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

1. Einleitung

Die Einleitung besteht aus der Motivation, der Problemstellung, der Zielsetzung und einem erster 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.

Konviergieren 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 dir 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.

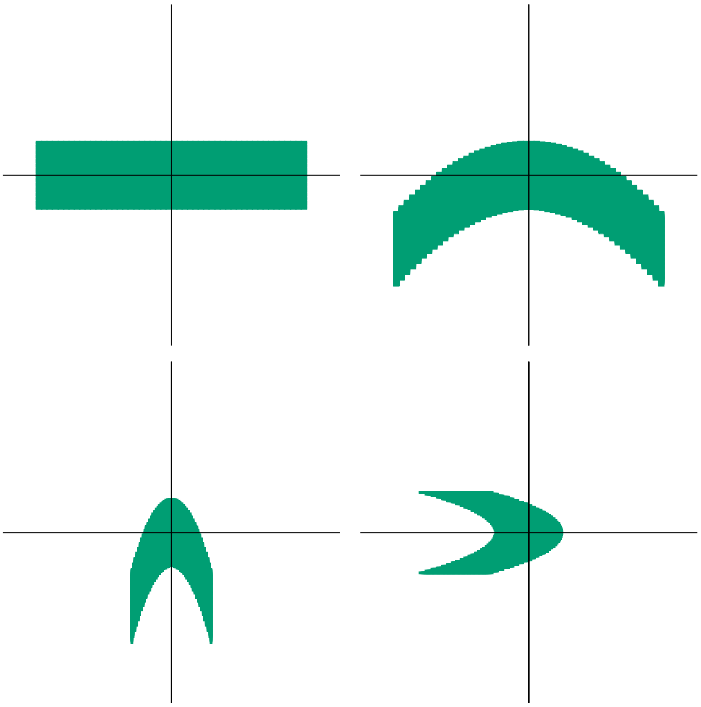

Abbildung 2.1: Einfache Evolution der Hénon Abbildung, aufgeteilt in das initiale Rechteck und die drei Zwischenschritte.

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.

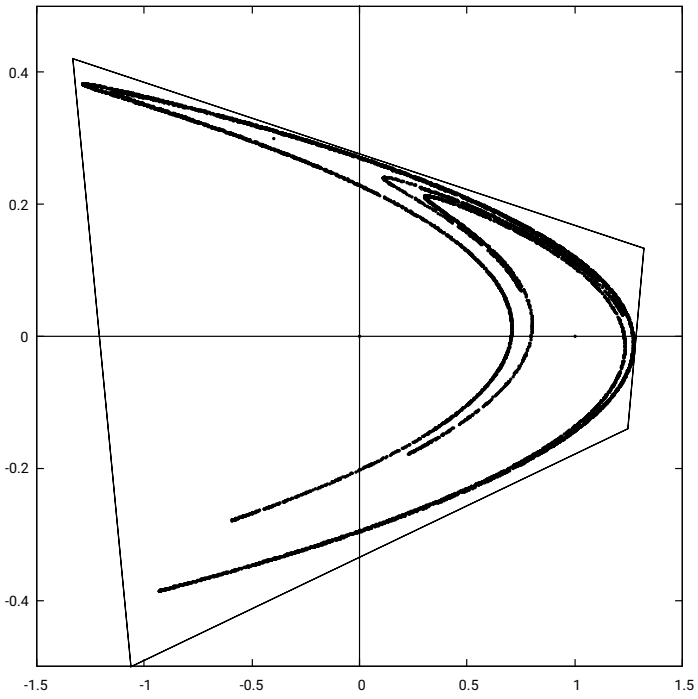

Abbildung 2.2: Seltsamer Attraktor der Hénon-Abbildung für $a = 1,4$ und $b = 0.3$ mit 10000 Punkten.

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 einem Monom c_0 des Grades 0, der das Kernintervall (*Kernel*) des Taylormodells darstellt. Eine Variable oder ein Fehlersymbol λ_i aus dem Vektor $\lambda = (\lambda_1, \ldots, \lambda_k)$ 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. 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 $T = \Sigma_n c_n \lambda^n$ 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.

