Untersuchungen zur Hénon Iteration

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

Universität Trier
FB IV - Informatikwissenschaften
Professur Arithmetische Algorithmen

Erstgutachter:	Prof. Dr. Norbert Müller
Zweitgutachter:	Prof Dr. Stephan Näher

Vorgelegt am xx.xx.xxxx von:

Benedikt Lüken-Winkels
Baltzstraße 6
54296 Trier
b.lueken.winkels@gmail.com
Matr.-Nr. 1138844

Zusammenfassung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

Das Rechnen mit reellen Zahlen und Funktionen auf denselben bedeutet die Verarbeitung einer unendlichen Menge an Informationen, die in der computergestützten Praxis lediglich mit einer endlichen Approximation beschrieben werden können [BrHW08]. Eine solche Beschränkung hat zu Folge, dass arithmetische Operationen keine exakten Ergebnisse ohne Fehler liefern. Dieser wird in der Disziplin der *Exakten Reellen Arithmetik* durch Intervalle abgeschätzt, die den tatsächlichen Wert mitsamt Rundungsfehler umschließen und so ein korrektes Ergebnis garantieren können. Eine entsprechende Implementierung bietet die C++-Softwarebibliothek `iRRAM` [Müll09], die reelle Zahlen zu einer beliebigen Genauigkeit, statt der häufig verwendetet *double*-Präzision, annähert. Jedoch entsteht durch die so betriebene Intervallarithmetik das Problem der Überschätzung des Wertebereichts, zum Beispiel beim mehrfachen Auftreten derselben Variable in einem Term. Ein weit verbreiteter Lösungsansatz ist die Verwendung von Taylormodellen, mit denen Zahlen als höherdimensionale Polynome mit reellen Koeffizienten und einem Fehlerintervall dargestellt werden [BrKM15] [MaBe01]. Zudem speichern Variablen ('Fehlersymbole') die Rechenfehler und deren Abhängigkeitsinformationen. Eine leicht veränderte Version der Taylormodelle wird im C++-Programm `Tangentspace`[1] implementiert. Die Taylormodelle bestehen hier aus linearen Polynomen mit beliebigen Intervallkoeffizienten und rationalen Endpunkten. Die Taylomodelle werden verwendet, um lineare Schranken für nichtlineare Funktionen zu bestimmen. Eine weitere Implementierung von Brauße et al. [BrKM15] verwendet reelle Endpunkte für die Intervallkoeffizienten und die Intervalle der Fehlersymbole. Das im Rahmen dieser Arbeit enstandene C++-Programm `HOTM` (*High Order Taylor Model*) bildet eine Kombination aus diesen beiden Herangehensweisen, da Intervalle mit beliebigen reellen Endpunkten verwendet werden.

Zielsetzung

Ziel dieser Arbeit war die Untersuchung nichtlinearer Taylormodelle in der praktischen Anwendung. Hierzu wurde das C++-Programm `HOTM` (*High Order Taylor Model*), beziehungsweise der gleichnamige Zahlentyp implementiert. Die Taylormodelle bestehen aus Polynomen mit Intervallkoeffizienten, welche die reellen Zahlen der `iRRAM`-Software-Bibilothek als Endpunkte verwenden. Mit den so definierten Taylormodellen und damit einer mehrstufigen Intervallarchitektur ergibt sich eine Vielzahl an Konfigurationsmöglichkeiten von der Breite der Endpunkte der Intervalle, bis hin zu abstrakteren Operationen auf den Taylormodellen. An Hand von Berechnungen der Hénon-Abblildung werden diese evaluiert und versucht, eine möglichst optimale Einstellung für die Parameter der `HOTM` im zweidimensionalen Raum ($\mathbb{R}^2$) zu finden.

[1] http://informatik.uni-trier.de/~mueller/Research/Tangentspace/ (Stand 29.03.2021)

Gliederung

Als praktisch ausgelegt Arbeit liegt der Schwerpunkt bei der Implementierung und den damit erzeugten experimentellen Ergebnissen. Die Basis hierfür bilden die in Kapitel 2 beschriebenen Grundlagen, welche sich in Hénon-Abbildung, Taylormodelle und Lyapunov Exponenten gliedern. Diese Überblicke sollen das Verständnis über Entscheidungen in der Implementierung, die Deutung der Grafiken und der Ergebnisse vereinfachen. In Kapitel 3 werden die implementierten Operationen und Algorithmen der `HOTM` beschrieben. Mit der Anwerndung bei der Hénon-Abbildung werden diese in Kapitel 4 evaluiert und verschiedenene Konfigurationen verglichen. Während der Implementierung und Durchführung der Experimente sind weiterführende Ideen und Anwendungsmöglichkeiten für die `HOTM` entstanden, auf die im Ausblick in Kapitel 5 neben der Diskussion der Ergebnisse eingegangen wird.

Relevante Literatur

Die für diese Arbeit relevante Literatur lässt sich in vier Kategorien aufteilen, vergleichbar mit den später eingeführten Abstraktionsebenen, die sich aus dem Aufbau der `HOTM` ergeben. Für die grundlegende Zahlendarstellung wir die Implementierung der reellen Zahlen der `iRRAM` von Müller verwendet. Funktionsweisen und mögliche Anpassungen der Software-Bibilothek können den Quellen [Müll09] und [Müll00] entnommen werden. Für eine theoretische Grundlage der hier praktisch angewendeten Berechenbaren Analysis wird auf ein Tutorial von Brattka et al. [BrHW08] verwiesen. Spandl untersucht in [Span10] mit Hilfe der `iRRAM` den Lyapunov Exponenten für dynamische Systeme und zieht dabei Intervallmethoden heran, was einen zentralen Teil dieser Arbeit darstellt.

Da die `HOTM` auf der nächst höheren Ebene aus Polynomen mit Intervallkoeffizienten bestehen, wurde eine allgemeine Intervallarithmetik, wie im Standardwerk von Moore [Moor79] beschrieben implementiert. Yan et al. führen in [Yan98] einen Algorithmus ein, der die Addition von Polynomen effizient gestaltet, welcher in [MoPe07] für die Multiplikation formuliert ist. Mit dieser Basis können die Taylormodelle, wie in [MaBe01] und [BrKM15] beschrieben, implementiert werden.

In [BrKM15] sind zudem weiterführende Operationen auf Taylormodellen zu finden. Diese werden unter anderem im Programm `Tangentspace`, auf dem die `HOTM`s aufbauen, verwendet und ermöglicht geometrische Lösungen für Konfliktgesteuerte Gleichungslöser. Die Funktionsweise der so implementierten Taylormodelle wird an Hand der Hénon-Abblildung untersucht. Hierzu existieren neben dem ursprünglichen Paper von M. Hénon [Hén76], viele Experimente und Untersuchungen, wie Oliviera et al., die sich in [MdOCC20] mit verschiedenen Parameterbelegungen beschäftigen.

2. Grundlagen

2.1 Hénon-Abbildung

Die Hénon-Abbildung bietet eine Möglichkeit, das Verhalten eines seltsamen Attraktors (*engl. 'strange attractor'*), wie dem des Lorenz Systems, mit einer vergleichbar einfachen Abbildung zu untersuchen [Hén76].

Konvergieren die Werte einer Funktion in einem bestimmten Wertebereich R, der Fangzone (*engl. 'trapping zone'*), gegen einen Punkt oder eine Kurve, spricht man von einem Attraktor. Dieser Attraktor kann jedoch auch eine komplexere Struktur haben. Auf einem solchen seltsamen Attraktor springen die Werte hin und her und reagieren hochempfindlich auf Änderungen der Initialbedingungen. Dieses Verhalten lässt mit der Abbildung $x_{i+1} = y_i + 1 - ax_i^2, y_{i+1} = bx_i$, beziehungsweise

$$f(x, y) = (y + 1 - ax^2, bx)$$

beobachten. f erfüllt dieselben Kriterien, wie der Lorenz-Attraktor, der durch eine dreidimensionale Differenzialgleichung entsteht, allerdings wurde f so definiert, dass auch höhere Iterationszahlen leichter zu berechnen und zu analysieren sein sollen. Die Hénon-Abbildung bildet den $\mathbb{R}^2$ auf sich selbst ab. Dieser Vorgang besteht aus drei Schritten. Man betrachte eine Fläche entlang der x-Achse gelegen:

Dehnen und Falten

$$T' : x' = x, y' = 1 + y - ax^2$$

Mit dem Parameter a kann die Stärke der Biegung gesteuert werden.

Kontrahieren

$$T'' : x'' = b \cdot x, y'' = y'$$

Ein $|b| < 1$ bedeutet, dass sich die Fläche zusammen zieht. Wird b zu groß gewählt, so entsteht eine zu starke Kontraktion und der Attraktor ist schwerer erkennbar. Ist b zu klein, ist der Effekt zu gering und das Verhalten der Abbildung ist nicht mehr chaotisch.

Spiegeln

$$T''' : x''' = y'', y''' = x''$$

Im letzten Schritt werden die Achsen vertauscht und somit die Fläche gepiegelt.

Die entstehende Abbildung hat unter Anderem folgende Eigenschaften:

- Invertierbar: (x_{n+1}, y_{n+1}) kann eindeutig auf (x_n, y_n) zurückgeführt werden.
- Kontrahiert Flächen: Mit $|b| < 1$ werden Flächen kleiner.

- Besitzt eine Fangzone, die einen Attraktor enthält. Allerdings landen nicht immer alle Orbits in der Fangzone, da wegen der Quadrierung durch T' Terme bestimmter Größe nach ∞ laufen können.

Abbildung 2.1 zeigt die einzelnen Schritte einer Evolution der Hénon-Abbildung anhand eines Rechtecks, welches als nichtlineares Taylormodell definiert wurde.

$$x_0 = 0 + 1 \cdot \lambda_1 \quad (\lambda_1 \in [0 \pm 0.4])$$

$$y_0 = 0 + 1 \cdot \lambda_2 \quad (\lambda_2 \in [0 \pm 0.05])$$

Das Rechteck wird gedehnt und gefaltet, dann kontrahiert und zuletzt rotiert, beziehungsweise gespiegelt.

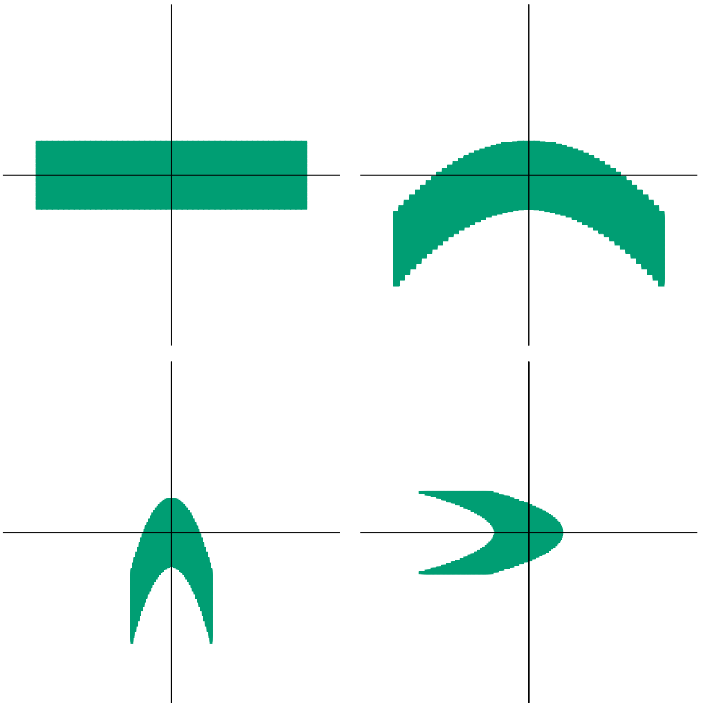


Abbildung 2.1: Einfache Evolution der Hénon Abbildung, aufgeteilt in das initiale Rechteck und die drei Zwischenschritte.

Für die Parameter $a = 1.4$ und $b = 0.3$ ergibt sich eine Fangzone in der sich die Funktionswerte auf einem seltsamen Attraktor bewegen, während die außerhalb gelegenen Punkte gegen unendlich laufen. Das bedeutet, dass ein Punkt, der in der Fangzone, beziehungsweise auf dem Attraktor liegt, wiederum auf diesen abgebildet wird. Der Attraktor ist in Abbildung 2.2 zu sehen. Hier wurden, ausgehend vom Punkt $(0, 0)$, 10000 Iterationen der Hénon-Abbildung berechnet und jeweils das Ergebnis eingezeichnet. Es ist deutlich erkennbar, dass sich der Attraktor teils nahe am Rande der Fangzone bewegt. Bereits bei einer leichten Überschätzung des Ergebnisses kann dies dazu führen, dass die Funktionswerte die Fangzone verlassen, auch wenn der tatsächliche Wert eigentlich in dieselbe abgebildet würde. Dies kommt zum Tragen, wenn Intervalle, als Zahlendarstellung gewählt werden, wie es bei `HOTM` der Fall ist, da diese immer auch einen Bereich um den Wert herum abdecken. Liegt dieser nun außerhalb der Fangzone, wächst der Bereich durch die Quadrierung in

jeder Iteration exponentiell und lässt somit keine aussagekräftigen Informationen ermitteln.

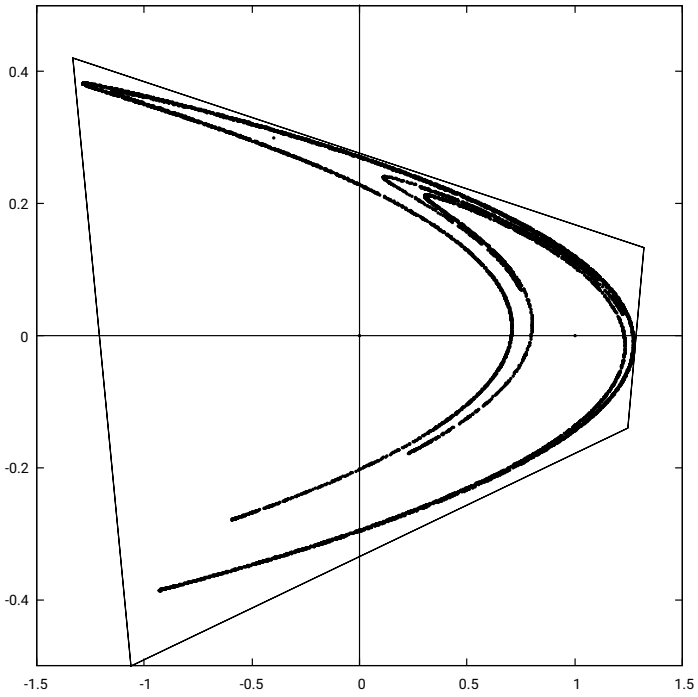


Abbildung 2.2: Seltsamer Attraktor der Hénon-Abbildung für $a = 1,4$ und $b = 0.3$ mit 10000 Punkten.

