

Implementierung hochgradiger Taylormodelle

Forschungspraktikum
Universität Trier
FB IV - Informatikwissenschaften
Professur Arithmetische Algorithmen

Betreuer: apl. Prof. Dr. Norbert Müller

Vorgelegt am xx.xx.xxxx von:

Benedikt Lüken-Winkels
Baltzstraße 6
54296 Trier
s4beluek@uni-trier.de
Matr.-Nr. 1138844

Inhaltsverzeichnis

1	Einleitung	1
1.1	Anwendung	2
1.2	Gliederung	2
2	Taylormodelle	3
2.1	Darstellung	3
2.2	Spezielle Operationen auf Taylormodellen	3
2.3	Sweeping-Strategien	4
2.3.1	Vergleich der Strategien	5
3	Implementierung der Polynome	7
3.1	Darstellung	7
3.2	Addition	8
3.3	Multiplikation	9
3.4	Inverse	10
4	iRRAM	13
4.1	Verwendung der iRRAM-REALs	13
4.2	Absolut vs relativ	13
4.3	Genauigkeiten stiff code	13
5	Evaluation	14
5.1	Linear vs Nicht-Linear	14
5.2	GMP - iRRAM	15
5.3	hotm - iRRAM Taylormodels	15
5.4	Logistic Map	15
5.5	Zusammenfassung	15
6	Diskussion und Ausblick	19
6.1	TODO	19
	Literaturverzeichnis	20

Abbildungsverzeichnis

2.1	Größenordnung der Breite des Kernintervalls bei der Berechnung von x_{250} mit Sweeping simple	6
2.2	Größenordnung der Breite des Kernintervalls bei der Berechnung von x_{250} mit Sweeping square_only	6
3.1	Naive Multiplikation gegen Multiplikation mit Geobuckets	10
5.1	$x_0 = [0 \pm 0] + [1 \pm 0] \cdot \lambda, \lambda \in [0.5 \pm \varepsilon]$	15
5.2	$x_0 = [0.5 \pm 0] + [1 \pm 0] \cdot \lambda, \lambda \in [0 \pm \varepsilon]$	16
5.3	$x_0 = [0.5 \pm 0] + [\varepsilon \pm 0] \cdot \lambda, \lambda \in [0 \pm 1]$	16
5.4	$x_0 = [0 \pm 0] + [0.5 \pm \varepsilon] \cdot \lambda, \lambda \in [1 \pm 0]$	17
5.5	$x_0 = [0.5 \pm 0] + [0 \pm \varepsilon] \cdot \lambda, \lambda \in [1 \pm 0]$	17
5.6	$x_0 = [0.5 \pm \varepsilon]$	18

Tabellenverzeichnis

3.1	Multiplikation: Geobuckets gegen Naiv	12
-----	---	----

1. Einleitung

Das Berechnen reeller Zahlen und Funktionen am Computer bringt einige Probleme mit sich, da diese nicht ohne Weiteres durch eine endliche Menge von Informationen dargestellt werden können (TTE). Mit der Verwendung von Intervallalgorithmen kann der Rechenfehler, beziehungsweise die Ungenauigkeit, welche sich durch die Approximation der Zahlen ergibt, berücksichtigt werden. Das Ziel dabei ist, den Fehler möglichst klein zu halten und dabei so wenig Geschwindigkeit, wie möglich bei der Berechnung zu verlieren. Mit Taylormodellen werden multivariate reelle Funktionen $f : \subseteq \mathbb{R}^k \rightarrow \mathbb{R}$ durch Polynome mit Intervallkoeffizienten zur lokalen Fehlerabschätzungen und Fehlersymbolen für Abhängigkeiten zwischen den einzelnen Monomen, dargestellt. Diese Herangehensweise soll das Problem der Überschätzung (des 'blow-ups') der Funktionswerte durch die Fehlerintervalle angehen, was man leicht an einem Beispiel verdeutlichen kann:

