

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

1	Einleitung	1
1.1	Motivation	1
1.2	Problemstellung	1
1.3	Zielsetzung	1
1.4	Gliederung/Aufbau der Arbeit	1
2	Grundlagen	2
2.1	Hénon-Abbildung	2
2.2	Taylormodell	4
2.2.1	Arithmetische Operationen auf Taylormodellen	6
2.2.2	Spezielle Operationen auf Taylormodellen	6
2.3	Lyapunov Exponent	6
3	Analyse	7
4	Entwurf / Konzeption	9
5	Implementierung	10
5.1	iRRAM	10
5.1.1	Präzision	11
5.1.2	Darstellung der Taylormodelle	11
5.2	Intervallarithmetik	12
5.3	Polynommultiplikation	12
5.4	Taylormodelle	14
5.4.1	Housekeeping-Methoden	14
6	Evaluation	16
7	Diskussion und Ausblick	17
8	Anhang	18
	Literaturverzeichnis	20

Abbildungsverzeichnis

2.1	Hénon-Abbildung: Einfache Evolution	3
2.2	Hénon-Abbildung: Attraktor	4
2.3	Linear und nichtlineare Taylormodelle: Vergleich	5
5.1	Ebenen der Polynomdarstellung mit REALs	11
5.2	Naive Multiplikation gegen Multiplikation mit Geobuckets	14

Tabellenverzeichnis

Listings

5.1	Beispielaufruf der Sweeping-Routine	15
-----	---	----

1. Einleitung

Die Einleitung besteht aus der Motivation, der Problemstellung, der Zielsetzung und einem ersten Überblick über den Aufbau der Arbeit.

1.1 Motivation

Warum ist das zu bearbeitende Themengebiet spannend und relevant?

1.2 Problemstellung

Welches Problem/welche Probleme können in diesem Themengebiet identifiziert werden?

1.3 Zielsetzung

Was ist das Ziel der Arbeit. Wie soll das Problem gelöst werden?

1.4 Gliederung/Aufbau der Arbeit

Was enthalten die weiteren Kapitel? Wie ist die Arbeit aufgebaut? Welche Methodik wird verfolgt?

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 recht einfachen Abbildung zu untersuchen.

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 durch eine dreidimensionale Differenzialgleichung entstehenden Lorenz-Attraktor, 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 große 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.

Rotieren

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

Im letzten Schritt werden die Achsen vertauscht und somit die Fläche um rotiert.

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 x^2 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.

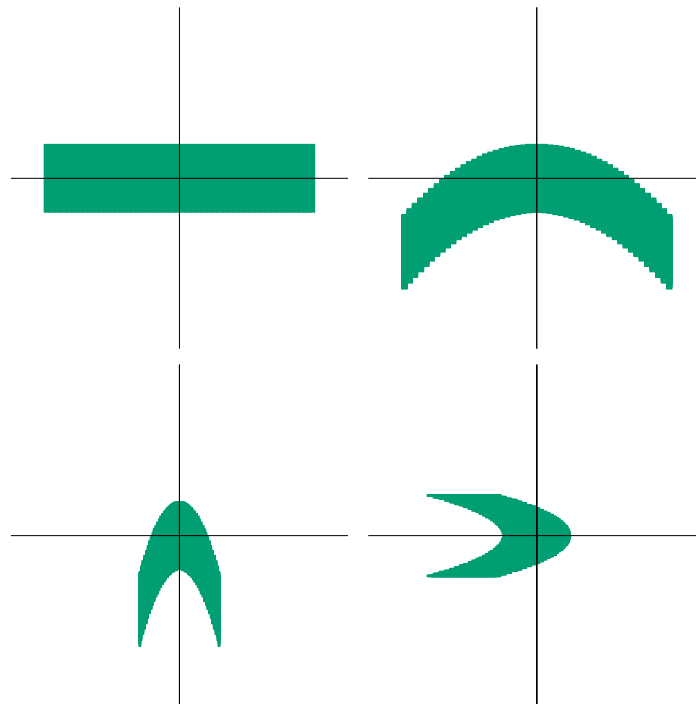


Abbildung 2.1: Einfache Evolution der Hénon Abbildung, aufgeteilt in das initiale Rechteck und die drei Zwischenschritte.

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, 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 dieser 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.