2.2 Taylormodelle

Ein Taylormodell im Sinne der Erweiterung von Brauße et al. in [BrKM15] des Grundmodells von Makino et al. in [MaBe01] besteht aus einem Polynom p mit $k \in \mathbb{N}$ Variablen und geschlossenen Intervallen als Koeffizienten. Das Monom c_0 des Grades 0, der das Kernintervall (*Kernel*) des Taylormodells darstellt, dient als Restintervall, das den Rechenfehler, über den keine Abhängigkeitsinformationen mehr verfügbar ist, enthält. Eine Variable oder ein Fehlersymbol λ_i aus dem Vektor $\lambda = (\lambda_1, \ldots, \lambda_k)$ steht für einen Wert aus dem dazugehörigen Supportintervall aus $S = (s_1, \ldots, s_k)$ mit $\lambda_i \in s_i$ und wird dazu verwendet, unbekannte Werte, Rechenungenauigkeiten und funktionale Abhängigkeiten innerhalb eines oder zwischen mehreren Taylormodellen abzubilden. Für die Variablen eines Monoms wird in dieser Arbeit wie in [BrKM15] eine Multiindexnotation verwendet. Für einen Index $n = (n_1, \ldots, n_k)$ ist das dazugehörige Produkt von Variablen definiert als $\lambda^n = \lambda_1^{n_1} \cdot \ldots \cdot \lambda_k^{n_k}$.

Die verwendeten Intervalle $c_n = [\tilde{c}_n \pm \varepsilon_n] \subseteq \mathbb{R}$ haben in `HOTM` reelle Endpunkte und stellen mit $c'_n = [\tilde{c}_n \pm 0]$ Punktintervalle dar. Mit einem so definierten Taylormodell $T = \Sigma_n c_n \lambda^n$ kann Exakte Reelle Arithmetik betrieben werden, indem Rundungsfehler und Ungenauigkeiten, die beim Rechnen mit endlicher Genauigkeit entstehen können als Intervalle in den Fehlersymbolen berücksichtigt werden. Dies ist zwar auch mit einfacher Intervallarithmetik möglich, jedoch leidet die Präzision des Ergebnisses einer solchen Berechnung stark unter der schnell wachsenden Überschätzung, die sich aus der Tendenz von Intervallen ergibt, bei jeder Rechenoperation zu wachsen.

In [BrKM15] werden drei Unterfamilien von Taylormodellen identifiziert, die sich in der Definition des Polynoms und dessen Koeffizienten unterschieden:

1. Affine Arithmetik: Polynome des Grades ≤ 1 mit Punktintervallen, außer beim Kernel.
2. Generalisierte Intervallarithmetik: Polynome des Grades ≤ 1 mit beliebigen Intervallen bei den Koeffizienten.
3. Klassische Taylormodelle: Polynome beliebigen Grades mit Punktintervallen, außer beim Kernel.

Das in dieser Arbeit verwendete Taylormodell ist eine Kombination aus 2. und 3., und besteht aus Polynomen beliebiger Ordnung mit beliebigen Intervallen als Koeffizienten. Dadurch können mit zwei solchen *nichtlinearen Taylormodellen* im Zweidimensionalen komplexere Strukturen, wie Kurven höherer Ordnung beschrieben werden.

Abbildung 2.3 zeigt die Darstellung linearer und nichtlinearer Taylormodelle für

$$\begin{aligned} x &= 0 + 1 \cdot \lambda_1 \quad (\lambda_1 \in [0 \pm 0.4]) \\ y &= 0 + 1 \cdot \lambda_2 \quad (\lambda_2 \in [0 \pm 0.1]) \end{aligned}$$

und deren Abbildungen durch $f(x, y) = (y+1-1.4x^2, 0.3x)$ mit einer Farbkodierung, die den Ursprung der abgebildeten Flächen indiziert.

Sowohl die linearen, als auch die nichlinearen Taylormodelle umschließen den korrekten Bereich der Funktionswerte, allerdings ist die abgebildete Fläche der Nichtlinearen näher an der Fläche, die sich ergäbe, betrachtete man das Rechteck als Menge Punkten und bildete sie einzeln ab. Es ergibt sich eine geringere Überschätzung, aber auch ein komplexeres Polynom.

2.2.1 Arithmetische Operationen auf Taylormodellen

Arithmetische Operationen auf Taylormodellen mit Intervallkoeffizienten bedeuten das Verrechnen von Polynomen, die wiederum Polynome ergeben. Für die Addition, Subtraktion und Multiplikation werden lediglich die entsprechenden Operationen auf die Polynome angewandt. Eine Divsion ist nur möglich, wenn der Divisor nicht die 0 enthält (siehe Kapitel 3.2).

2.2.2 Spezielle Operationen auf Taylormodellen

Um den durch Berechnungen wachsenden Grad des Polynomes und die Breite der Intervallkoeffizienten zu kontrollieren, wird in [BrKM15] *Sweeping* und *Splitting*, also Fegen und Teilen, vorgestellt.

Sweeping reduziert den Grad eines Monoms $c_n\lambda_i^k$ mit $\lambda_i \in s_i$, indem eines der Fehlersymbole durch das entsprechende Intervall aus S ersetzt wird:

$$c_n\lambda_i^k \rightsquigarrow c_n s_i \lambda_i^{k-1}$$

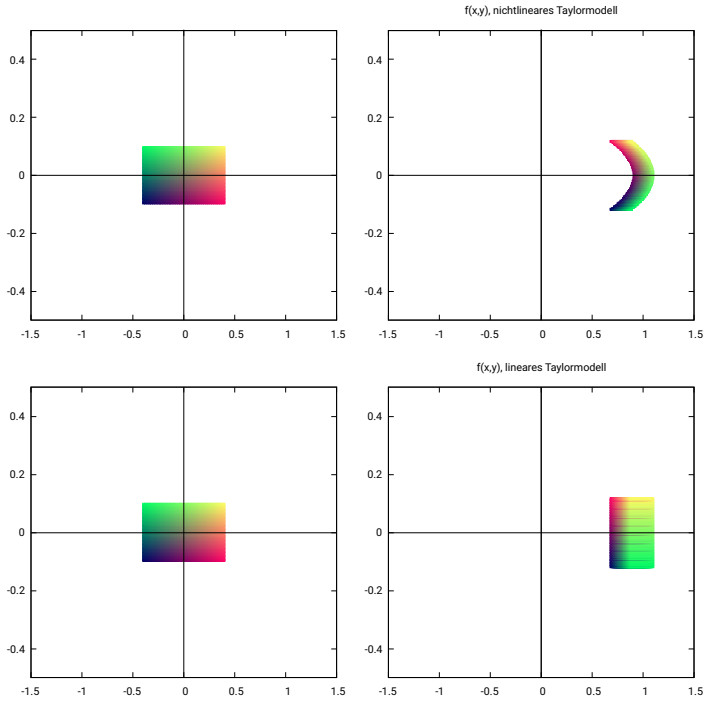


Abbildung 2.3: Darstellung nichtlinearer und linearer Taylormodelle, die ein Rechteck beschreiben und deren Abbildung durch die Funktion $f(x, y) = (y + 1 - 1.4x^2, 0.3x)$. Die Farbkodierung indiziert den Ursprung der abgebilteden Flächen.

Dies hat natürlich zur Folge, dass die Breite der Koeffizienten wächst und damit die Überschätzung der tatsächlichen Werte. Durch Splitting wird ein Monom in zwei Monome mit Punktintervallen zerteilt und ein neues Fehlersymbol eingeführt, das mit dem neuen Koeffizienten der Breite des Intervalls entspricht:

$$[\tilde{c}_n \pm \varepsilon_n] \rightsquigarrow [\tilde{c}_n \pm 0] + [\varepsilon_n \pm 0] \cdot \lambda_n \quad , \lambda_n \in [0 \pm 1]$$

Es ergibt sich der gegenteilige Effekt des Sweepings, da die Intervalle kleiner werden, der Grad des Polynoms jedoch erhöht wird.

2.3 Lyapunov Exponent

Mit dem Lyapunov Exponenten kann für ein dynamisches System (D, f) mit Phasenraum D und einer Funktion $f : D \to D$ angegeben werden, mit welcher Rate sich zwei beliebig nahe Punkte $x_0 \in D$ und $x_0 + \varepsilon$ mit $\varepsilon > 0$ voneinander entfernen, beziehungsweise annähen. Für eine iterierende Funktion $x_{n+1} = f(x_n)$ mit diskreten Zeitschritten wird der Lyapunov Exponent wie folgt definiert [PlBr89]:

$$LE(x_o) = \lim_{n\to\infty} \frac{1}{n} \sum_{n=0}^{n-1} \ln |f'(x_i)| \tag{2.1}$$

Beziehungsweise entwickelt sich die Entfernung zwischen den zwei Punkten ε zum Zeitpunkt n mit:

$$\varepsilon \cdot e^{n \cdot LE(x_0)} = |f^n(x_0 + \varepsilon) - f^n(x_0)| \tag{2.2}$$

Für $LE(x_0) < 0$ kontrahiert, für $LE(x_0) > 0$ wächst der Abstand zwischen den Punkten über den Verlauf der Schritte. $|f^n(x_0+\varepsilon) - f^n(x_0)|$ ist eine lokale Annäherung an den Lyapunov Exponenten zum Zeitpunkt n. Betrachtet man diese Punkte als Randpunkte eines Intervalls I so bildet der Lyapunov Exponent eine untere Schranke für dessen Ausdehnung, da I durch die Anwendung von Rechenoperation den Wertebereich überschätzt.

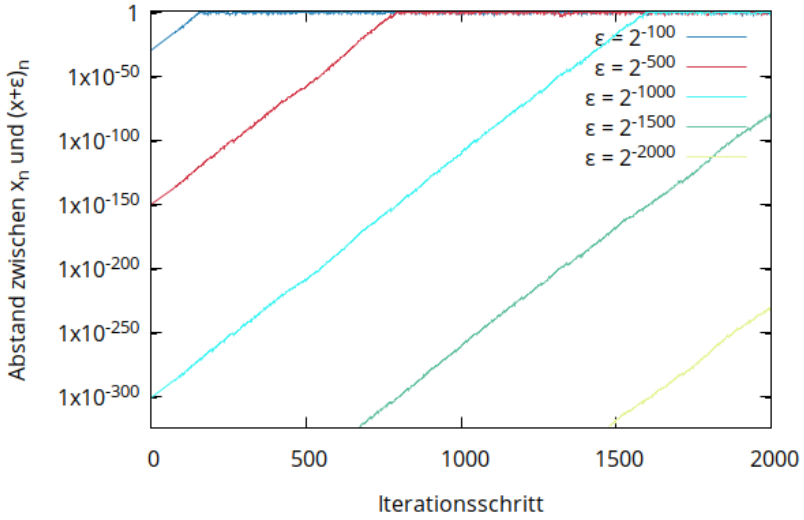


Abbildung 2.4: Lokale Annäherung an den Lyapunov Exponenten der Hénon-Iteration mit Punkten x_0 und $x_0 + \varepsilon$, und den Parametern $a = 1.4, b = 0.3$

Abbildung 2.4 zeigt eine lokale Annäherung an die Entfernung zwischen x_n und $(x+\varepsilon)_n$, wie in 2.2 bei n Iterationen und verschiedenen Größen für ε. Die Graphen stellen jeweils die Entwicklung des Abstandes $|(x+\varepsilon)_n - x_n|$ für $f^n(x, y)$, beziehungsweise $f^n(x+\varepsilon, y+\varepsilon)$ dar. Abhängig vom initialen Abstand ε bleiben die Punkte eine längere Zeit dicht beieinander, bevor der Abstand umd den Wert 1 osziliert. Ist dieser Zustand erreicht, bewegen sich die Punkte scheinbar unabhängig voneinander auf dem Attraktor, bleiben also in der Fangzone und entfernen sich daher nicht weiter.

In Abbildung 2.5 ist zu sehen, wie sich die Ausbreitung entwickelt, wenn statt zwei Punkten, Intervalle für x_0 und y_0 mit der Breite ε verwendet werden; jeweils in Schwarz eingezeichnet. Hier zeigt sich, dass sich die Breite des Intervalls x_n deutlich schneller vergrößert[1], als es der Abstand zwischen unabhängigen Punkten mit x_n und $(x+\varepsilon)_n$. Das entstehende Wrapping wächst umso schneller, je mehr die Intervallbreite $|x_n|$ dem Wert 1 nähert. Dort, wo die einzelnen Punkte um die 1 oszilieren, laufen die Intervallgrenzen innerhalb weniger Iterationen gegen ∞.