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 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 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

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

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

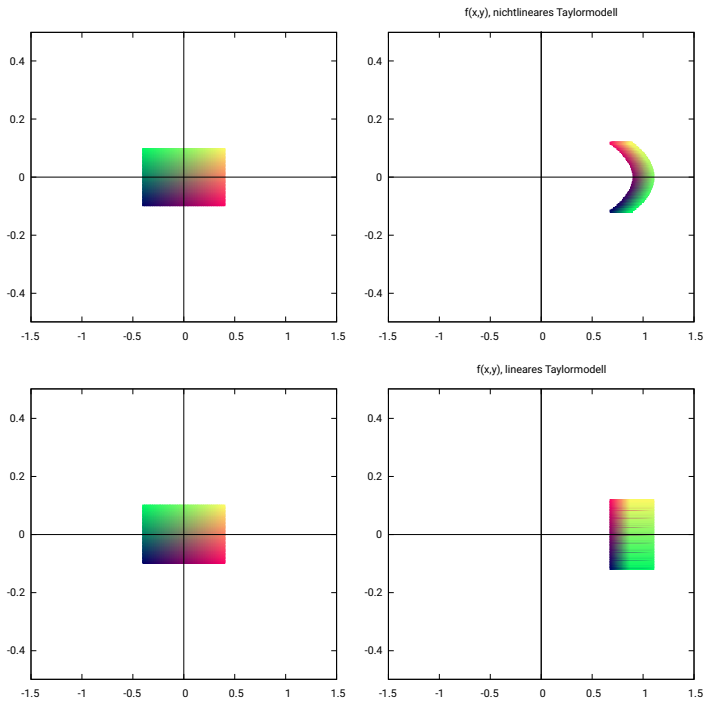

Abbildung 2.3: Darstellung nichtlinearer und linearer Taylormodelle, die ein Rechteck beschreiben und deren Abbildung durch die Funktion $f(x, y) = (y + 1 - 1.4x^2, 0.3x)$. Die Farbkodierung indiziert den Ursprung der abgebilteden Flächen.

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 Arithmetik auf Taylormodellen

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 Fehlersymbole durch das entsprechende Intervall aus S ersetzt wird:

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

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:

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

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

2.3 Mehrwertige Funktionen

3. Analyse

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.

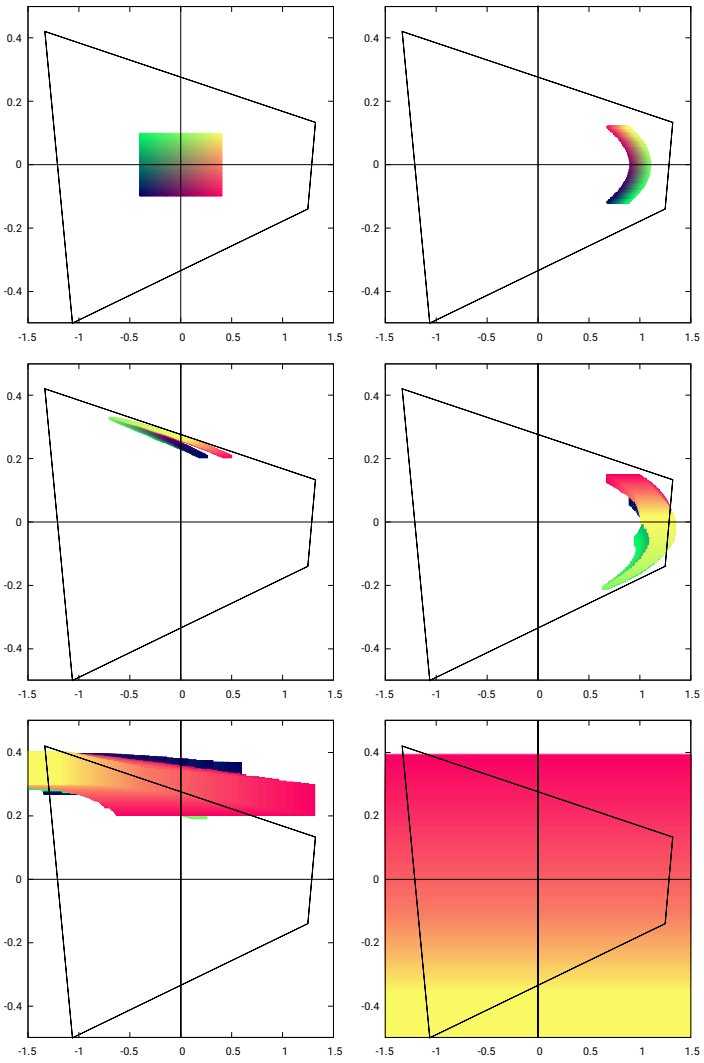

Abbildung 3.1

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

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

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. 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 gro0e 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 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 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 er 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 `REAL`s. 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.