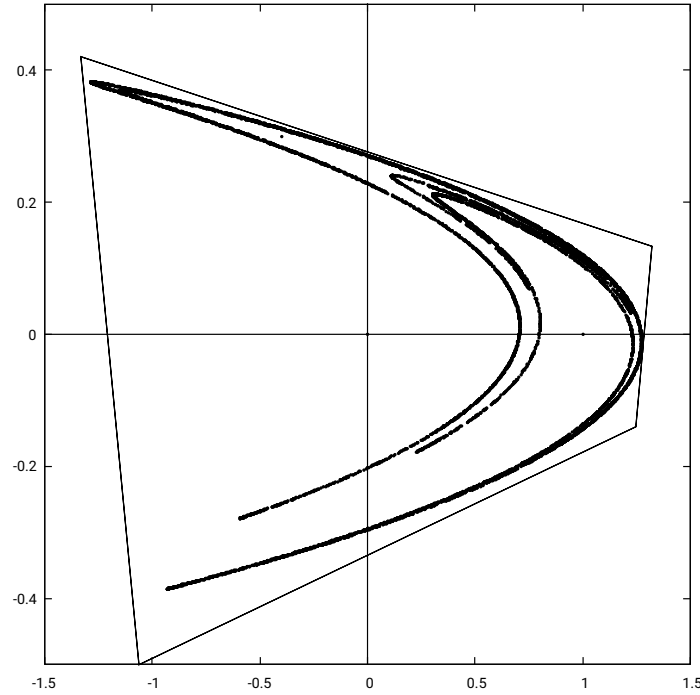


Abbildung 2.2: Seltsamer Attraktor der Hénon-Abbildung für $a = 1,4$ und $b = 0.3$ mit 10000 Punkten.

2.2 Taylormodell

Ein Taylormodell im Sinne der Erweiterung von [BrKM15] des Grundmodells von [MaBe01] besteht aus einem Polynom p mit $k \in \mathbb{N}$ Variablen, geschlossenen Intervallen als Koeffizienten. Darin enthalten ist immer ein Monom c_0 des Grades 0, der das Kernintervall (*Kernel*) des Taylormodells darstellt. Eine Variable oder ein Fehlersymbol λ_i aus dem Vektor $\lambda = (\lambda_1, \dots, \lambda_k)$ steht für einen Wert aus dem dazugehörigen Supportintervall aus $S = (s_1, \dots, s_k)$ mit $\lambda_i \in s_i$ und wird dazu verwendet, unbekannte Werte, Rechenungenauigkeiten und funktionale Abhängigkeiten innerhalb eines oder zwischen mehreren Taylormodellen abzubilden. Die verwendeten Intervalle $c_n = [\tilde{c}_n \pm \varepsilon_n] \subseteq \mathbb{R}$ haben in `hotm` reelle Endpunkte und stellen mit $c'_n = [\tilde{c}_n \pm 0]$ auch Punktintervalle dar. Mit einem so definierten Taylormodell $T = \sum_n c_n \lambda^n$ 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.

Des Weiteren können mit Taylormodellen im in dieser Arbeit behandelten zweidimensionalen Fall auch komplexere Flächen als achsenparallele Rechtecke mit Hilfe von Intervallen beschrieben werden.

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

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 kann mit zwei solchen *nichtlinearen Taylormodellen* im Zweidimensionalen komplexere Strukturen, wie Kurven höherer Ordnung beschrieben werden.

Abbildung 2.3 zeigt lineare und eine nichtlineare Taylormodelle für

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

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

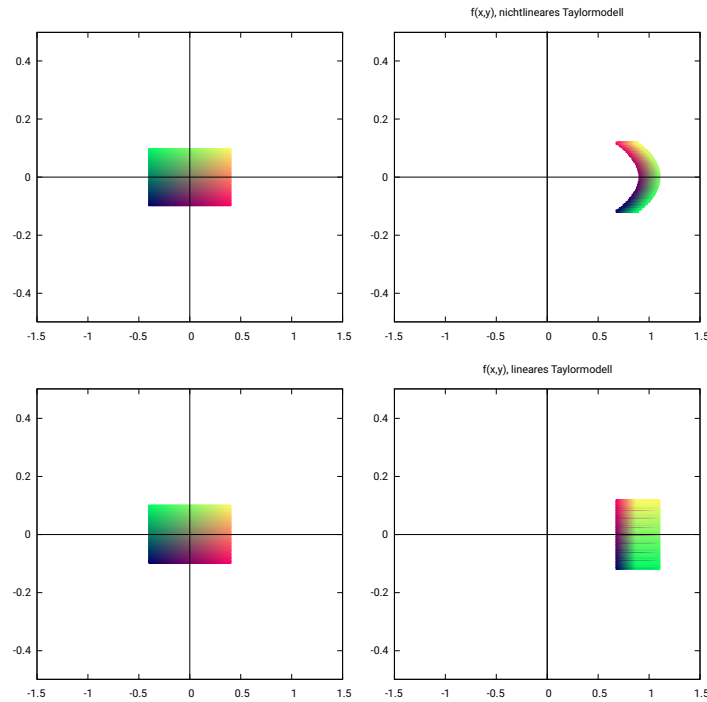


Abbildung 2.3: Darstellung nichtlinearer und linearer Taylormodelle, die ein Rechteck beschreiben und deren Abbildung durch die Funktion $f(x, y) = (y + 1 - 1.4x^2, 0.3x)$. Die Farbkodierung indiziert den Ursprung der abgebildeten Flächen.