Seien $x_0 \in [1, 2], x_1 \in [2, 3]$. Es existiert also die Information, in welchen Wertebereichen x_0 und x_1 jeweils liegen, beziehungsweise wie ungenau die einzelnen Werte bekannt sind. Werden diese nun ohne Berücksichtigung der Abhängigkeiten verrechnet wie hier,

$$\begin{aligned}(x_0 + x_1) - x_0 &\rightarrow [1 + 2, 2 + 3] - [1, 2] \\ &= [1, 4]\end{aligned}$$

so entsteht beim Zusammenrechnen die Überschätzung $[1, 4]$, während das tatsächliche Ergebnis $[2, 3]$ einen deutlich kleineren Bereich abdeckt. Zwar ist auch die Überschätzung korrekt, jedoch könnte das Ergebnis genauer errechnet werden. Bei iterierenden Funktionen, wie

$$x_{n+1} = c \cdot x_n \cdot (1 - x_n)$$

welche in dieser Forschungsarbeit häufiger betrachtet wird. Für verschiedene Werte von c und x_0 weist die Funktion sowohl konvergierendes, als auch chaotisches Verhalten auf.

In dieser Forschungsarbeit wird in x_0 ein gewisser Fehler kodiert, um beispielsweise Messungenauigkeiten zu simulieren. Dieser Fehler ergibt in jeder Iteration eine wachsende Überschätzung, da x_0 nicht als genauer Wert vorliegt. Unkontrolliert wächst dieses Wrapping sehr schnell, sodass nach einigen Iterationen keine genauen Aussagen mehr über ein Ergebnis getroffen werden können. Durch verschiedene Techniken wird daher versucht, die Intervallbreite zu beschränken und klein zu halten, indem neue Variablen (Fehlersymbole) eingeführt werden, welche den Fehler und dessen Ursprung repräsentieren. Je länger die Abhängigkeitsinformationen über diese Fehler bei einer wachsenden Anzahl von Iterationen beibehalten, also die Intervalle nicht wieder eingesetzt werden, desto größer wird das berechnete Polynom

und dessen Grad. Es ergibt sich ein Trade-off zwischen der Menge an behaltener Information und Wartbarkeit des zu berechnenden Polynoms.

Eine Herangehensweise an dieses Problem ist die Verwendung von Taylormodellen ([MaBe01], [BrKM15]), welche im Fokus dieser Forschungsarbeit stehen. Die Implementierung im Rahmen dieses Forschungsprojektes geschah in der Programmiersprache *C++*, wobei für die Koeffizienten (und die Intervalle der Fehlersymbole) zum Einen die Rationalen Zahlen `mpq_class` der GMP-Bibliothek ([Grth20]) und zum Anderen die reellen Zahlen der `IRRAM`-Bibliothek ([Müll00]) verwendet wurden. In beiden Fällen wurde das Programm `Tangentspace`¹, einem Programm zum Erstellen linearer Schranken für nichtlinearer Funktionen an einem gegebenen Punkt, derart erweitert, als dass die bei den Berechnungen entstehenden Polynome Grade > 1 erreichen können. Um ein Produkt zweier Taylormodelle (zweier Polynome) linear zu halten, wird beim paarweisen Multiplizieren der Monome jeweils ein Fehlersymbol durch sein Intervall ersetzt, wodurch sich das Intervall des Koeffizienten und damit die Ungenauigkeit der Berechnung vergrößert. Wird dies nicht getan, bleiben die Fehlerintervalle zunächst klein, jedoch nicht das Polynom, welches in Grad und Anzahl der Monome wächst. Mit dem im Rahmen dieses Forschungsprojektes entstandenen Programm `hotm` (High Order Taylor Models) soll das bestmögliche Maß zwischen der Reduzierung des Grades, beziehungsweise der Vereinfachung der Polynome und dem Erhalt der Abhängigkeitsinformationen gefunden werden, um dann an verschiedenen Stellen verwendet zu werden.

1.1 Anwendung

Taylormodelle höherer Ordnung könnten in `Tangentspace` dazu verwendet werden, um diese in CDCL-artigen (Conflict Driven Clause Learning) SMT Programmen einzupflegen, wo derzeit lineare Taylormodelle verwendet werden. Außerdem können sie als Zahlentyp zur Berechnung reeller Funktionen, wie zum Beispiel Iterationen der Hénon-Abbildung dienen.