2.3.1 Verlustrate der Aussagekraft

In [Span10] wird der Lyapunov Exponent in den Kontext der Berechenbaren Analysis gerückt und in Verbindung mit der Entwicklung von Rundungsfehlern in reeller Arithmetik gebracht. Der Wert von x_n wird zu einem beliebigen Zeitpunkt n von

[1] Das Intervall y_n verhält sich vergleichbar, wird jedoch von x_{n-1} dominiert

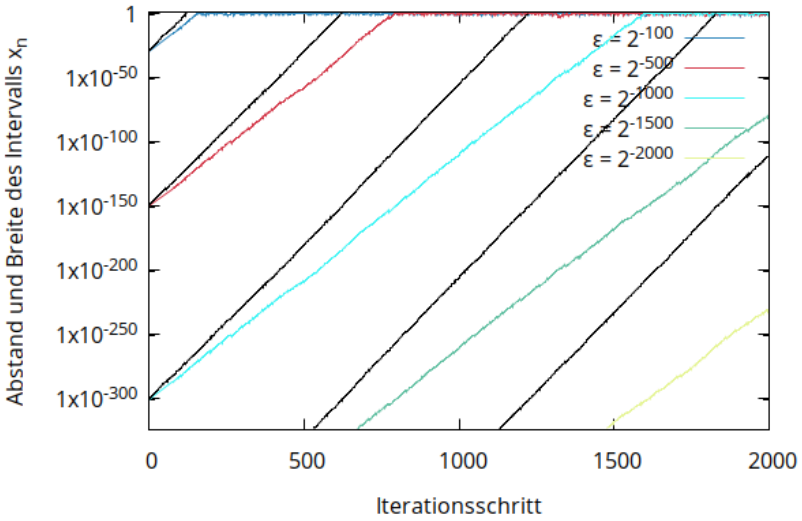


Abbildung 2.5: Lokale Annäherung an den Lyapunov Exponenten der Hénon-Iteration mit Intervallen der Breite $|x_0| = \varepsilon$, und den Parametern $a = 1.4, b = 0.3$ im Vergleich zur Entwicklung des Abstandes zweier Punkte mit Abstand ε

einem Intervall $[x_n^l, x_n^u]$ umschlossen, sodass $x_n \in [x_n^l, x_n^u]$ gilt. Die Schranken x_n^l und x_n^u sind jeweils Fließkommazahlen der Länge m_n. $\hat{x}_n$ stellt eine Approximation von x_n dar, indem der Mittelpunkt des Intervalls bestimmt und dann auf m_n Stellen gerundet wird:

$$\hat{x}_n = rd(\frac{x_n^l + x_n^u}{2}, m_n) \tag{2.3}$$

Für die Diskrepanz zwischen $\hat{x}_n$ und x_n e_n soll bei einer Genauigkeit $p \in \mathbb{Z}$ gelten, dass

$$e_n = |\hat{x}_n - x_n| \leq 10^{-p} \tag{2.4}$$

gilt. Die Wachstumsrate für $m_{min}(x, n, p)$ gibt an, wie groß m sein muss, damit 2.4 für den Startwert x nach n Iterationen bis zu einer Genauigkeit p erfüllt ist. Diese Rate wird definiert als

$$\sigma(x, p) = \lim_{n \to \infty} \frac{m_{min}(x, n, p)}{n} \tag{2.5}$$

Mit m_{min} kann nun die Verlustrate der Aussagekraft der errechneten Werte pro Zeitschritt durch

$$\sigma(x) = \lim_{p \to \infty} \sigma(x, p) \tag{2.6}$$

angegeben werden.

Die in dieser Arbeit verwendeten Taylormodelle bestehen aus Polynomen mit Intervallkoeffizienten und reellen Endpunkten. Für die praktische Umsetzung wurde die Software-Bibliothek `iRRAM` [Müll09] verwendet, welche reelle Zahlen, ähnlich

wie oben beschrieben, darstellt und annähert. Stellen diese Taylormodelle nun ein Intervall dar, so ist das Ziel, für die Darstellung der reellen Zahlen einen möglichst kleinen Wert für m zu finden, sodass die Verlustrate der Aussagekraft der reellen Intervallgrenzen erlaubt, dass sich das Intervall für eine Iterationszahl n so nah wie möglich am Lyapunov Exponenten entwickelt.

Da sich die Hénon-Abbildung nun im $\mathbb{R}^2$ bewegt, bedeutet eine Definition der Taylormodelle als Intervall, dass diese ein Rechteck aufspannen. Eine Berechnung von Hénon-Iterationen auf einem größeren Rechteck hat zu Folge, dass eine kritische Intervallbreite, bei der die Ausdehnung nicht mehr dem Lyapunov-Exponenten gleicht, sondern exponentiell schnell wächst, schneller erreicht wird. Um diesem Effekt entgegenzuwirken, kann das Rechteck und damit die Taylormodelle partitioniert und die Berechnung auf den kleineren Flächen fortgeführt werden. Mit der Entwicklung der Breite kann nun abgeschätzt werden, wie fein die Partitionierung sein muss, um eine gewisse Anzahl an Iterationen berechnen zu können.

3. Implementierung

3.1 iRRAM

Die Software-Bibliothek `iRRAM` [Müll09] implementiert die reellen Zaheln und basiert auf Intervallen als Zahlentyp, um diese mit einer beliebigen Genauigkeit darstellen zu können, ähnlich, wie in Kapitel **??** beschrieben. Zunächst wird mit Double-Präzision, also 64-Bit Zahlen gerechnet, welche verwendet werden, bis das Ergebnis für die angefragte Präzision nicht mehr genau ausgegeben werden kann, beziehungsweise bis zu einer bestimmten Anzahl an Bits. Ist dies der Fall, geschieht eine Iteration mit einer erhöhten Genauigkeit, also längeren Zahlen für die Intervallränder, welche dann mit Hilfe von `MPFR` dargestellt werden. Für eine solche Iteration werden gerade so viele Zwischenergebnisse während der Berechnung gespeichert, dass eine Wiederholung der Schritte mit höherer Präzision möglich ist. Da sich durch die Wiederholung mit höherer Genauigkeit Änderungen bei bereits durchgeführten Berechnungen ergeben können, müssen während der Laufzeit Sichtbarkeit von Variablen und Zwischenergebnissen für den Nutzer genau kontrolliert und unter Umständen beschränkt werden, da sonst unerwartetes Verhalten und Exceptions entstehen können, indem die zugegriffenen Werte gegebenfalls nicht in der aktuellen Iteration existieren.

Einige der für rationale Zahlen zur Verfügung stehenden Funktionen, wie der Vergleich zweier Zahlen, sind mit reellen Zahlen nicht ohne weiteres Möglich. Dies gilt insbesondere für den Test auf Gleichheit und die Vorzeichenfunktion *sign*. Bei all diesen Funktionen handelt es sich im Reellen (und bei der `iRRAM`) um mehrwertige Funktionen, da sich das jeweilige Ergebnis mit veränderter Präzsion in der Darstellung der Zahlen ändern kann. Dieses Problem wird in `hotm` durch den `SignType` adressiert. Zusätzlich zu den Werten 'positiv' (=`POS`) und 'negativ' (=`NEG`), kann die Vorzeichenfunktion `sign` den Wert 'ambivalent' (=`AMBI`) ausgeben, wenn nicht entscheidbar ist, wo genau die reelle Zahl um die Null liegt. Hierfür erhält die Vorzeichenfunktion einen Parameter, der den Bereich der Unsicherheit definiert. Ist der Ausgabewert 'ambivalent', so wird in den Funktionen, welche die Vorzeichenfunktion aufrufen, der schlechteste Fall im Hinblick auf die Genauigkeit, beziehungsweise die Intervallbreite des Ergebnisses angenommen.

Eine weitere Besonderheit ergibt sich aus dem Aufbau der Zahlen. Es entstehen zwei Intervall-'Ebenen': Zum Einen, die Darstellung des Koeffizienten als Intervall aus Mitte und Radius. Zum Anderen die Darstellung von Mitte und Radius, als `iRRAM-REAL`, jeweils wiederum als Intervall mit Wert und Fehler, wie in Grafik 3.1 zu sehen ist. Dies sorgt für eine recht hohe Komplexität, allerdings lässt sich die Ungenauigkeit sehr genau steuern, indem zum Beispiel der Rechenfehler des Mittelpunkts eines Koeffizienten auf den Radius 'verlagert' wird. So vergrößert sich zwar

der Radius des Koeffizienten, welcher dadurch ungenauer wird, jedoch verkleinert sich der Rechenfehler auf der Zahlenebene der REALs.

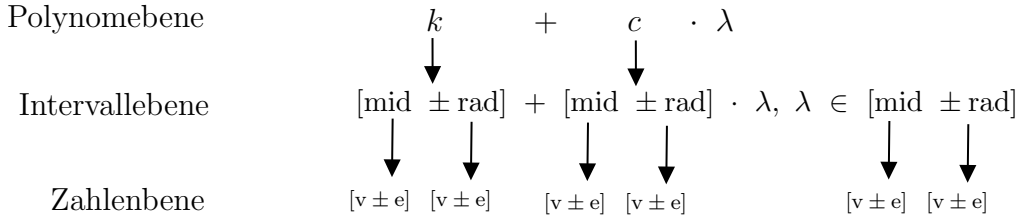


Abbildung 3.1: Ebenen der Polynomdarstellung mit REALs

3.1.1 Präzision

Werden zwei iRRAM-REALs x und y mit einer jeweiligen Präzision p_x und p_y verrechnet $z = x \circ y$, so entscheidet die Präzisions-Strategie der iRRAM `prec_policy`, ob für für diese Operation absolute oder relative Genauigkeit verwendet wird. Zudem existiert eine globale, an die Iteration der iRRAM gebundene Genauigkeit `actual_precision` p_g , die sich mit jeder Iteration verringert, beginnend mit dem Wert $p_{g0} = -50$. Diese Werte sind negativ und repräsentieren, die Anzahl an Bits, für die eine Zahl genau bekannt ist. Bei $z = x \circ y$ ist die Genauigkeit p_z durch

$$p_z = \begin{cases} max(p_x, p_y, p_g) & \text{für absolute Genauigkeit} \\ max(p_x, p_y, max(p_x, p_y) - 50 + p_g) & \text{für relative Genauigkeit} \end{cases}$$

gegeben. Für die Anwendung in HOTM eignet sich die Verwendung von relativer Genauigkeit am besten, da die Manipulation der Fehlerbreite durch Cleaning große Unterschiede in der Genauigkeit zwischen Zahlen ergibt.

3.2 Intervallarithmetik

Arithmetik auf Taylormodellen zu betreiben bedeutet auf der untersten Ebene, mit Intervallen zu rechnen. Um diese wiederum als Zahlentyp zu verwenden, müssen die Operationen angepasst werden [Moor79].

Grundrechenarten

$$[x_1, x_2] + [y_1, y_2] = [x_1 + y_1, x_2 + y_2]$$

$$[x_1, x_2] - [y_1, y_2] = [x_1 - y_2, x_2 - y_1]$$

$$[x_1, x_2] \cdot [y_1, y_2] = [min(x_1y_1, x_1y_2, x_2y_1, x_2y_2), max(x_1y_1, x_1y_2, x_2y_1, x_2y_2)]$$

$$[x_1, x_2]/[y_1, y_2] = [min(x_1/y_1, x_1/y_2, x_2/y_1, x_2/y_2), \\ max(x_1/y_1, x_1/y_2, x_2/y_1, x_2/y_2)]$$

Potenzfunktion

Die Potenzfunktion unterscheidet zwischen geradem und ungeradem Exponenten.

$$[x_1, x_2]^n = \begin{cases} [x_1^n, x_2^n] & falls\ x_1 > 0\ oder\ n\ ungerade; \\ [x_2^n, x_1^n] & falls\ x_2 < 0\ und\ n\ gerade, \\ [0, max(x_1^n, x_2^n)] & falls\ 0 \in [x_1, x_2]\ und\ n\ gerade; \end{cases}$$

Kehrwert

Der Kehrwert eines Intervalls kann nur bestimmt werden, wenn es nicht die 0 enthält:

$$1/[x_1, x_2] = [1/x_2, 1/x_1]$$

Wenn $x_1 \leq 0 \leq x_2$, bedeutet das, dass für ein $x \in [x_1, x_2]$, $\frac{1}{x} \geq \frac{1}{x_2}$ oder $\frac{1}{x} \leq \frac{1}{x_1}$ gelten muss und das Intervall damit unbegrenzt ist.

Die im Basisprogramm `tangentspace` vorhandene Implementierung wurde erweitert, um Punktintervalle gesondert zu behandeln und die nun auf reellen Zahlen basierenden Intervalldefinitionen zu verwenden. Dies hat zur Folge, dass die Mehrwertigkeit der Vorzeichenfunktion berücksichtigt werden muss. Tritt der Fall der Unsicherheit ein, so muss mit einer Überschätzung gerechnet werden, die in jedem Fall korrekt ist.

3.3 Polynomarithmetik

Die Verwendung von Geobuckets ist eine Methode, die Addition [Yan98] und Mulitplikation [MoPe07] zweier Polynome effizient zu gestalten, indem die Zahl der dafür nötigen Monomvergleiche reduziert wird.

3.3.1 Polynomaddition

Die Addition zweier Polynome p_1 und p_2 stellt ein Zusammenfügen zweier Listen von Monomen (*Merge*) dar. Sind diese geordnet, so sind im schlimmsten Falle $\#p_1 + \#p_2$ Vergleiche nötig, während es für ungeordnete Polynome $\#p_1 \cdot \#p_2$ sind, da alle Monome miteinander verglichen werden müssen. Die hierfür verwendete partielle Ordnung $\succ$ sortiert die Monome nach dem Index der Variablen und deren Exponenten:

- $\lambda_1 \succ \lambda_2 \succ ... \succ \lambda_k$
- $\lambda_n^i \lambda_m^j \succ \lambda_n^{i'} \lambda_m^{j'}$ für $n > m$, falls

  $i > i'$, oder

  $i = i'$ und $j > j'$

Algorithmus **??** erzeugt aus zwei nach der oben definierten Ordnung $\succ$ sortierten Polynomen wiederum ein sortiertes Polynom. Endet die While-Schleife in Zeile 2, so ist im Regelfall eines der Polynome noch nicht gänzlich betrachtet, was bedeutet, dass alle übrigen Monome in diesem Polynom (im Hinblick auf $\succ$) kleiner sind, als die des anderen Summanden und daher dem Ergebnispolynom angehangen werden können, ohne sie weiter zu betrachten.

3.3.2 Polynommultiplikation

Betrachtet man die Summe dreier Polynome $p_1+p_2+p_3$ mit $\#p_1 >> \#p_2 = \#p_3 = 1$, benötigt das Aufsummieren von links nach rechts bis zu $2\#p_1 + 1$ Vergleiche, da mit dem oben verwendeten Additionsverfahren zweifach in p_1 eingefügt wird und potentiell jedes Monom miteinander verglichen werden muss[1]. Summiert man jedoch von rechts nach links so werden lediglich $\#p_1+2$ Vergleiche benötigt. Nach dieser Idee kann die Anzahl der Monomvergleiche reduziert werden, indem zunächst Polynome gleicher Länge miteinander addiert werden.