Sowohl die linearen, als auch die nichtlinearen Taylormodelle umschließen den korrekten Bereich der Funktionswerte, allerdings ist die abgebildete Fläche der Nichtlinearen näher an der Fläche, die sich ergäbe, betrachtet 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. Die Division erfordert

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 Fehler-symbole durch das entsprechende Intervall aus S ersetzt wird:

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

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 der Breite des Koeffizienten entspricht:

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

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

2.3 Lyapunov Exponent

3. Analyse

Werden die Initialwerte für Iterationen der Hénon-Abbildung (x_0, y_0) als Taylormodelle mit Intervallen 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.

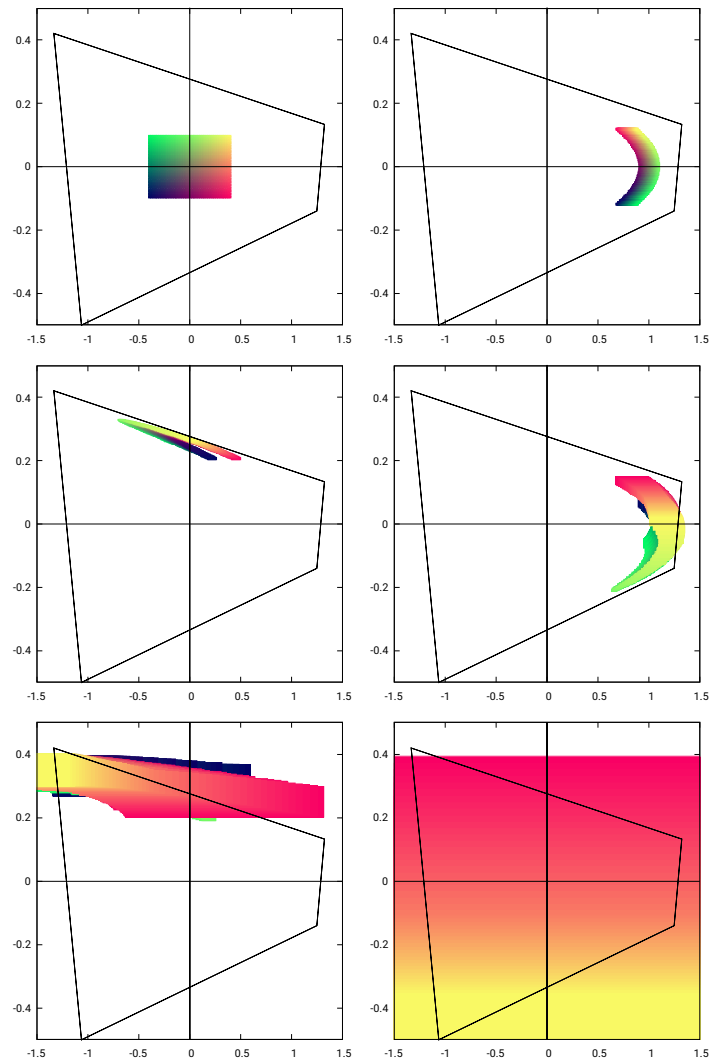


Abbildung 3.1

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 3.1 zeigt pro Graph eine Iteration mit

$$\begin{aligned}x_0 &= 0 + 1 \cdot \lambda_1 & (\lambda_1 \in [0 \pm 0.4]) \\y_0 &= 0 + 1 \cdot \lambda_2 & (\lambda_2 \in [0 \pm 0.1])\end{aligned}$$

und der Fangzone als schwarzes Tetragon. Es ist erkennbar, dass das Rechteck nach wenigen Iterationen die Fangzone verlässt und einige Intervalle ein starkes Rauschen verursachen. Um dem Wachstum der Intervalle entgegenzuwirken und die Berechnung der Hénon-Abbildung auch für solche Flächen 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 Intervalle, 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 Taylormodelle nun 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?