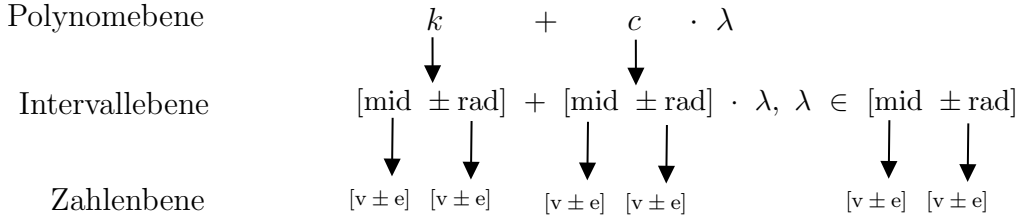

Abbildung 5.1: Ebenen der Polynomdarstellung mit REALs

5.2 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 [**?**].

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

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

5.3 Darstellung der Taylormodelle

5.4 Polynommultiplikation

In einer linearen Implementierung der Taylormodelle wird bei der Multiplikation immer eines der Fehlersymbole gesweept, sodass die Länge eines Polynoms nicht über die Dimension hinausgeht und somit der Schwerpunkt bei der Intervallarithmetik liegt. Lässt man allerdings höhere Ordnungen zu, werden auch die Polynome länger und ein weiteres Problem tritt auf. Um die oben (5.4) 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 $O(nm + (nm)^2)$

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 $\#p_1 >> \#p_2 = \#p_3 = 1$, 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.

Sei $x_0 = 0.5 + 1\lambda_1 + 1\lambda_2 + 1\lambda_3$ und einem Startfehler von 2^{-21}, also $\lambda_i \in [0 \pm 2^{-20}]$ für $i \in \{1, 2, 3\}$. Berechnet man nun $x_{i+1} = 3.75 \cdot x_i \cdot (1 - x_i)$ zeigen sich deutliche Unterschiede bei den benötigten Monomvergleichen.

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

- $\lambda_1 \succ \lambda_2 \succ ... \succ \lambda_k$

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

  $i > i'$, oder

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

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.

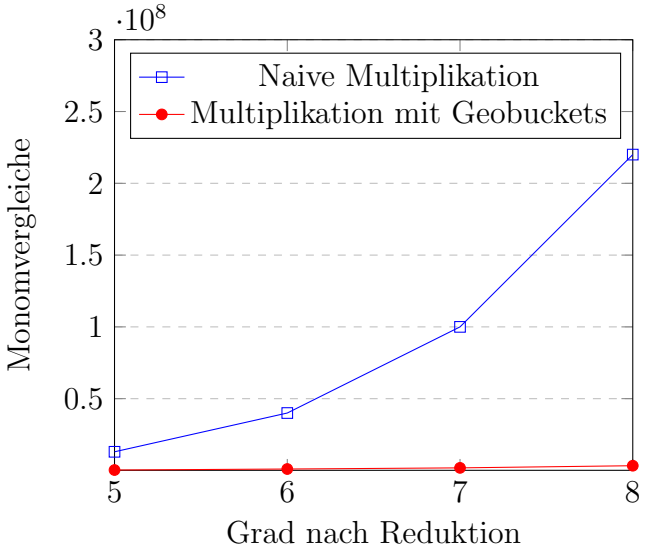

Abbildung 5.2: Naive Multiplikation gegen Multiplikation mit Geobuckets

Algorithm 1: Polynommultipliaktion mit Geobuckets

input: Polynome p_1, p_2

```
// Initialize
n ← #p_1, m ← #p_2 ;
p ← [ ] ;
Allocate buckets with space for polynomials: {2m, 4m, ..., 2^⌈log_2(n)⌉−1 m};
i ← 0;
// Main Loop
while i < n do
    p ← p_1[i] · p_2;
    if i < (n − 1) then
        p ← p + p_1[i + 1] · p_2;
    i ← i + 2;
    j ← 0;
    while buckets[j].not_empty() do
        p ← p + buckets[j] ;
        buckets[j] ← [ ];
        j ← j + 1;
    end
    if i < n then
        buckets[j] ← p;
        p ← [ ] ;
end
// Merge each bucket into the final polynomial
foreach bucket ∈ buckets do
    p ← p + bucket;
end
return p
```

6. Evaluation

7. Diskussion und Ausblick

(Keine Untergliederung mehr)

Literaturverzeichnis

[BrKM15] F. Braße, 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.

[Grtdt20] T. Granlund und G. the development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.2.0. Auflage, 2020. http://gmplib.org/.

[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.

[] 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.

[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.

[Weih00] K. Weihrauch. *Computable Analysis - An Introduction.* Texts in Theoretical Computer Science. An EATCS Series. Springer. 2000.

[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