Um die oben (3.3) definierte Ordnung $\prec$ auf dem Polynom $p = p_1 \cdot p_2$ zu erhalten, müssen die $n \cdot m$ Monome ($n := \#p_1, m := \#p_2$), die bei der Multiplikation entstehen, sortiert werden. Werden diese sequentiell zu p hinzugefügt bedeutet das 2 Vergleiche, dann 3, dann 4, und so weiter, bis hin zu nm Vergleichen:

$$2 + 3 + 4 + 5 + ... + nm = \frac{nm \cdot (nm + 1)}{2} - 1$$

So ergibt sich für die naive Multiplikation eine quadratische Laufzeit in O-Notation von $O(nm + (nm)^2)$ Mit Geobuckets werden die Zwischenergebnisse der Polynommultiplikation in geometrisch wachsenden 'Buckets' gespeichert. Hat ein Bucket seine Kapazität erreicht, wird das Polynom zum nächst größeren Bucket hinzuaddiert. Der Unterschied zum Originalalgorithmus besteht darin, dass die Größe der hinzukommenden Polynome bereits bekannt ist und die Bucketkapazität dementsprechend angepasst werden kann. Algorithmus 4 zeigt die in `HOTM` implementierte Version der Geobucketmultiplikation. Durch den Teile und Herrsche Ansatz, kann die Multiplikation mit Geobuckets so in $O(nm \log nm)$ durchgeführt werden.

[1] $\#p$ liefert die Anzahl an Monomen des Polynoms p

3.4 Taylormodelle

Taylormodelle bestehen in HOTM aus einer Liste von nach der Ordnunng $\succ$ (siehe 3.3) sortierten Monomen. Für die Supportintervalle der Fehlersymbole wird eine statische Liste, für die Grenzwerte der Housekeeping-Methoden und den Index neuer Fehlersymbole jeweils statische Variablen angelegt.

3.4.1 Housekeeping-Methoden

Die Housekeeping-Methoden Splitting und Sweeping werden in [BrKM15] beschrieben und dienen der Kontrolle der Intervallkoeffizienten und deren Wachstum. Wann die Methoden angewandt werden, wird durch die manuell justierbaren Parameter MACRO_THRESHOLD für Splitting und MICRO_THRESHOLD für Cleaning festgelegt. Da für diese beiden Methoden Vergleiche reeller Zahlen nötig sind, um das Überschreiten eines Grenzwertes zu bestimmnen, werden die Zahlen mit Funktionen der iRRAM in rationale Approximationen zerlegt.

Ziel der Housekeeping-Methoden ist, eine Balance zwischen der Breite der Intervallkoeffizienten und dem Erhalt von Abhängigkeitsinformationen durch längere Polynome, also mehr Fehlersymbolen, zu halten, sodass die Monomzahl und der Informationsverlust gleichsam klein bleiben.

Cleaning

Wie oben beschrieben, ergibt die Verwendung der iRRAM-REALs als Intervallgrenzen zweistufige Intervalle mit der Zahlen- und Intervallebene (siehe Abbildung 3.1). Mit fortlaufenden Rechnungen wächst die Breite der Intervalle auf Zahlenebene kontinuierlich und kann ohne weitere Methoden nur durch ein Erhöhen der verwendeten Präzision wieder verringert werden, beziehungsweise durch eine Iteration der iRRAM. *Cleaning* verlagert den Rechenfehler der Zahlenebene auf die Intervallebene und macht ihn damit auch für andere Housekeeping-Methoden sichtbar. Das Intervall $I = [m \pm r]$ mit $m = [c_m \pm \varepsilon_m]$ und $r = [c_r \pm \varepsilon_r]$ wird um ε_m und ε_r vergrößert, sodass das I zwar wächst, jedoch dessen Endpunkte exakt sind:

$$\begin{aligned}[[c_m \pm \varepsilon_m] \pm [c_r \pm \varepsilon_r]] &\rightsquigarrow [[c_m \pm 0] \pm [c_r + \varepsilon_m + \varepsilon_r \pm 0]] \\ &= [c'_m \pm \underbrace{c'_r}_{c_m + \varepsilon_m + \varepsilon_r}]\end{aligned}$$

Algorithmus **??** zeigt eine vereinfachte Darstellung der Cleaning Prozedur.

Splitting

Durch Cleaning, Sweeping oder eine Initialisierung der Taylormodelle mit einer gewissen Breite wachsen die Radii der Koeffizienten im Laufe einer Berechnung auch auf Intervallebene. Um die Überschätzung, die durch Intervallarithmetik entsteht zu kontrollieren, kann durch *Splitting* ein Monom in zwei Monome mit Punktintervallen als Koeffizienten und einem neuen Fehlersymbol aufgeteilt werden:

$$[\tilde{c}_n \pm \varepsilon_n] \;\rightarrow [\tilde{c}_n \pm 0] + [\varepsilon_n \pm 0] \cdot \lambda_n \quad , \lambda_n \in [0 \pm 1]$$

Dadurch erhöht sich der Grad des Polynoms, jedoch verringert sich die Überschätzung durch Intervallarithmetik. Eine Alternative das Splitting zu realisieren ist, den Fehler statt wie in [BrKM15] im Koeffizienten des neuen Monoms, im neuen Fehlersymbol zu kodieren:

$$[\tilde{c}_n \pm \varepsilon_n] \rightsquigarrow [\tilde{c}_n \pm 0] + [1 \pm 0] \cdot \lambda_n \quad , \lambda_n \in [0 \pm \varepsilon_n]$$

Dies hat den Vorteil, dass die Größe der Fehlersymbole beim Sweeping berücksichtigt werden kann.

Sweeping

Cleaning und Splitting sorgen dafür, dass die Intervallbreite der Koeffizienten klein bleibt, jedoch erhöht sich der Grad der Polynome, was zum Beispiel bei der Multiplikation von Taylormodellen zu Ineffizienz durch die exponentiell wachsende Anzahl an Monomen führt. Um diesem Effekt entgegenzuwirken, kann mit *Sweeping* ein Fehlersymbol durch sein Supportintervall ersetzt werden:

$$c_n \lambda_i^k \rightsquigarrow c_n s_i \lambda_i^{k-1}$$

Hierbei ist zu beachten, dass das n-fache Sweepen eines Fehlersymbols, welches die 0 enthält, mit n gerade eine geringere Breite in den Koeffizienten einführt, als n ungerade (siehe oben 3.2). Daraus ergeben sich zwei Sweeping-*Strategien*.

square_only Die Sweeping-Strategie `square_only` beschränkt das Sweeping auf gerade Potenzen. So wird die angestrebte Reduktion des Grades eines Monoms nur erreicht, wenn alle Variablen einen geraden Exponenten haben.

square_first Mit `square_first` wird der angestrebte Grad erreicht, indem zunächst möglichst viele Fehlersymbole gerade gesweept werden. Falls dadurch jedoch nicht die gesamte Reduktion möglich ist, wird der Rest ungerade gesweept.

Der Aufruf der Methode erhält in `hotm` drei Parameter; das zu reduziere Taylormodell, den Grad, auf das Taylormodell durch Sweeping reduziert werden soll und die Strategie:

```
tmsimple x;
x = sweep_to(x, 2, SQUARE_ONLY);
```

Listing 3.1: Beispielaufruf der Sweeping-Routine

3.4.2 Darstellung der Taylormodelle

Die grafischen Darstellungen der Taylormodelle in dieser Arbeit wurden mit `gnuplot` [WiKm20] erstellt und zeigen je eine Übermenge des tatsächlichen Taylormodells. Eine Darstellung eines Taylormodells entsteht, indem für jedes der k Fehlersymbol aus $\lambda = (\lambda_1, \ldots, \lambda_k)$ ein Wert festgelegt und dann das Taylormodell evaluiert wird. Um eine Übermenge zu zeichnen muss der gesamte Wertebereich eines jeden Fehlersymbols abgedeckt werden. In `hotm` geschieht das, indem für eine gegebene *Auflösung* $a \in \mathbb{N}$ die Supportintervalle $S = (s_1, \cdots s_k)$ in jeweils a Intervalle geteilt

und für jede Kombination mit eingesetzten Werten ein von den anderen unabhängiges Rechteck gezeichnet wird. So ergeben sich für Taylormodelle mit k Variablen a^k Rechtecke. Eine solche Darstellung wird in [BrKM15] als $image(T) \subseteq \mathbb{R}^d$ definiert. Die hier verwendete Darstellung wird um die verwendete Auflösung erweitert: $image_{acc}(T, a)$.

Seien T_1 und T_2 Taylormodelle mit $k = 2$ Variablen, die eine zweidimensionale Fläche aufspannen, bei denen die initialen Fehlersymbole erhalten bleiben und keine neuen Fehlersymbole eigeführt werden, so kann durch die Abhängigkeitsinformation der Fehlersymbole Ursprung und Abbild der einzelnen Regionen dieser Fläche farblich induziert werden. Abbildung 3.3 zeigt eine Iteration der Hénon-Abbildung eines Rechtecks mit verschiedenen Auflösungen. Es ist gut zu erkennen, dass die gröbere Auflösungen, die feinere überdeckt und $image_{acc}(T_1, 100) \times image_{acc}(T_2, 100) \subseteq image_{acc}(T_1, 10) \times image_{acc}(T_2, 10) \subseteq image_{acc}(T_1, 2) \times image_{acc}(T_2, 2)$ gilt. Durch die Einfärbung der Rechtecke können verschiedene Effekte beobachtet werden:

1. Die Rechtecke überlagern sich im Abbild, obwohl sie sich im Ursprung nicht überschneiden und die Hénon-Abbildung an sich eindeutig invertierbar ist. Grund hierfür ist die entstehende Überschätzung durch Intervallarithmetik, die besonders bei Multiplikation, beziehungsweise Quadrierung der Intervalle zum Tragen kommt.

2. Das in den Grafiken eingezeichnete Tetragon stellt die Fangzone R für die hier verwendeten Parameter $a = 1.4$ und $b = 0.3$ dar. Mit Hilfe der Einfärbung ist erkennbar, wie Regionen außerhalb von R wiederum außerhalb abgebildet werden, während innerhalb liegende diese nach der Abbildung nicht verlassen.

Algorithm 1: Algorithmus für Cleaning eines Taylormodell

Eingabe : Taylormodell T: Liste von n Koeffizienten $c_i = [m_i \pm r_i]$ und $m_i = [\hat{c}_{m_i} \pm \varepsilon_{m_i}],\ r_i = [\hat{c}_{r_i} \pm \varepsilon_{r_i}]$, Grenzwert δ_c

Ausgabe: Taylormodell T': Liste von Monomen mit Punktintervallkoeffizienten

```
1  T' ← T                                   (Kopiere das alte Taylormodell)
2  forall i ∈ {1, ..., n} do
3      if ε_{m_i} > δ_c then                (Überschreitet Fehler des Mittelpunktes den Grenzwert)
4          r'_i ← r_i + ε_{m_i}             (Erweitere das Intervall um den Fehler)
5          m'_i ← [c_{m_i}]                 (Setze die Intervallbreite des Mittelpunktes auf 0)
6      end
7      if ε_{r_i} > δ_c then                (Überschreitet Fehler des Radius' den Grenzwert)
8          r'_i ← r_i + ε_{r_i}             (Erweitere das Intervall um den Fehler)
9          r'_i ← [c_{r_i}]                 (Setze die Intervallbreite des Radius' auf 0)
10     end
11 end
12 return T'                                (T' hat Punktintervalle oder sehr kleine Intervalle auf Zahleneben)
```

Algorithm 2: Algorithmus für Splitting

Eingabe : Taylormodell T: Liste von n Monomen $m_i = c_i \cdot \lambda^i$ mit Koeffizienten $c_i = [\hat{c}_i \pm \varepsilon_i]$ und der Liste der Fehlersymbole λ^i, Grenzwert für Splitting δ_s

Ausgabe: Taylormodell T': Liste von Monomen mit Punktintervallkoeffizienten

forall $i \in \{1, \ldots, n\}$ **do**
  if $\varepsilon_i > \delta_s$ **then** *(Die Intervallbreite überscheitet den Grenzwert)*
    $s_{new} \leftarrow [0 \pm \varepsilon_i]$ *(Führe neues Fehlersymbol ein)*
    $S \leftarrow S \cup s_{new}$ *(Erweitere die Liste der Supportintervalle)*
    $m'_i \leftarrow [\hat{c}_i] \cdot \lambda^i$ *(Entferne den Radius von m_i)*
    $m''_i \leftarrow [1] \cdot \lambda^i \lambda_{new}$ *(Erstelle ein neues Monom mit neuem Fehlersymbol)*
    $T' \leftarrow T' + m'_i + m''_i$
  else
    $T' \leftarrow T' + m_i$
  end
end
return T' *(Das neue Polynom hat maximal $2n + 1$ Monome)*

Algorithm 3: Algorithmus für Sweeping mit der Beschränkung auf quadratisches Sweeping (square_only)

Eingabe : Taylormodell T: Liste von n Monomen $m_i = c_i \cdot \lambda^i$ mit Koeffizienten $c_i = [\hat{c}_i \pm \varepsilon_i]$ und der Liste der Fehlersymbole λ^i, Zielgrad γ

Ausgabe: Taylormodell T': Liste von Monomen mit Punktintervallkoeffizienten

forall $i \in \{1, \ldots, n\}$ **do**
  $o \leftarrow \sum_{\lambda^l_j \in \lambda^i} l$ *(Bestimme den Grad des Monoms)*
  if $o \leq \gamma$ **then**
    $T' \leftarrow T' + m_i$ *(Überspringe dieses Monom, wenn der Zielgrad bereits korrekt ist)*
  else
    $r \leftarrow o - \gamma$ *(Bestimme die nötige Gradreduktion, um das Ziel zu erreichen)*
    $c'_i \leftarrow c_i$
    $\lambda^{i'} \leftarrow \{\lambda^l_j | \lambda^l_j \in \lambda^i \wedge |s_j| < |s_{j+1}|\}$ *(Sortiere die λ aufsteigend nach ihrer Größe)*
    forall $\lambda^l_j \in \lambda^{i'}$ **do** *(Iteriere über die geordnete Liste von λ)*
      $p \leftarrow min(l, r)$ *(Bestimme die größtmögliche Reduktion für dieses λ)*
      if $p > 2$ ***and*** $2 \nmid p$ **then** $p \leftarrow p - 1$ *(Wähle geraden Exponenten)*
      $c'_i \leftarrow c'_i \cdot s_j^p$ *(Sweepe den einen Teil des Fehlersymbols in den Koeffizienten)*
      $l \leftarrow l - p$ *(Verringere den Exponenten des reduzierten λ)*
      $r \leftarrow r - p$ *(Aktualisiere den Wert der noch nötigen Gradreduktion)*
      if $r < 2$ **then break** *(Keine quadratische Reduktion mehr möglich)*
    end
    $\lambda^{i'} \leftarrow \{\lambda^l_j | \lambda^l_j \in \lambda^{i'} \wedge \lambda_j \succ \lambda_{j+1}\}$ *(Stelle die Ordnung $\succ$ wieder her)*
    $T' \leftarrow T' + c'_i \cdot \lambda^{i'}$
  end
end
return T'