Betrachtet man die Fehlerintervalle, beziehungsweise deren Breite als Rauschen, können Taylormodelle dazu genutzt werden, physikalische Vorgänge, die eine gewisse Unsicherheit der Werte haben, darzustellen. Hierzu kann der Fehler durch die Definition des Startwertes als Taylormodell ausgedrückt werden. Um eine sich ändernde Umgebung darzustellen könnte im Falle der Anwendung bei der Logistic Map durch die Konstante in jeder Iteration ein neues, leicht verändertes Fehlersymbol eingeführt werden; oder zu jeder Iteration wird ein bestimmtes Fehlerintervall addiert.

1.2 Gliederung

In diesem Forschungsbericht wird der Aufbau und die Funktionsweise der einzelnen Bestandteile der Implementierung der `hotm` neben der Hintergrund für dieselben beschrieben.

¹<http://informatik.uni-trier.de/~mueller/Research/Tangentspace/>

2. Taylormodelle

Ein Taylormodell, in [MaBe01] eingeführt und in [BrKM15] erweitert, ist ein Polynom $T = \sum_n c_n \lambda^n$, mit k Variablen. λ_i ist ein Fehlersymbol aus $\lambda = (\lambda_1, \dots, \lambda_k)$, dem Vektor der Fehlersymbole, wodurch funktionale Abhängigkeiten zwischen Monomen und anderen Taylormodellen hergestellt werden können. Die Koeffizienten des Polynoms sind Intervalle $c_i = [\tilde{c}_i \pm e_i] \subseteq \mathbb{R}$ mit denen der aktuelle Rechenfehler ('approximation error') abgeschätzt wird. Ist ein Wert genau bekannt, also ohne Ungenauigkeit oder Fehler, kann dieser als Punktintervall mit $c_i = [\tilde{c}_i \pm 0]$ definiert werden, also mit einem Zentrum ohne Radius.

2.1 Darstellung

Ein Taylormodell besteht in der Implementierung aus dem Kernintervall `kernel`, einer Liste von sortierten von Monomen und einer Liste von Supportintervallen, den Intervallen, der Fehlersymbole. Die Liste der Supportintervalle ist für jedes Taylormodell dieselbe.

2.2 Spezielle Operationen auf Taylormodellen

Quadratisches Sweeping

Um die Länge der Polynome und deren Grad in den Taylormodellen zu kontrollieren können einzelne Fehlersymbole in einem Monom durch ihr Intervall ersetzt werden. So erhöht sich zwar die Breite des Koeffizienten, allerdings wird auch das Verrechnen mit einem anderen, nun gleichen Monom ermöglicht. Dieses *Sweeping* bietet eine natürliche Möglichkeit, den durch die Fehlersymbole eingeführten Fehler klein zu halten, indem versucht wird, bei *quadratischen* Terme nicht ein Fehlersymbol nach dem anderen zu Sweepen, sondern deren Quadrate einzusetzen:

Gegeben sei ein Monom eines Polynoms $\dots + c \cdot \lambda^3 + \dots$ mit $\lambda \in [-1.5, 1.5]$. Nun sieht man, dass zwar λ sowohl positiv, als auch negativ sein kann, jedoch das Quadrat nicht. Daher kann man statt

$$\begin{aligned} & \dots + c \cdot \lambda^3 + \dots \\ \rightarrow & \dots + c \cdot [-1.5, 1.5] \cdot \lambda^2 + \dots \\ \rightarrow & \dots + c \cdot [-1.5, 1.5] \cdot [-1.5, 1.5] \cdot \lambda + \dots \\ \rightarrow & \dots + c \cdot [-2.25, 2.25] \cdot \lambda + \dots \end{aligned}$$

das Quadrat einsetzen

$$\begin{aligned} & \dots + c \cdot \lambda^3 + \dots \\ \rightarrow & \dots + c \cdot [0, 2.25] \cdot \lambda + \dots \end{aligned}$$

und somit das Wachstum des Rechenfehlers verlangsamen.