4. Entwurf / Konzeption

5. Implementierung

5.1 iRRAM

Die Software-Bibliothek `iRRAM` [Müll09] basiert auf Intervallen als Zahlentyp, um diese mit einer beliebigen Genauigkeit darstellen zu können. 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, 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 nicht die gesamte Berechnung in jeder Iteration wiederholt wird, 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 gegebenenfalls 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 Präzision 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`, also auch wiederum jeweils als Intervall mit Wert und Fehler, wie in Grafik 5.1 zu sehen ist. Im Vergleich zu den `mpq-RATIONALS` erhöht sich hier zwar die Komplexität deutlich, allerdings lassen sich Ungenauigkeiten 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. Ein ähnlicher Effekt sollte sich durch Rundung auch mit den mpq-RATIONALS erreichen lassen.

Das Verlagern der Ungenauigkeit (e in der Grafik 5.1) auf den Wert des Radius' (v in der Grafik) (Cleaning oder auch *Micro-Housekeeping*) verringert die Intervallbreite und erzeugt Punktintervalle auf der Zahlenebene, allerdings werden die Intervalle auf der Intervallebene breiter. Hier greift dann wiederum der Polish-Mechanismus, der auf Polynomebene Monome hinzufügt, um aus den zu groß gewordenen Intervallen wiederum Punktintervalle zu machen (Splitting oder auch *Macro-Housekeeping*). So wird der Rechenfehler von der untersten Ebene bis zur Polynomebene propagiert. Diese Housekeeping-Funktionen werden durch Parameter gesteuert, die bestimmen, ab wann ein Intervall zu breit ist und die jeweilige Prozedur angewandt werden soll.

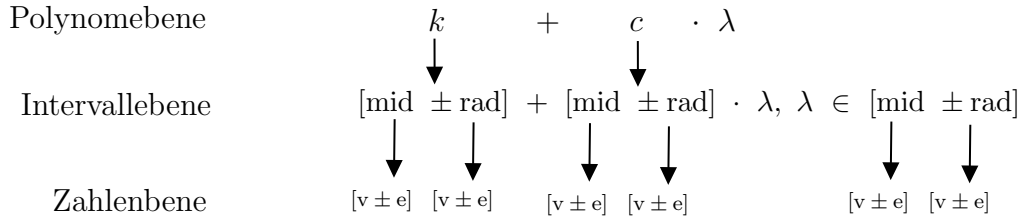


Abbildung 5.1: Ebenen der Polynomdarstellung mit REALs

5.1.1 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 `prec_policy`, ob 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

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

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 und die Skalierung zu aufwändig ist.

5.1.2 Darstellung der Taylormodelle

Die grafischen Darstellungen der Taylormodelle in dieser Arbeit wurden mit `gnuplot` [?] erstellt und zeigen je eine Übermenge des tatsächlichen Taylormodells. Eine Darstellung eines Taylormodells entsteht, indem für jedes der k Fehlersymbol aus $\lambda = (\lambda_1, \dots, \lambda_k)$ 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 $S = (s_1, \dots, s_k)$ in jeweils a Intervalle geteilt und für jede Kombination mit eingesetzten Werten ein Rechteck gezeichnet wird. So ergeben sich für Taylormodelle mit k Variablen a^k Rechtecke.

Für T

5.2 Intervallararithmetik

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

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

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

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

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

Potenzfunktion

Die Potenzfunktion unterscheidet zwischen geradem und ungeradem Exponenten.

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

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.

5.3 Polynommultiplikation

In einer linearen Implementierung der Taylormodelle wird bei der Multiplikation immer eines der Fehlersymbole gesweeppt, sodass die Länge eines Polynoms nicht über die Dimension hinausgeht und somit der Schwerpunkt bei der Intervallararithmetik liegt. Lässt man allerdings höhere Ordnungen zu, werden auch die Polynome länger und ein weiteres Problem tritt auf. Um die oben (5.3) 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:

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

So ergibt sich für die naive Multiplikation eine quadratische Laufzeit in O -Notation von $O(nm + (nm)^2)$

Für effizientere Polynommultiplikation existieren verschiedene Algorithmen. Viele der schnellen bekannten Algorithmen basieren auf der Schnellen-Fourier-Transformation, wobei die Polynome in Stützvektoren und wieder zurück umgewandelt werden. Da es sich bei den Koeffizienten den Monome um Intervalle mit möglicherweise unendlichen Schranken handelt, ist das Rückführen eines Stützvektors nicht immer ohne Weiteres möglich.