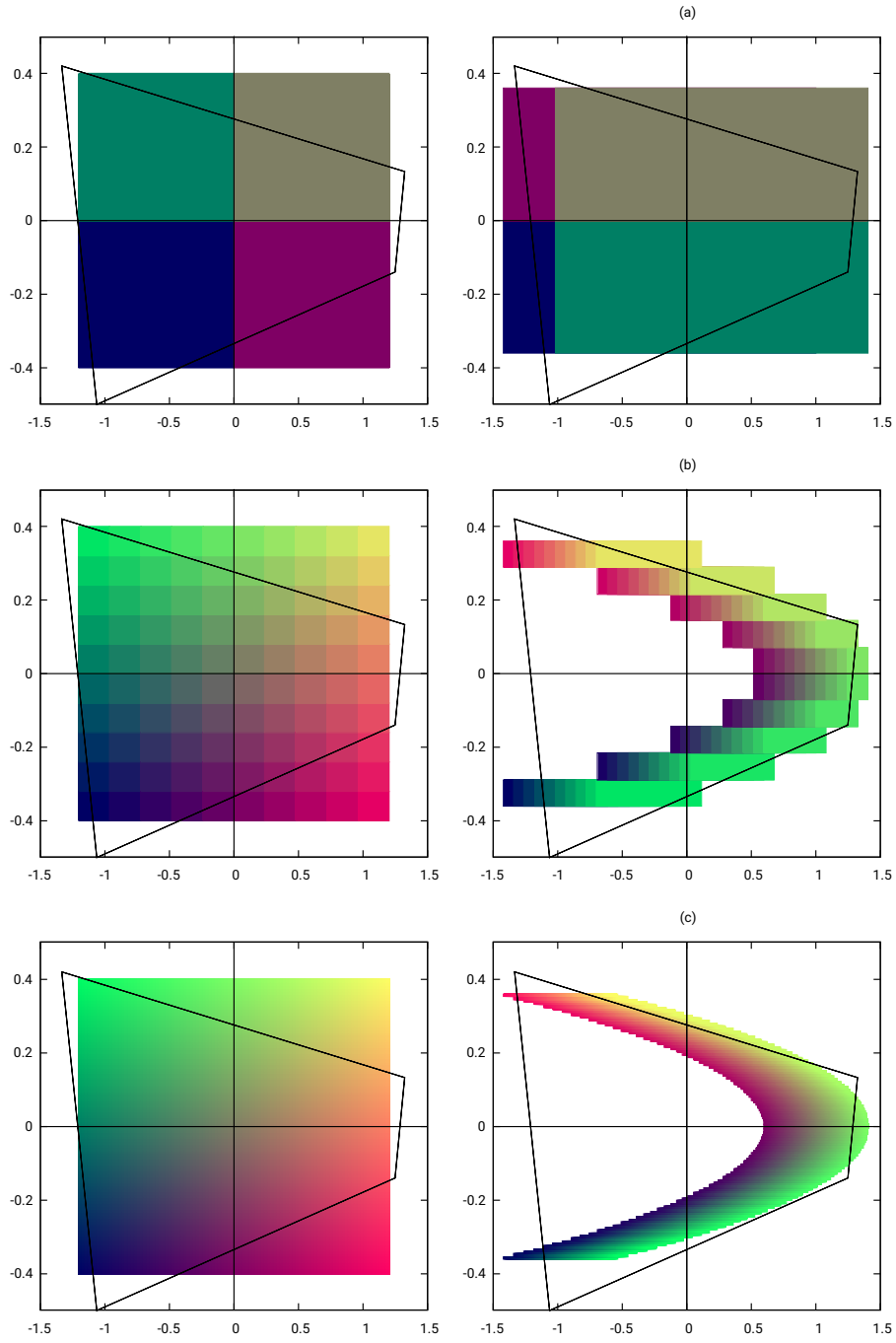


Abbildung 3.2: Jede Zeile zeigt eine einfache Abbildung der Rechtecke durch die Hénon-Abbildung mit $a = 1.4$ und $b = 0.3$ mit verschiedenen Auflösungen für die Darstellung. (a): 4 Rechtecke; (b): 100 Rechtecke, (c): 10000 Rechtecke. Das Tetragon stellt die Fangzone für die verwendeten Parameter dar.

4. Evaluation

Werden die Initialwerte für Iterationen der Hénon-Abbildung (x_0, y_0) als Taylormodelle mit Itervallen definiert, kann eine Fläche beschrieben werden, die durch die Abbildung gespiegelt, gedehnt und verzerrt wird. Liegt diese für $a = 1.4$ und $b = 0.3$ komplett innerhalb der Fangzone R, so wird jeder Punkt in der Fläche wiederum nach R abgebildet.

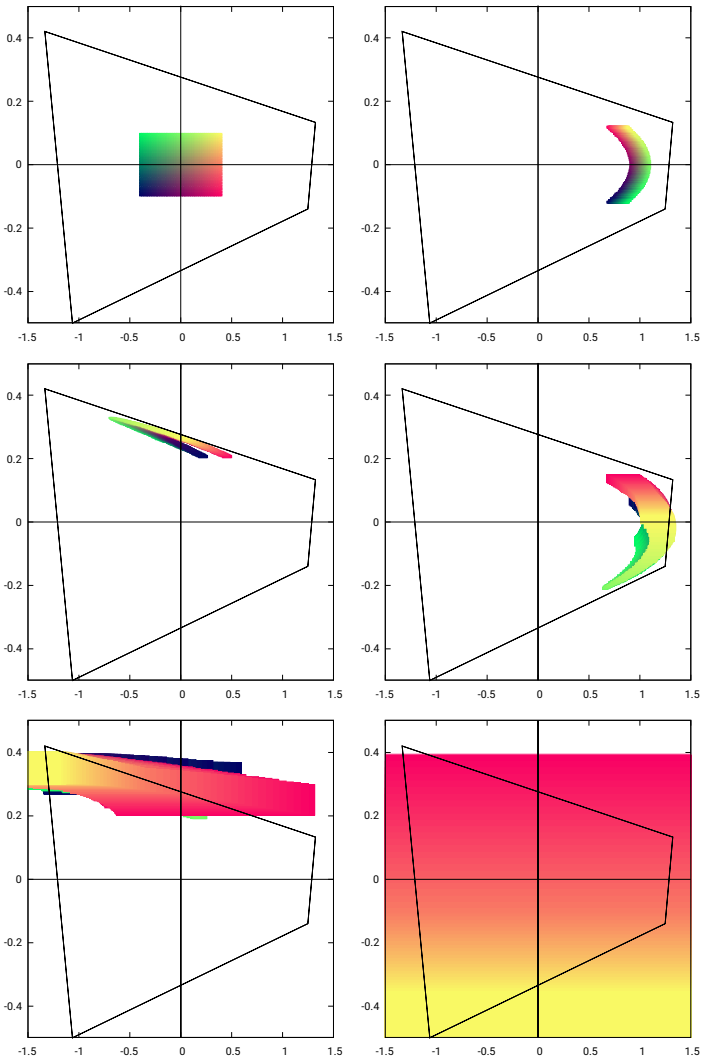


Abbildung 4.1: Iterationen der Hénon-Abbildung ausgehend von einem größeren Rechteck

Eine Abbildung der Fläche im Ganzen führt jedoch zu einer Überschätzung, die in jeder Iteration zunimmt, da nun mit Intervallen statt mit Punkten gerechnet wird. Abbildung 4.1 zeigt pro Graph eine Iteration mit

$$x_0 = 0 + 1 \cdot \lambda_1 \quad (\lambda_1 \in [0 \pm 0.4])$$
$$y_0 = 0 + 1 \cdot \lambda_2 \quad (\lambda_2 \in [0 \pm 0.1])$$

und der Fangzone als schwarzes Tetragon. Es ist erkennbar, dass das Rechteck nach wenigen Iterationen die Fangzone verlässt und einige Itervalle ein starkes Rauschen verursachen. Abbildung 6.1 zeigt Iterationen mit kleineren initialen Intervallen, wodurch die Überschätzung erst zu einem späteren Zeitpunkt zu groß wird. Um dem Wachstum der Intervalle entgegenzuwirken und die Berechnung der Hénon-Abbildung auch für solche Flächem mit höheren Iterationszahlen zu ermöglichen, können die Taylormodelle in Partitionen aufgeteilt werden. Die kleinste Partitionierung wäre die Aufteilung der Fläche in unendlich viele Punkte, was nicht praktikabel wäre, allerdings hohe Iterationszahlen ermögliche, wie in Abbildung 2.2 zu sehen ist. Das andere Extrem stellen große Itervalle, beziehungsweise keine Partitionierung dar, was im Vergleich nur einen Bruchteil des Rechenaufwandes bedeutet, jedoch schnell zu starker Überschätzung führt.

Um die Fragestellung, wie die Hénon-Abbildung mit Taylormodellen am besten berechnet werden kann zu erörtern, wird das Problem in drei Sektionen unterteilt:

1. Wie klein müssen die Taylormodelle sein, damit längere Berechnungen möglich sind?
2. Wie verhalten sich die verschiedenen Partitionsgrößen bei unterschiedlichen Konfigurationen der Taylormodelle?
3. Welche Partition ist für eine gegebene Anzahl an Iterationen nötig?

Die Implementierung nichtlinearer Taylormodelle ergibt eine Vielzahl von Konfigurationsmöglichkeiten und damit einen großen Suchraum nach der optimalen Einstellung:

- Grenzwert für Cleaning δ_c, Splitting δ_s
- Grad der Reduktion durch Sweeping
- Anzahl der zu erhaltenden Fehlersymbole
- Strategie beim Sweeping
- Heuristik für die Reihenfolge, in der Fehlersymbole gesweept werden
- Vorgehen beim Splitting
- Definition des initialen Taylormodells

Diese Konfigurationen werden in `hotm` als JSON-Datei mit Hilfe einer JSON-Bibliothek[1] eingelesen und verarbeitet, um wiederholte Durchläufe mit leicht veränderten Parametern oder Batch-Runs zu vereinfachen. Listing 6.1 zeigt ein Beispiel einer Konfigurationsdatei, mit der versuchsweise 1000 Iterationen der Hénon-Abbildung berechnet werden sollen.

[1] https://github.com/nlohmann/json (Stand Dezember 2020)

4.1 Housekeeping-Grenzwerte

Der Grenzwert einer Housekeeping-Methode gibt an ab welchem Wert die jeweiligen Methoden angewandt werden. Bei einem intervall in `hotm` $[m \pm r]$ mit $m = [c_m \pm \varepsilon_m]$ und $r = [c_r \pm \varepsilon_r]$ als `iRRAM-REAL`s wird

- Cleaning angewandt, wenn $\varepsilon_m < \delta_c$ oder $\varepsilon_r < \delta_c$ gilt und
- Splitting angewandt, wenn $c_m < \delta_s$ oder $c_m < \delta_s$ gilt.

Um diese Werte zu untersuchen, wurde betrachtet, wie sich die Größe des Rechtecks, welches durch die Intervalle x und y aufgespannt wird, abhängig von den Grenzwerten δ_c und δ_s auswirkt, beziehungsweise, wieviele Iterationen der Hénon-Abbildung bei einer festen Genauigkeit der `iRRAM-REAL`s möglich sind, bis das Rechteck eine Fläche[2] von $> 2^{-5}$ erreicht hat.

1000 Bits Präzision $x = [0 \pm 2^{-1000}], y = [0 \pm 2^{-1000}]$		
δ_c $(2^{-\delta_c})$	δ_s $(2^{-\delta_s})$	Iterationen
-	-	451
10	10	162
10	5	160
100	100	273
100	50	258
500	500	794
500	250	746
1000	1000	1432
1000	500	1317

10000 Bits Präzision $x = [0 \pm 2^{-10000}], y = [0 \pm 2^{-10000}]$		
δ_c $(2^{-\delta_c})$	δ_s $(2^{-\delta_s})$	Iterationen
-	-	4360
100	100	1463
100	50	1450
1000	1000	2641
1000	500	2530
5000	5000	7849
5000	2500	7289
10000	10000	14455
10000	5000	13401

Tabelle 4.1: Berechnung der Fläche des Rechtecks mit verschiedenen Grenzwerten für Cleaning δ_c und Splitting δ_s und festgelegter Präzision.

Tabelle 4.1 zeigt experimentelle Ergebnisse für $x_0 = 0 + 1 \cdot \lambda_1$ und $y_0 = 0 + 1 \cdot \lambda_2$ mit $\lambda_1, \lambda_2 \in [0 \pm \varepsilon]$. Sowohl für 1000, als auch für 10000 Bits Genauigkeit reagiert das Ergebnis sehr sensibel auf die Grenzwerte der Housekeeping-Methoden. Falsch gewählte Werte vergrößern sogar die entstandene Überschätzung, im Vergleich zuer Berechnung ohne Cleaning und Splitting, jeweils in Zeile 1 der Tabellen zu sehen. Werden die Grenzwerte durch die `iRRAM`-Präzision bestimmt, so bleibt der Fehler am längsten klein und die höchste Iterationszahl ist möglich.

Wie in Abbildung 4.2 zu sehen ist, hat eine Definition der Grenzwerte unter die zugrunde liegende Genauigkeit keinen Effekt mehr auf die Performanz der Berechnung, da sich die Iterationszahl nicht weiter erhöht.

[2] Ab einer Fläche von zirka 2^-5 ist die Überschätzung der Intervalle zu groß und wächst innerhalb weniger Iterationen (< 10) gegen ∞.

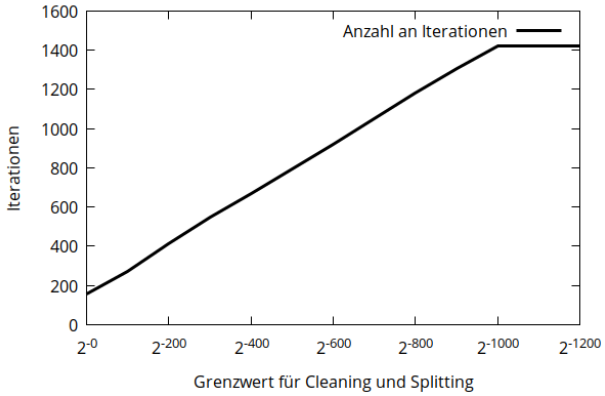


Abbildung 4.2: Iterationszahl der Hénon-Abbildung mit fester Genauigkeit von 1000 Bits.