Polishing

Rechenoperationen auf Intervallen sorgen dafür, dass sich diese Vergrößern. Mit gut gewählten Sweeping-Strategien kann dieser Effekt zwar verlangsamt, aber nicht aufgehalten werden. Wie in Kapitel 5 zu sehen ist, kann ab einer bestimmten Intervallbreite keine Aussage mehr über das Ergebnis getroffen werden. Mit *Polishing* [BrKM15] kann ein zu großes Intervall eines Koeffizienten in zwei Monome mit jeweils einem Punktintervall und einem neuen Fehlersymbol aufgeteilt werden. Das verlängert zwar das Polynom, verhindert jedoch ein zu großes Wachstum der Intervallbreite. Eine Implementierung in `hotm` ist derzeit nicht vorhanden, aber angedacht.

2.3 Sweeping-Strategien

In einer linearen Implementierung der Taylormodelle und deren Polynome, muss Sweeping in jeder Multiplikation geschehen, da sich sonst der Grad des Polynome erhöhte. Folgt man der Idee des *quadratischen Sweepens*, so ergibt sich bei der Multiplikation zweier Polynome und dementsprechend der paarweisen Multiplikation der Monome:

$$(c_1 \cdot \lambda_i) \cdot (c_2 \cdot \lambda_j) = \begin{cases} c_1 c_2 s_j \lambda_i & \text{wenn } i \neq j \text{ und } s_i > s_j \\ c_1 c_2 s_i \lambda_j & \text{wenn } i \neq j \text{ und } s_i \leq s_j \\ c_1 c_2 s_i^2 & \text{wenn } i = j \end{cases} \quad \lambda_i \in s_i, \lambda_j \in s_j$$

Im dritten Fall, also bei $i = j$, wird das Ergebnis der Multiplikation zum Kernintervall addiert. Das Problem bei dieser Herangehensweise ist, dass ein Fehler, der ins Kernintervall gesweept wurde, nicht mehr mit einem Fehlersymbol (λ) assoziiert werden kann und damit keinerlei Abhängigkeitsinformationen mehr enthält. Zwar tritt dieser Fall nicht so häufig auf, wie die beiden anderen, allerdings bedeutet das auch, dass nicht so häufig quadratisch gesweept wird. Eine nicht-lineare Implementierung erlaubt ein Wachsen der Grade der Polynome und andere Sweeping-Strategien. In `hotm` geschieht das Sweeping eines Taylormodells t per Funktionsaufruf unter Angabe der zu verwendenden Sweeping-Strategie und dem Grad, auf den t reduziert werden soll. Wird beispielsweise als Grad „1“ mitgegeben, ist die Ausgabe höchstens linear, es werden also keine Monome auf den Grad 0 reduziert. Um ein Taylormodell zu evaluieren und auf ein einziges Intervall zu reduzieren, muss der Zielgrad „0“ sein.

Da sich der Grad eines Polynoms nach dem Grad des größten Monoms richtet, muss jedes Monom auf den Zielgrad reduziert werden. Dafür wird über die Fehlersymbole der einzelnen Monome iteriert und eine der Strategien angewandt:

`square_only`

Das Sweeping wird auf quadratisches Sweeping beschränkt. Für jedes Fehlersymbol mit einem Exponenten > 1 wird versucht, einen möglichst großen, geraden Wert von demselben abzuziehen.

Beispiel `sweep_to(1)`:

$$\begin{aligned} & \dots + c \cdot \lambda_1^2 \lambda_2^5 \lambda_3^1 + \dots && (\lambda_i \in s_i) \\ & \dots + c s_1^2 \cdot \lambda_2^5 \lambda_3^1 + \dots && \lambda_1 \text{ wird quadriert und somit quasi weggekürzt} \\ & \dots + c s_1^2 s_2^2 \cdot \lambda_2^1 \lambda_3^1 + \dots && \lambda_2 \text{ kann nur 2-mal quad. gesweept werden} \end{aligned}$$

Unter Verwendung von `square_only` ist es möglich, dass das Ergebnis, wie im Beispiel, nicht den angegebenen Zielgrad erreicht.