Ein weiterer Ansatz besteht darin, die Anzahl der benötigten Monomvergleiche zu reduzieren, wenn die Monome einsortiert werden müssen. Betrachtet man die Summe dreier Polynome $p_1 + p_2 + p_3$ mit $\#p_1 \gg \#p_2 = \#p_3 = 1$, benötigt das Aufsummieren von links nach rechts $2\#p_1 + 1$ Vergleiche, da mit dem oben verwendeten Additionsverfahren zweifach in p_1 eingefügt wird. 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. Mit Geobuckets, von Yan ([Yan98]) eingeführt und Monagan und Pearce ([MoPe07]) für die Multiplikation angepasst, 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 1 zeigt die in `hotm` implementierte Version der Geobucketmultiplikation. Durch den Teile und Herrsche Ansatz, kann die Multiplikation mit Geobuckets in $O(nm \log nm)$ durchgeführt werden.

Wird wie in [Yan98] eine Ordnung \succ auf den Monomen definiert, können die Polynome sortiert werden.

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

Diese Ordnung ist partiell, da gleiche Monome, beziehungsweise Monome mit gleichen Fehlersymbolen nicht vergleichbar sind. Tritt dieser Fall während der Addition zweier Polynome auf, bedeutet das, dass die Koeffizienten beider Monome addiert werden und das so entstandene Monom in das Polynom aufgenommen wird. Haben die verglichenen Monome die selbe Anzahl an Variablen, entscheidet die Größe des Exponenten des Fehlersymbols mit dem kleinsten Index. Algorithmus 2 implementiert den Vergleich zweier Monome, beziehungsweise derer Variablen.

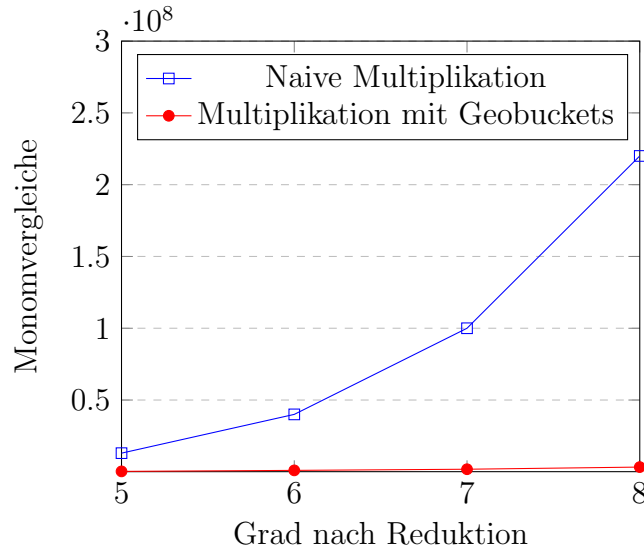


Abbildung 5.2: Naive Multiplikation gegen Multiplikation mit Geobuckets

5.4 Taylormodelle

Taylormodelle bestehen in `hotm` aus einer List von nach der Ordnung \succ (siehe 5.3) sortierten Monomen. Für die Supportintervalle der Fehlersymbole wird eine statische Liste (`static std::list`), für die Grenzwerte der Housekeeping-Methoden und den Index neuer Fehlersymbole jeweils statische Variablen angelegt.

5.4.1 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 bestimmen, werden die Zahlen mit Funktionen der `iRRAM` in rationale Approximationen zerlegt.

Cleaning

Wie oben beschrieben, ergibt die Verwendung der `iRRAM-REALs` als Intervallgrenzen zweistufige Intervalle mit der Zahlen- und Intervallebene (siehe Abbildung 5.1). 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:

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

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:

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

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:

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

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 5.2). 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 5.1: Beispielaufruf der Sweeping-Routine

6. Evaluation

7. Diskussion und Ausblick

(Keine Untergliederung mehr)

8. Anhang

Algorithm 1: Polynommultiplikation mit Geobuckets

input: Polynome p_1, p_2

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

Algorithm 2: Vergleich Variablen zweier Monome

input: Zwei Listen von Fehlersymbolen ess_1, ess_2

```

1  $size \leftarrow \max\{length(ess_1), length(ess_2)\};$ 
2  $i \leftarrow 0;$ 
3 while  $i < size$  do
4   if  $i \geq length(ess_1)$  or  $ess_1[i] \succ ess_2[i]$  then
5     return right
6   else if  $i \geq length(ess_2)$  or  $ess_1[i] \prec ess_2[i]$  then
7     return left
8 end
9 return equal

```

Literaturverzeichnis

- [BrKM15] F. Brauße, 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.
- [MaBe01] K. Makino und M. Berz. Higher order verified inclusions of multidimensional systems by taylor models. *Nonlinear Analysis: Theory, Methods and Applications* 47(5), 2001, S. 3514–3503.
- [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.
- [Müll09] N. Müller. Enhancing imperative exact real arithmetic with functions and logic. 2009.
- [] 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.

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