vol II (L. B. Rall, ed.), 1965.

[38] ______, *Interval analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1966.

[39] S. Newhouse, Private communication.

[40] N. Revol, *Multiple precision floating-point interval library*, Tech. report, SPACES, INRIA Lorraine and Arenaire, INRIA Rhone-Alpes, 2002.

[41] J. R. Shewchuk, *Adaptive precision floating-point arithmetic and fast robust geometric predicates*, Discrete & Computational Geometry **18** (1997), 305–363.

[42] P. Snopok, *Optimization of accelerator parameters using normal form methods on high-order transfer maps*, Ph.D. thesis, Michigan State University, East Lansing, Michigan, USA, 2007.

[43] N. Wiener, *Cybernetics: or control and communication in the animal and the machine*, MIT Press, 1948.

[44] A. Wittig and M. Berz, *Computation of high-order maps to multiple machine precision*, International Journal Modern Physics A **24** (2008), no. 5, 1019–1039.

[45] ______, *Rigorous high precision interval arithmetic in COSY INFINITY*, Communications of the Fields Institute **in print** (2008).

[46] ______, *High period fixed points, stable and unstable manifolds, and chaos in accelerator transfer maps*, Vestinik Mathematica **in preparation** (2010).

[47] A. Wittig, M. Berz, J. Grote, K. Makino, and S. Newhouse, *Rigorous and accurate enclosure of invariant manifolds on surfaces*, Regular and Chaotic Dynamics **15** (2010), no. 2-3, 107–126.

[48] A. Wittig, M. Berz, and S. Newhouse, *Computer assisted proof of the existence of high period periodic points*, Communications of the Fields Institute **in print** (2008).