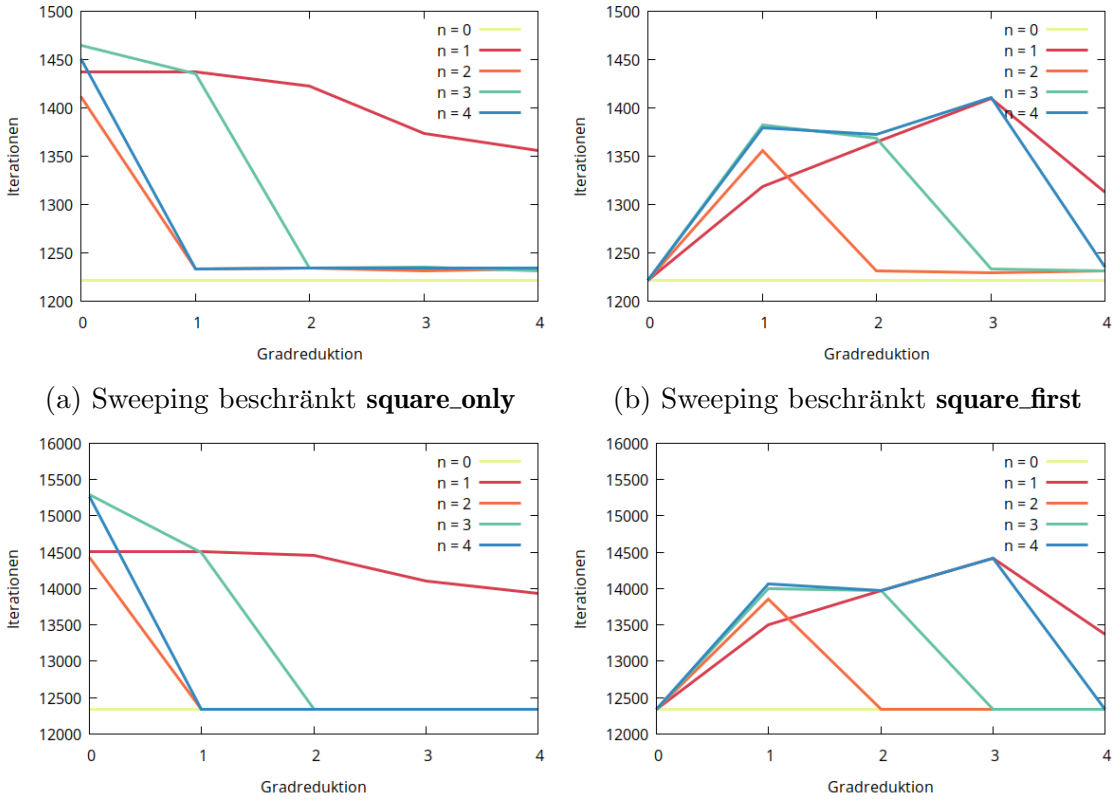
4.2 Sweeping

Für eine Sweeping-Konfiguration kommen drei einander beeinflussende Parameter in Betracht: die Sweeping-Strategie, die Gradreduktion und die Anzahl der zu erhaltenen Fehlersymbole. Der Housekeeping-Vorgang für ein Taylormodell inklusive Sweeping besteht wiederum aus vier Schritten:

1. **(Sweeping)** Reduktion des Taylormodells bis zum angegebenen Grad, je nach Strategie
2. **(Sweeping überschüssiger Fehlersymbole)** Entfernen aller Fehlersymbole bis auf die n-größten (abhängig von deren Support-Space)
3. **(Cleaning)** Verlagern der durch die Schritte 1 und 2 entstandenen Rechenfehler auf der Ebene der `iRRAM-REAL`s auf die Radii der Intervalle
4. **(Splitting)** Einführen neuer Fehlersymbole für Monome, deren Koeffizienten durch die Schritte 1-3 zu stark gewachsen sind.

Abbildung 4.3 zeigt die Performanz der Berechnung der Hénon-Abbildung, gemessen an der erreichten Iterationszahl mit statischer Präzision in verschiedenen Fällen. Wie zuvor ist die Berechnung beendet, sobald die Fläche des Rechtecks einen Schwellenwert überschreitet. Zu sehen ist, bei welcher Kombination von Parametern für Sweeping (Schritt 1) und Sweping überschüssiger Fehlersymbole (Schritt 2) die Überschätzung am langsamsten wächst und somit die meisten Iterationen errechnet werden können. Die Zeiteffizienz ist in diesem Falle kein Faktor. Es ist zu erkennen, dass sich die Graphen innerhalb derselben Sweeping-Strategie kaum unterscheiden. Eine Verzehnfachung des Exponenten in der Breite der Intervalle und der Bits für die Präzision verzehnfacht auch die erreichte Iterationszahl. Die besten Ergebnisse werden beim Beschränken auf **square_only** (quadratisches Sweeping) für $n = 3$ und Sweeping zum Grade 0; bei **square_first** für $n = 1$, $n = 4$ und Sweeping zum Grade 3 erreicht. Insgesamt wird jedoch mit **square_only** zum Grade 0, also einem Sweeping bis zu einem Taylormodell, dessen Monome Variablen mit höchstens Grad 1 haben, die höchste Iterationszahl erzielt:

$$c \cdot \lambda_1^3 \lambda_2^2 \lambda_3^1 \lambda_4^3 \overset{sweep\ to\ 0}{\underset{square\_only}{\rightsquigarrow}} c' \cdot \lambda_1^1 \lambda_2^0 \lambda_3^1 \lambda_4^1$$



(a) Sweeping beschränkt **square_only**

(b) Sweeping beschränkt **square_first**

(c) Sweeping beschränkt **square_only**

(d) Sweeping beschränkt **square_first**

Abbildung 4.3: Anzahl an Iterationen mit Erhalt der n größten Fehlersymbole. (a) (b): 1000 Bits Präzision , $x_0 = y_0 = [0 \pm 2^{1000}]$. (c)(d): 10000 Bits Präzision , $x_0 = y_0 = [0 \pm 2^{10000}]$

Im Vergleich mit rein linearen Taylormodellen zeigt sich eine Verbesserung (siehe Abbildung 4.4). Der Unterschied besteht darin, dass die nichtlinearen Taylormodelle während der Berechnung und besonders während der Multiplikation Punktintervalle erhalten können und sich stattdessen der Grad des Polynoms erhöht. Nach jeder Iteration werden diese dann zunächst im Grad reduziert, um diesen dann durch Splitting auf den Monomen um maximal 1 zu erhöhen. Für lineare Taylormodelle steht lediglich das Splitting für den Kernel zur Verfügung, da sonst nichtlineare Polynome entstünden. Des Weiteren wird hier bereits während der Multiplikation gesweept und Punktintervalle werden zu regulären Intervallen. Die Performanz des Sweepings **square_only** zum Grade 0 verbessert sich im Schnitt mit einer wachsenden Zahl der zu erhaltenen Fehlersymbole, wie in Abbildung 4.4 zu sehen ist. Jedes Fehlersymbol erhöht die Menge an erhaltener Abhängigkeitsinformation und sorgt dafür, dass ein Taylormodell dessen Ränder besser beschreiben kann. Allerdings stellt die Praktikabilität dieser Herangehensweise ein Problem dar, da so sehr große Polynome entstehen, die während einer Iteration der Hénon-Abbildung exponentiell in ihrer Länge wachsen.

Dieser Effekt ist in Tabelle 4.2 zu sehen. Eine Erhöhung der Anzahl der erhaltenen Fehlersymbole hat einen drastischen Einfluss auf die Durchschnittliche Dauer einer

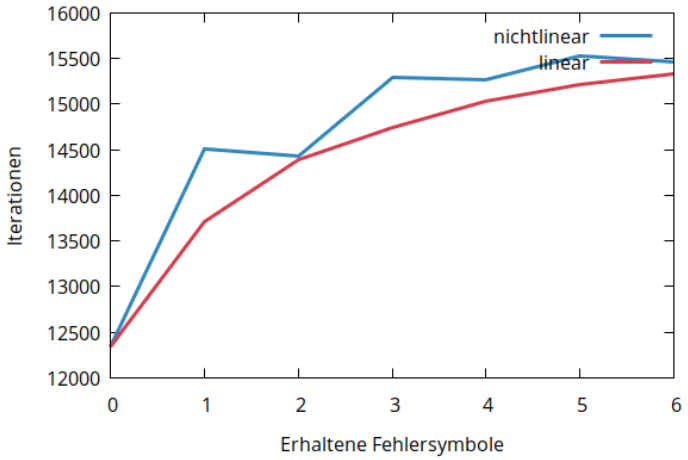


Abbildung 4.4: Gegenüberstellung von $x_0 = y_0 = [0 \pm 2^{10000}]$ als lineare Taylormodelle und nichtlineare Taylormodelle mit quadratischem Sweeping zum Grade 0. Jeweils bleiben verschiedene Anzahlen von Fehlersymbolen pro Iteration erhalten. Gemessen wird die Iterationszahl der Hénon-Abbildung mit fester Genauigkeit von 10000 Bits, bis $|x_n| \cdot |y_n|$ einen Grenzwert von 2^{-6} überschreitet.

Iteration in ms, bis das Programm bei größeren Werten quasi nicht mehr lauffähig ist ($> 100ms$ pro Iteration). Betrachtet man nun das Verhältnis zwischen der Verbesserung der Performanz (Abbildung 4.4) und der Verschlechterung der Laufzeit (Tabelle 4.2), so bildet das Setzen der erhaltenen Fehlersymbole auf 3 mit einem Zielgrade von 0 bei der Beschränkung auf quadratisches Sweeping einen guten Trade-off, weshalb diese Konfiguration als Voreinstellung für Sweeping in der Implementierung (Kapiter 4.7) vorgesehen ist.

	Erhaltene Fehlersymbole pro Iteration				
Gradreduktion	0	1	2	3	4
0	0.166	0.603	2.097	4.358	13.354
1	0.166	0.939	2.358	7.470	>100
2	0.166	1.645	5.750	20.339	>100
3	0.166	2.284	2.977	74.172	>100
4	0.166	2.658	3.622	>100	>100
	ms pro Iteration				

Tabelle 4.2: Durchschnittliche Dauer einer Iteration der Hénon-Abbildung in ms mit Taylormodelle der Größe $[0 \pm 2^{-10000}]$ bei festgelegter Präzision von 10000 Bits und 1000 Iterationen.

4.3 Cleaning

Durch die mehrstufige Intervallstruktur in den Koeffizienten der Polynome ist es mit Cleaning möglich, den Rechenfehler, beziehungsweise die Intervallbreite der Intervallgrenzen auf einer höheren Abstraktionsebene zu verwalten. Wie in Paragraph

3.4.1 beschrieben, wird das Intervall um die Intervallbreite des Mittelpunktes und des Radius' erweitert und schließt es somit komplett ein. Dies hat darüber hinaus den Effekt, dass sämtlicher entstandene Rechenfehler durch Splitting reduziert werden kann.

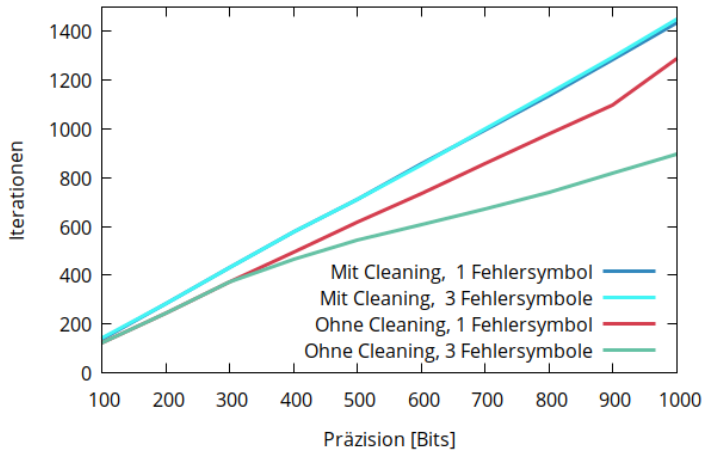


Abbildung 4.5: Iterationszahl der Hénon-Abbildung mit festem Radius der intialen Taylormodelle $x_0 = y_0 = [0 \pm 2^{-1000}]$ mit wachsender Präzision, um den Effekt des Cleanings beschreiben.

Abbildung 4.5 zeigt die Auswirkung der Anwendung von Cleaning bei initialen Taylormodellen x_0, y_0 der Breite 2^{-1000} mit wachsender Genauigkeit. Cleaning sorgt für ein langsameres Wachstum des Rechtecks $|x_n| \cdot |y_n|$ nach n Iterationen im Vergleich zur Berechnung ohne Cleaning. Bei einer höheren Anzahl der erhaltenen Fehlersymbole pro Iteration wird der Effekt noch deutlicher, da nun mit deutlich längeren Polynomen gerechnet und dementsprechend mehr arithmetische Operationen auf den Koeffizienten angewandt werden. So bleibt zwar mehr Information über Form der Taylormodelle erhalten, jedoch dominiert die Überschätzung auf der Zahlenebene das Ergebnis beim Verzicht auf Cleaning, während sich die Performanz im anderen Fall leicht verbessert.

4.4 Splitting

Durch Splitting wird ein Monom mit Intervallkoeffizienten in zwei Monome mit einem neuen, noch nicht verwendeten Fehlersymbol und Punktintervallkoeffizienten aufgeteilt. Für diese Housekeeping-Methode wurden zwei Möglichkeiten betrachtet. Zum Einen die in [BrKM15] verwendete Definition des neuen Fehlersymbols als Einheitsintervall:

$$[c \pm \varepsilon] \underset{split}{\rightsquigarrow} [c] + [\varepsilon] \cdot \lambda_{new}, \lambda_{new} \in [0 \pm 1] \tag{4.1}$$

Zum Anderen die Umlagerung des Radius ε in das neue Fehlersymbol:

$$[c \pm \varepsilon] \underset{split}{\rightsquigarrow} [c] + [1] \cdot \lambda_{new}, \lambda_{new} \in [0 \pm \varepsilon] \tag{4.2}$$

Beim Sweeping hat sich gezeigt, dass die Überschätzung des Taylormodells bei einer Priorisierung kleinerer Fehlersymbole, beziehungsweise beim Erhalt von Fehlersymbolen mit größerem Support-Space, langsamer wächst. In einer linearen Implementierung der Taylormodelle mit Variante 4.1 wird eine solche Ordnung durch einen Vergleich der Koeffizienten hergestellt, da jedes Monom maximal nur ein Fehlersymbol enthalten kann. Ein nichtlineares Monom kann jedoch von mehreren Fehlersymbolen abhängen, wodurch die Information über die Größe des Fehlersymobls nicht im Koeffizienten kodiert werden kann und zudem innerhalb eines Monoms sortiert werden muss. Daher eignet sich Herangehensweise 4.2 für die Implementierung nichtlinearer Taylormodelle besser, als die Verwendung von Einheitsintervallen, wie in Abbildung 4.6 an einem Bespiel gezeigt wird. Hier werden beide Varianten auf die Entwicklung der Überschätzung von zwei Taylormodellen untersucht, wobei mit der Definition des Fehlersymbols als reguläres Intervall 4.2 mehr Iterationen der Hénon-Abbildung berechnet werden können, bis die Fläche des Rechtecks einen Schwellenwert überschreitet.