Algorithm 1: Sweeping eines (nicht-linearen) Taylormodells

input: Taylormodell t , Grad deg , Strategie $strat$

```

1 Polynomial  $p \leftarrow []$ ;
2 foreach Monomial  $m \in t.polynomial$  do
3    $rest \leftarrow degree(m) - deg$ ;
4   if  $rest < 1$  then
5      $p.insert(m)$ ;
6   else
7     foreach ErrorSymbol  $es \in m$  do
8       if  $rest < threshold$  then
9         break;
10      Sweep according to Strategy;
11       $rest \leftarrow rest - \text{amount swept}$ ;
12   end
13    $p.insert(m)$ ;
14 end
15 return  $t$ 

```

simple

simple Sweeping verwendet keinerlei 'Tricks' beim Sweeping.

square_first

Hier wird zunächst **square_only** auf das Monom angewandt, um danach die restlichen Grade ohne quadratisches Sweeping zu reduzieren. Allerdings hat diese Methode keinerlei Vorteile gegenüber **simple**, da der Effekt des Quadrierens von einem nicht-quadratischen Sweep wieder aufgehoben wird.

2.3.1 Vergleich der Strategien

Um den Effekt der verschiedenen Strategien zu betrachten, wurde die Intervallbreite des evaluierten x_{250} für $x_0 = 0.5 + 1\lambda$, einem Startfehler von 2^{-101} , also $\lambda \in [0 \pm 2^{-100}]$ und $x_{i+1} = 3.25 \cdot x_i \cdot (1 - x_i)$. Diese Variante der Logistischen Gleichung (engl. *Logistic Map*) konvergiert gegen zwei Werte. Nach jeder Iteration wurde ein **sweep_to(1)** aufgerufen. Abbildung 2.1 zeigt die Größenordnung der Breite des Kernintervalls nach jeder Iteration, abhängig von der verwendeten Strategie. Für **simple** lässt sich ein stetiges Wachstum der Breite beobachten, bis die bei der zirka 230ten Iteration > 1 wird. Ab diesem Zeitpunkt ist der Rechenfehler zu groß und die Ausgaben werden falsch, konvergieren also nicht mehr gegen die Fixpunkte. Mit **square_first** wiederum steigt die Genauigkeit zunächst bis zur zirka 105ten Iteration, gerundet wird und die Breite zu wachsen beginnt, allerdings nicht über ein gewisses Niveau hinaus, wodurch die Werte korrekt bleiben.

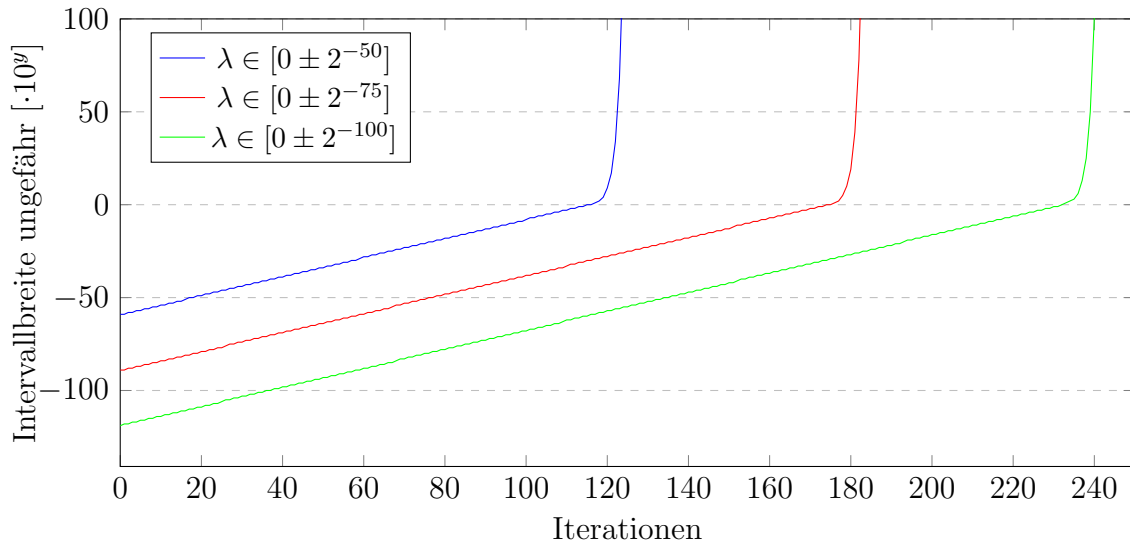


Abbildung 2.1: GröÖenordnung der Breite des Kernintervalls bei der Berechnung von x_{250} mit Sweeping simple