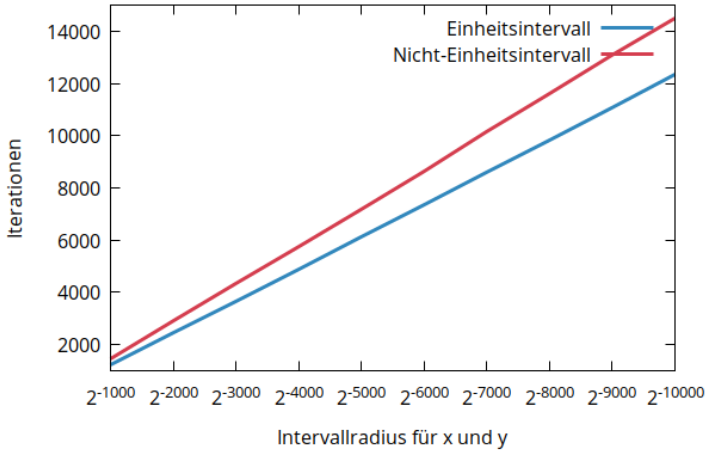


Abbildung 4.6: Iterationszahl der Hénon-Abbildung mit 10000 Bits fester Genauigkeit.

4.5 Initiales Taylormodell

Für die Hénon-Abbildung wird ein Punkt $(x_0, y_0) \in \mathbb{R}^2$ wiederum in den $\mathbb{R}^2$ abgebildet. x_0 und y_0 können nun als Taylormodelle verschiedener Form definiert werden:

$$[c] + [1] \cdot \lambda, \ \lambda \in [0 \pm \varepsilon] \tag{4.3}$$
$$[c] + [\varepsilon] \cdot \lambda, \ \lambda \in [0 \pm 1] \tag{4.4}$$
$$[0] + [c \pm \varepsilon] \cdot \lambda, \ \lambda \in [1] \tag{4.5}$$
$$[c \pm \varepsilon] \tag{4.6}$$

Die verschiedenen Taylormodelle wurden verwendet, um die Intervalle $[0 \pm 2^{-1000}]$, beziehungsweise $[0 \pm 2^{-10000}]$ in x_0 und y_0 darzustellen. Tabelle 4.3 zeigt, wieviele Iterationen mit der jeweiligen Definition möglich sind, bis die Fläche des Rechtecks von $|x| \cdot |y|$ einen Schwellenwert überschreitet.

	1000 Bit Präzision		10000 Bit Präzision	
Taylormodell	Iterationen	$\|x\| \cdot \|y\|$	Iterationen	$\|x\| \cdot \|y\|$
TM 4.3	1437	0.17	14506	0.78
TM 4.4	1227	0.49	12338	0.19
TM 4.5	1437	0.37	14505	0.17
TM 4.6	1437	0.35	14505	0.18

Tabelle 4.3: Berechnung der Fläche des Rechtecks mit verschiedenen Definitionen der initialen Taylormodelle für x und y bei festgelegter Präzision.

Bis auf Taylormodell-Definition 4.4 erreichen (beinahe) alle Berechnung dieselbe Iterationszahl, variieren jedoch in der Größe des Rechtecks, welches von x und y aufgespannt wird. Mit der Definition 4.3 als intiales Taylormodell wird für 10000 Bit Genauigkeit eine minimal höhere Iterationszahl und bei 1000 Bit Genauigkeit ein kleineres Rechteck, als in den anderen Varianten erreicht.

4.6 Anwendung der Parameter

In diesem Kapitel wurden mit verschiedenen Belegungen für die zu Verfügung stehenden Parameter in `HOTM` versucht, eine möglichst optimale Konfiguration im Hinblick auf die Eindämmung des Wrapping-Effektes zu finden. Die besten Ergebnisse werden hierbei erzielt für:

- **Sweeping** mit ausschließlich quadratischem Sweeping zum Grade 0,
- **Cleaning** und **Splitting** mit einem Grenzwerten orientiert an der Genauigkeit,
- **Initialen Taylormodellen** für ein Intervall $[c \pm \varepsilon]$ der Form $[c] + [1] \cdot \lambda,\ \lambda \in [0 \pm \varepsilon]$,
- Erhalt von 3 **Fehlersymbolen**.

Abbildung 4.7 zeigt die Entwicklung bei Berechnungen der Hénon-Abbildung. Jeweils zu sehen sind (1) der Abstand der x-Koordinate zweier Punkte $a = (0, 0)$ und $b = (2^{-p}, 2^{-p})$ in Farbe, (2) die Breite des Intervalls $|x_n|$ für $(x_0, y_0) = ([0 \pm 2^{-(p-1)}], [0 \pm 2^{-(p-1)}])$ in grau und (3) die Breite des ausgewerteten Taylormodells $|x_n|$ für zwei Taylormodelle $x_0 = [0] + [1] \cdot \lambda_0, \lambda_0 \in [0 \pm 2^{-(p-1)}]$ und $y_0 = [0] + [1] \cdot \lambda_1, \lambda_1 \in [0 \pm 2^{-(p-1)}]$ in schwarz. Der Verlauf der unabhängigen Punkte (1) stellt eine lokale Annäherung an den Lyapunov Exponenten dar und damit eine untere Schranke an die Ausbreitung eines Rechtecks pro Iteration der Hénon-Abbildung dar, indem sie dessen Eckpunkte markieren. Es ist deutlich erkennbar, dass sich das Taylormodell (3) langsamer ausbreitet und damit näher an der Schranke bewegt, als die Berechnung mit reiner Intervallartihmetik (2).

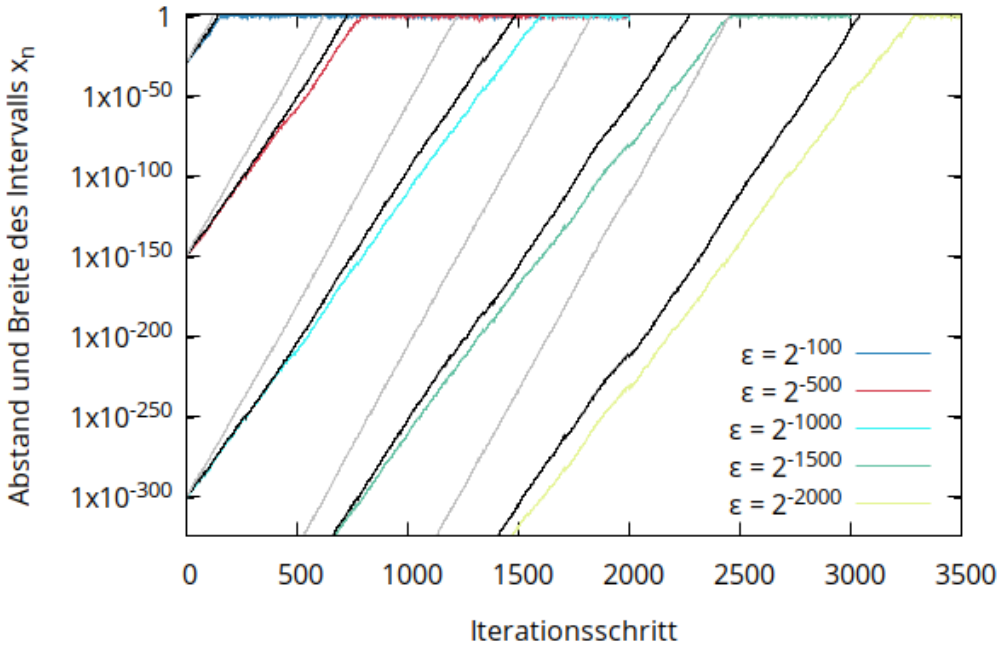


Abbildung 4.7: Lokale Annäherung an den Lyapunov Exponenten der Hénon-Iteration mit Intervallen der Breite $|x_0| = \varepsilon$, und den Parametern $a = 1.4, b = 0.3$ im Vergleich zur Entwicklung des Abstandes zweier Punkte mit Abstand ε

4.7 Umsetzung

Zwei Implementierungen der Hénon-Abbildung für die Parameter $a = 1.4$ und $b = 0.3$ sind in den Listings 4.1 und 4.2 zu sehen. Es werden drei verschiedenene Zahlentypen verwendet, die jedoch alle auf den `iRRAM-REALs` aufbauen: *High Order Taylor Modell* `HOTM`, *Real Number* `num` und *Real Number Interval* `numint`. Die jeweils definierte Funktion `compute()` in Zeile 4 stellt den in der `iRRAM` verwendeten Rahmen für Iterationen, beziehungsweise Erhöhungen der Genauigkeit der `REALs` dar. Mit dem Funktionsaufruf `HOTM::init()` in Zeile 7 werden die zuvor in diesem Kapitel beschriebenen Grenzwerte und Parameter für die Housekeeping-Methoden mit Werten initialisiert, die experimentell die beste Performanz lieferten. Die Parameter a und b werden in Zeile 10 als `num` (`iRRAM::REAL`) instanziiert. Die Durchführung der Housekeeping-Methoden ist ein Überschreiben des Zuweisungsoperators '=' für die `HOTM`-Taylormodelle realisiert. Während der Berechnung in den Zeilen 12 und 13 können sich die Taylormodelle aufblähen und werden dann bei der Zuweisung reduziert. In den Zeilen 14 und 15 findet kein Housekeeping mehr statt, gesteuert durch einen Parameter, der den Zustand des Taylormodells beschreibt [3].

Um die Anpassung der Parameter der `HOTM` zu vereinfachen stehen verschiedene Möglichkeiten zur Verfügung:

[3] Operationen auf dem Taylormodell markieren es als 'not cleaned'. Dieser Wert wird dann durch die Anwendung von Housekeeping zurückgesetzt.

Initiale Taylormodelle

In der linken Implementierung 4.1 werden x und y als Punktintervalle, also als Taylormodelle mit lediglich einem Kernel definiert:

$$\texttt{HOTM::x(0)} \rightsquigarrow x \coloneqq [0]$$
$$\texttt{HOTM::y(0)} \rightsquigarrow y \coloneqq [0]$$

Unter Verwendung des Konstruktors auf der rechten Seite wird aus der Eingabe eines Intervalls mit Mittelpunkt c und Radius ε je ein Taylormodell mit einem Fehlersymbol, welches ε enthält und dem Kernel als Punktintervall c:

$$\texttt{HOTM::x(0,0.001)} \rightsquigarrow x \coloneqq [0] + [1] \cdot \lambda_0,\ \lambda_0 \in [0 \pm 0.001]$$
$$\texttt{HOTM::y(0,0.001)} \rightsquigarrow y \coloneqq [0] + [1] \cdot \lambda_1,\ \lambda_1 \in [0 \pm 0.001]$$

Anzahl der zu erhaltenden Fehlersymbole

Die Initialisierungsfunktion `init()` akzeptiert eine `int` ≥ 0 als Eingabe. Damit kann die Anzahl der zu erhaltenden Fehlersymbole pro Taylormodellangepasst werden, wie in Listing 4.2 Zeile 9. Der Standardwert liegt bei 3. Außerdem besteht die Möglichkeit, diesen Wert für jedes Taylormodell individuell zu bestimmen (Listing 4.2 Zeile 11). Ist dieser Wert nicht gesetzt, so wird er aus dem `init()` Aufruf abgeleitet.

Manuelles Housekeeping

Sollen die Housekeeping-Methoden nur für bestimmte Taylormodelle oder zu einem anderen Zeitpunkt, als bei der Zuweisung durchgeführt werden, so wird die Funktion `HOMT::kousekeep` aufgerufen. Ein Aufruf, wie in Listing 4.2 Zeile 16, schaltet das Housekeeping während der Zuweisung aus. Zudem beim Entfernen bis auf die m-größten Fehlersymbole nun für alle im Aufruf enthaltenen Taylormodelle durchgeführt, statt einzeln. Das bedeutet, dass nach diesem Aufruf in allen Taylormodellen dieselben Fehlersymbole übrig bleiben.

```
1  #include "iRRAM.h"
2  #include "hoTM.h"
3  using namespace iRRAM;
4
5  void compute(){
6   int n;
7   cin >> n;
8
9   HOTM::init();
10  HOTM x(num(0));
11  HOTM y(num(0));
12  HOTM x_new, y_new;
13
14  num a(1.4), b(0.3);
15  for (int i = 0; i < n; i++) {
16
17    x_new = 1+y-a*(x*x);
18    y_new = b*x;
19    x = x_new;
20    y = y_new;
21  }
22  cout<<"x:"<< numint(x) <<"\n";
23  cout<<"y:"<< numint(y) <<"\n";
24 }
```

Listing 4.1:
Implementierung mit Punktintervallen

```
1  #include "iRRAM.h"
2  #include "hoTM.h"
3  using namespace iRRAM;
4
5  void compute(){
6   int n;
7   cin >> n;
8
9   HOTM::init(2);
10  HOTM x(num(0), num(0.001));
11  HOTM y(num(0), num(0.001), 0);
12  HOTM x_new, y_new;
13
14  num a(1.4), b(0.3);
15  for (int i = 0; i < n; i++) {
16    HOTM::housekeep(x,y);
17    x_new = 1+y-a*(x*x);
18    y_new = b*x;
19    x = x_new;
20    y = y_new;
21  }
22  cout<<"x:"<< x <<"\n";
23  cout<<"y:"<< y <<\n";
24 }
```

Listing 4.2:
Implementierung mit Intervallen

5. Fazit

5.1 Diskussion

Der Schritt von linearen zu nichtlinearen Taylormodellen erhöht in der Praxis zwar leicht deren Potential, Rundungsfehler zu kontrollieren, jedoch erhöht sich die Komplexität der Berechnung und der Strukturen um ein vielfaches, was sich besonders in der Laufzeit, als auch im Aufwand niederschlägt. Dadurch, dass jede Zahl der `HOTM` von einem Intervall dargestellt wird, beziehungsweise eine `iRRAM-REAL` ist, sorgt jede arithmetische Operation wegen der Eigenschaften der Intervallarithmetik zu einer Überschätzung. Wachsen nun die Polynome in ihrer Länge, kann zwar das Anhängigkeitsproblem der Variablen angegangen werden, jedoch erhöht sich auch die Anzahl der ausgeführten Operationen, was letztendlich die Ausdehnung der Taylormodelle dominiert, wie in Kapitel 4 zu sehen ist.