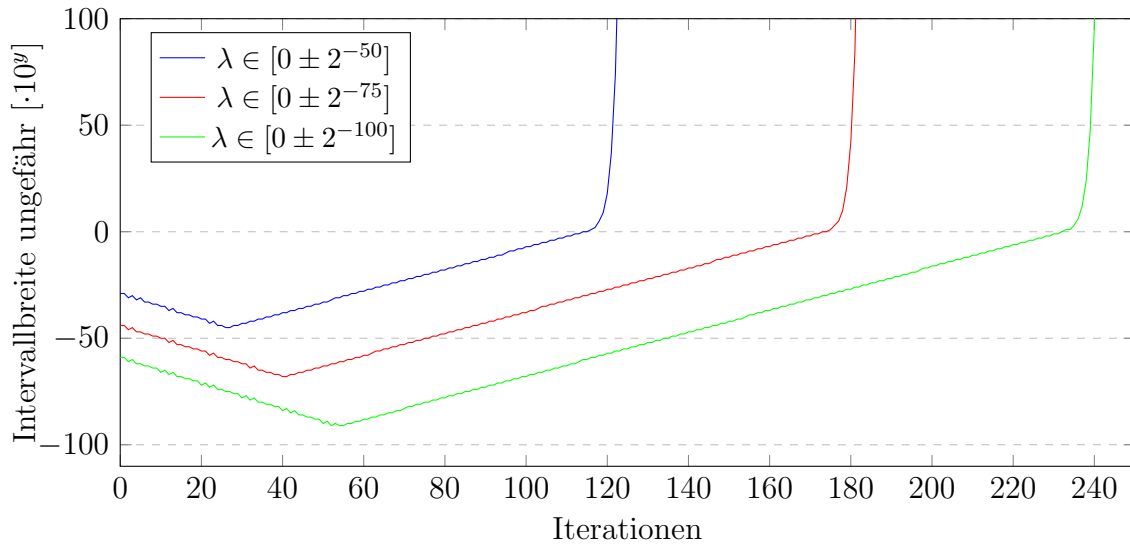


Abbildung 2.2: GröÖenordnung der Breite des Kernintervalls bei der Berechnung von x_{250} mit Sweeping square_only

3. Implementierung der Polynome

Die Implementierung der einzelnen Bestandteile ist objektorientiert. Dieses Kapitel bietet einen Überblick über die Struktur des Programmes und den wichtigsten zu Verfügung stehenden Funktionen.

3.1 Darstellung

Intervalle

Intervalle enthalten Informationen über deren Zentrum **center** und Radius **radius**, beziehungsweise deren Breite, welche, je nach gewähltem Zahlentypen, reelle oder rationale Zahlen sind. Zudem können Intervalle ein- oder beidseitig unendlich sein, weshalb wiederum eine Abfrage nach **IntType**¹, also dem Intervalltypen möglich ist. Existiert eine Schranke, also ist das Intervall endlich oder einseitig unendlich, kann diese abgefragt werden.

Beim Verrechnen von Intervallen sind folgende Regeln zu beachten:

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

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

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

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

Hier wird der Effekt deutlich, dass Intervalle nach jeder Operation die Tendenz haben, zu wachsen. Wird beispielsweise ein Intervall von sich selbst subtrahiert, wird es größer, statt sich wegzukürzen.

Fehlersymbole

Ein Fehlersymbol λ besteht aus seinem Exponenten, einem **Pointer** auf das Intervall in dem sein Wert liegt, welches sich während der Berechnung verändern kann und dem Index des Fehlersymbols.

Monome

Monome werden durch einen Intervallkoeffizienten und eine Liste von nach ihrem Index aufsteigend geordneten Fehlersymbolen dargestellt. Die Implementierung der Liste der Fehlersymbole als **std::vector**, ermöglicht die Verwendung der *C++ Standard Template Library* für Funktionen, wie zum Beispiel die Berechnung des Grades eines Monoms.

¹FINITE, LOWER_BOUND, UPPER_BOUND, ENTIRE

Polynome

Ein Polynom setzt sich aus einem Kernintervall (**kernel**), also dem Monom ohne Fehlersymbole und einer Liste ungleicher Monome zusammen. Ungleich bedeutet in diesem Fall, dass sich die Zusammensetzung oder der Exponent einzelner Fehlersymbole zwischen den Monomen unterscheidet.

3.2 Addition

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 die Polynom-Summe 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 diese Ordnung und hat den Vorteil, dass nicht immer alle Fehlersymbole aus beiden Listen betrachtet werden müssen. Sobald sich die beiden Monome, genauer deren Fehlersymbole, an der ersten Stelle unterscheiden, terminiert die Schleife. Die Vorbedingung hierfür ist, dass die Fehlersymbole in den Listen aufsteigend sortiert sind.

In vorhergegangenen Implementierungen der Fehlersymbole waren diese noch keine Objekte. Stattdessen hatte jedes Monom einen **vector** mit einem Eintrag pro Fehlersymbol. Der Index des Fehlersymbol entsprach dem Index innerhalb des Vectors, der Wert im **vector** dem Exponenten, was die Datenstruktur vereinfachte, aber erforderte, dass alle Monome dieselbe Anzahl an Fehlersymbolen hatten. Da sich diese allerdings ändern kann, wenn durch ein **Polish** ein neues λ hinzugefügt oder eines der λ s durch **Sweeping** kein Vorkommen mehr hat und entfernt werden kann, wurde eine andere Herangehensweise verwendet.

Algorithm 2: max_errorsymbols

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

```

Werden nun sortierte Polynome p_1, p_2 addiert, geschieht lediglich ein Zusammenführen (merge) zweier geordneter Listen, was im schlimmsten Fall $\#p_1 + \#p_2$ Vergleiche der Monome benötigt².

Algorithm 3: Addition zweier Polynome

input: Polynome p_1, p_2

```

1  $p \leftarrow (), i \leftarrow 0, j \leftarrow 0$  ;
2 while  $i < \#p_1$  and  $j < \#p_2$  do
3    $max \leftarrow max\_errorsymbols(p_1[i], p_1[j])$  ;
4   if  $max = left$  then
5      $p.append(p_1[i])$ ;
6      $i \leftarrow i + 1$ ;
7   else if  $max = right$  then
8      $p.append(p_2[j])$ ;
9      $j \leftarrow j + 1$ ;
10  else
11     $// max = equal$ 
12     $p.append(p_1[i] + p_2[j])$ ;
13     $p_1.remove(i)$ ;
14     $i \leftarrow i + 1; j \leftarrow j + 1$ ;
15 end
16 Append the rest to  $p$ ;
17 return  $p$ 
```

Algorithmus 3 erzeugt aus zwei nach der oben definierten Ordnung \prec 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 \prec) kleiner sind, als die des anderen Summanden und daher dem Ergebnispolynom angehängen werden können, ohne sie weiter zu betrachten.

3.3 Multiplikation

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 (3.2) 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 $O(nm + (nm)^2)$

² $\#p$ liefert die Anzahl an Monomen des Polynoms p

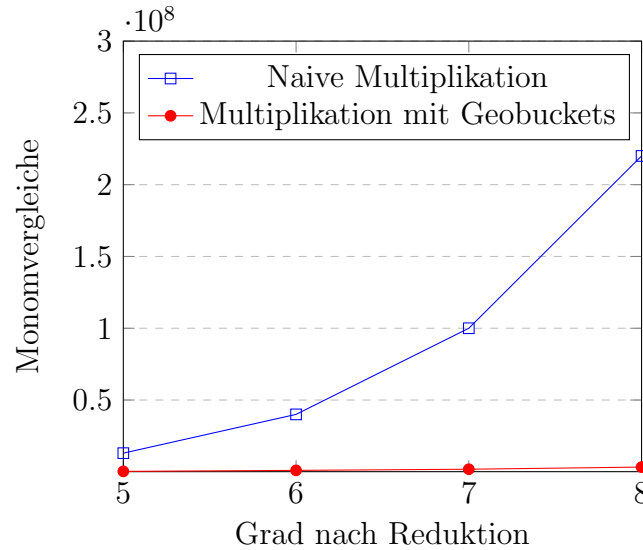


Abbildung 3.1: Naive Multiplikation gegen Multiplikation mit Geobuckets

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. (TODO Nachgucken FFT Intervallarithmetik)

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 kennt und die Bucketkapazität dementsprechend anpassen kann. Algorithmus ?? 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, beziehungsweise dem Aufruf der Funktion `max_monomial` (Algorithmus 2).

3.4 Inverse

Algorithm 4: 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  $p$ 