Durch den vielschichtigen Aufbau des Zahlentyps `HOTM`, ist es meist schwierig zu sagen, ob eine Konfiguration optimal ist, da sehr viele Faktoren eine Rolle spielen. Um beispielsweise die Ausdehnung eines Rechtecks zu untersuchen, mussten starke Einschränkungen an den unterliegenden `iRRAM`-Iterationen vorgenommen werden. Darunter fällt die Fixierung der Genuaigkeit der `REAL`s, die eigentlich dynamisch zu Laufzeit angepasst wird. Ohne eine solche Fixierung und andere Beschränkungen, was nicht möglich, die experimentellen Ergebnisse klar zu deuten und die Faktoren voneinander zu trennen. Diese Einschränkungen entsanden aus einer längeren Auseinandersetzung mit dem zu Grunde liegenden Problem und dem Wissen darüber, wann ein Rechteck bestimmter Größe, bei einer gewissen Genauigkeit die Fangzone in der Hénon-Abbildung verlässt. Um die `HOTM` als generischen Zahlentypen für Intervallarithmetik verwenden zu können, sind der Zeit noch solche Informationen und entsprechende Anpassungen nötig, da Berechnungen in einigen Fällen sonst nicht nmöglich sind.

5.2 Ausblick

In dieser Arbeit wurde eine grundlegende Implementierung für das Rechnen mit nichtlinearen Taylormodellen auf den reellen Zahlen beschrieben und angefertigt. An verschiedenen Stellen ist es jedoch möglich und teils notwendig, die Arbeit fortzuführen.

5.2.1 Partitionierung

5.2.2 Anwendungen

Gefahr des Overfittings auf Henon

5.2.3 Laufzeitoptimierung

5.2.4 Sweeping-Heuristiken

5.2.5 Schnittstellen

Die Implementierung der `iRRAM`, beziehungsweise der `HOTM` bietet ein sehr hohes Abstrantionsniveau für die komplexen Berechnungen, die damit ausgeführt werden, sodass die reellen Zahlen und die nichtlinearen Taylormodelle als reguläre Zahlentypen verwendbar sind. Neben der reinen Entwicklungsarbeit, stellt allerdings auch die Verbreitung der Ideen und der Programme als solches ein Problem dar, da sich deren Verwendung auf `C++`-Programme beschränkt. Eine Schnittstelle zu anderen Programmiersprachen, wie beispielsweise in der `Python`-Bibliothek `numpy`, eigentlich in `FORTRAN` implementiert, würde die Verwendung der populärsten Programmiersprache[1] eröffnen.

Für die Umsetzung wäre eine mögliche Lösung, die `iRRAM`, beziehungsweise `HOTM` in einem eigenen Thread laufen zu lassen und mit asynchronen Anfragen Ergebnisse für Berechnungen zu erhalten.

[1] https://de.statista.com/infografik/22669/popularitaet-von-programmiersprachen/ (Stand 26.03.2021)

Literaturverzeichnis

[BrHW08] V. Brattka, P. Hertling und K. Weihrauch. *A Tutorial on Computable Analysis*, S. 425–491. Springer New York, New York, NY. 2008.

[BrKM15] F. Braußе, M. V. Korovina und N. T. Müller. Using Taylor Models in Exact Real Arithmetic. In I. S. Kotsireas, S. M. Rump und C. K. Yap (Hrsg.), *Mathematical Aspects of Computer and Information Sciences - 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers*, Band 9582 der *Lecture Notes in Computer Science*. Springer, 2015, S. 474–488.

[Hén76] M. Hénon. A two-dimensional mapping with a strange attractor. *Comm. Math. Phys.* 50(1), 1976, S. 69–77.

[MaBe01] K. Makino und M. Berz. Higher order verified inclusions of multidimensional systems by taylor models. *Nonlinear Analysis: Theory, Mehtods and Applications* 47(5), 2001, S. 3514–3503.

[MdOCC20] V. Martins de Oliveira, D. Ciro und I. L. Caldas. Dynamical trapping in the area-preserving Hénon map. *The European Physical Journal Special Topics* 229(8), May 2020, S. 1507–1516.

[Moor79] R. E. Moore. *Methods and applications of interval analysis*. SIAM studies in applied mathematics. SIAM. 1979.

[MoPe07] M. B. Monagan und R. Pearce. Polynomial Division Using Dynamic Arrays, Heaps, and Packed Exponent Vectors. In V. G. Ganzha, E. W. Mayr und E. V. Vorozhtsov (Hrsg.), *Computer Algebra in Scientific Computing, 10th International Workshop, CASC 2007, Bonn, Germany, September 16-20, 2007, Proceedings*, Band 4770 der *Lecture Notes in Computer Science*. Springer, 2007, S. 295–315.

[Müll00] N. T. Müller. The iRRAM: Exact Arithmetic in C++. In J. Blanck, V. Brattka und P. Hertling (Hrsg.), *Computability and Complexity in Analysis, 4th International Workshop, CCA 2000, Swansea, UK, September 17-19, 2000, Selected Papers*, Band 2064 der *Lecture Notes in Computer Science*. Springer, 2000, S. 222–252.

[Müll09] N. Müller. Enhancing imperative exact real arithmetic with functions and logic. 2009.

[PlBr89] P. Plaschko und K. Brod. *Deterministisches Chaos — Eine Einführung*, S. 359–424. Springer Berlin Heidelberg, Berlin, Heidelberg. 1989.

[Span10] C. Spandl. Computational Complexity of Iterated Maps on the Interval (Extended Abstract). In X. Zheng und N. Zhong (Hrsg.), *Proceedings Seventh International Conference on Computability and Complexity in Analysis, CCA 2010, Zhenjiang, China, 21-25th June 2010*, Band 24 der *EPTCS*, 2010, S. 139–150.

[WiKm20] T. Williams, C. Kelley und many others. Gnuplot 5.4: an interactive plotting program. http://gnuplot.sourceforge.net/, 2020.

[Yan98] T. Yan. The Geobucket Data Structure for Polynomials. *J. Symb. Comput.* 25(3), 1998, S. 285–293.

6. Anhang

Algorithmen für Geobuckets

Algorithm 4: Polynommultipliaktion mit Geobuckets

input: Polynome p_1, p_2

```
1  // Initialize
2  n ← #p_1, m ← #p_2 ;
3  p ← [ ] ;
4  Allocate buckets with space for polynomials: {2m, 4m, ..., 2^⌈log_2(n)⌉−1 m};
5  i ← 0;
6  // Main Loop
7  while i < n do
8  |   p ← p_1[i] · p_2;
9  |   if i < (n − 1) then
10 |   |   p ← p + p_1[i + 1] · p_2;
11 |   i ← i + 2;
12 |   j ← 0;
13 |   while buckets[j].not_empty() do
14 |   |   p ← p + buckets[j] ;
15 |   |   buckets[j] ← [ ];
16 |   |   j ← j + 1;
17 |   end
18 |   if i < n then
19 |   |   buckets[j] ← p;
20 |   |   p ← [ ] ;
21 end
22 // Merge each bucket into the final polynomial
23 foreach bucket ∈ buckets do
24 |   p ← p + bucket;
25 end
26 return p
```

Algorithm 5: Addition zweier Polynome

input: Polynome p_1, p_2

$p \leftarrow (), i \leftarrow 0, j \leftarrow 0$;
while $i < \#p_1$ ***and*** $j < \#p_2$ **do**
  $max \leftarrow max\_errorsymbols(p_1[i], p_1[j])$;
  if $max = left$ **then**
    $p.append(p_1[i])$;
    $i \leftarrow i + 1$;
  else if $max = right$ **then**
    $p.append(p_2[j])$;
    $j \leftarrow j + 1$;
  else
    // $max = equal$
    $p.append(p_1[i] + p_1[j])$;
    $p_1.remove(i)$;
    $i \leftarrow i + 1; j \leftarrow j + 1$;
end
Append the rest to p;
return p

Algorithm 6: Vergleich Variablen zweier Monome

input: Zwei Listen von Fehlersymbolen ess_1, ess_2

$size \leftarrow max\{length(ess_1), length(ess_2)\}$;
$i \leftarrow 0$;
while $i < size$ **do**
  if $i \geq length(ess_1)$ ***or*** $ess_1[i] \succ ess_2[i]$ **then**
    return right
  else if $i \geq length(ess_2)$ ***or*** $ess_1[i] \prec ess_2[i]$ **then**
    return left
end
return equal

Details der Implementierung

```
[
 {
   "comment": "Error 100",
   "rel": 1,
   "linear": 0,
   "keep": 2,
   "error_in_lambda": true,
   "strategy": "SQUARE_ONLY",
   "iter": 1000,
   "prec": 50,
   "error": 1000,
   "decimals": 20,
   "sweep_to": 2,
   "micro_thresh": 1,
   "macro_thresh": 1,
   "a": 1.4,
   "b": 0.3,
   "support_space": [
     {
       "point": false,
       "mid": "0",
```

```
        "rad": "e"
      },
      {
        "point": false,
        "mid": "0",
        "rad": "e"
      }
    ],
    "x": {
      "kernel": {
        "point": true,
        "mid": "1",
        "rad": "0"
      },
      "monomials": [
        {
          "point": true,
          "mid": "1",
          "rad": "0",
          "lambdas": [{
            "exp": 1,
            "index": 0
          }]
        }
      ]
    },
    "y": {
      "kernel": {
        "point": true,
        "mid": "0.1",
        "rad": "0"
      },
      "monomials": [
        {
          "point": true,
          "mid": "1",
          "rad": "0",
          "lambdas": [{
            "exp": 1,
            "index": 1
          }]
        }
      ]
    }
  }
]
```

Listing 6.1: Bespielkonfiguration

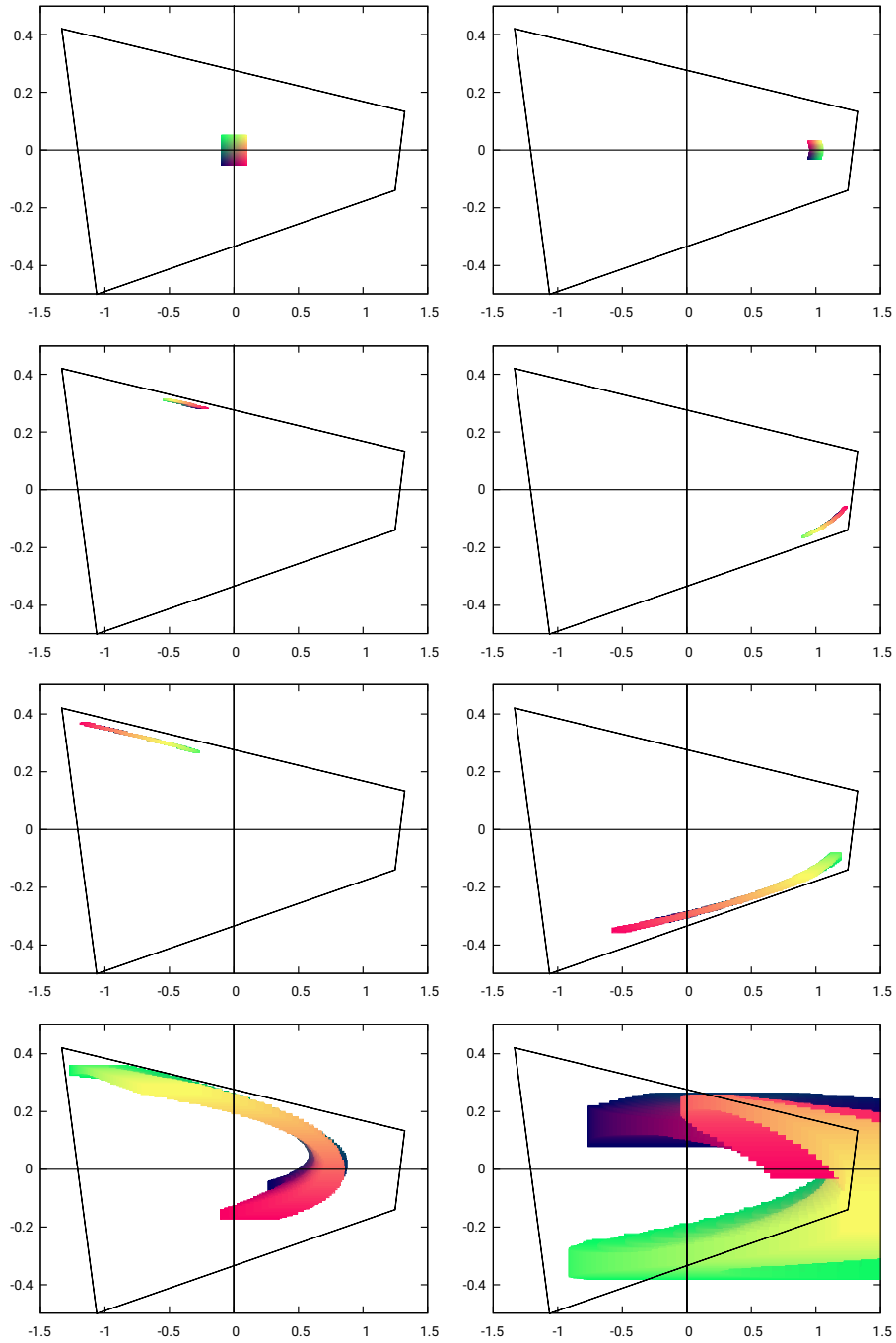


Abbildung 6.1: Mehrere Iterationen mit Farbkodierung

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Bachelor-/Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vor-gelegt. Sie wurde bisher auch nicht veröffentlicht.

Trier, den xx. Monat 20xx