```

Naive Multiplikation		
Grad nach Reduktion	CPU-Zeit	#Vergleiche
5	2.375 s	$\approx 1.3 \cdot 10^7$
6	5.281 s	$\approx 4 \cdot 10^7$
7	11.281 s	$\approx 1 \cdot 10^8$
8	22.406 s s	$\approx 2.2 \cdot 10^8$
Multiplikation mit Geobuckets		
Grad nach Reduktion	CPU-Zeit	#Vergleiche
5	1.875 s	$\approx 3 \cdot 10^5$
6	3.859 s	$\approx 1 \cdot 10^6$
7	6.406 s	$\approx 1.8 \cdot 10^6$
8	11.14 s	$\approx 3.3 \cdot 10^6$

Tabelle 3.1: Berechnung von x_{10} mit Reduzierung des Grades nach jeder Iteration. Gemessen mit dem iRRAM-internen Debugmodus und Code-Coverage.

4. 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. 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 die Zugriffe für den Nutzer auf Variablen und Zwischenergebnisse genau kontrolliert und unter Umständen beschränkt werden, da sonst unerwartetes Verhalten und Exceptions entstehen können.

4.1 Verwendung der iRRAM-REALs

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 des Ergebnisses angenommen.

Die V

5. Evaluation

Für die Initialisierung der Taylormodelle, beziehungsweise der Einführung des Fehlers gibt es mehrere Möglichkeiten. Für die Berechnung von $x_{i+1} = c \cdot x_i \cdot (1 - x_i)$ eignet sich die Definition des 'Startwertes' x_0 als Taylormodell.

Man wähle $x_0 = 0.5$ und einen Fehler ε (zum Beispiel $\varepsilon = 2^{-100}$), so kann das Taylormodell auf verschiedene Art und Weise definiert werden, um diesen Wertebereich abzudecken:

$$x_0 = [0 \pm 0] + [1 \pm 0] \cdot \lambda \quad \lambda \in [0.5 \pm \varepsilon] \quad (5.1)$$

$$x_0 = [0.5 \pm 0] + [1 \pm 0] \cdot \lambda \quad \lambda \in [0 \pm \varepsilon] \quad (5.2)$$

$$x_0 = [0.5 \pm 0] + [\varepsilon \pm 0] \cdot \lambda \quad \lambda \in [0 \pm 1] \quad (5.3)$$

$$x_0 = [0 \pm 0] + [0.5 \pm \varepsilon] \cdot \lambda \quad \lambda \in [1 \pm 0] \quad (5.4)$$

$$x_0 = [0.5 \pm 0] + [0 \pm \varepsilon] \cdot \lambda \quad \lambda \in [1 \pm 0] \quad (5.5)$$

$$x_0 = [0.5 \pm \varepsilon] \quad (5.6)$$

In 5.4 und 5.5 wird der Fehler nicht im Fehlersymbol λ 'versteckt', sodass er dauerhaft in der Rechnung vorhanden und nicht ohne `polish` zu verkleinern ist.

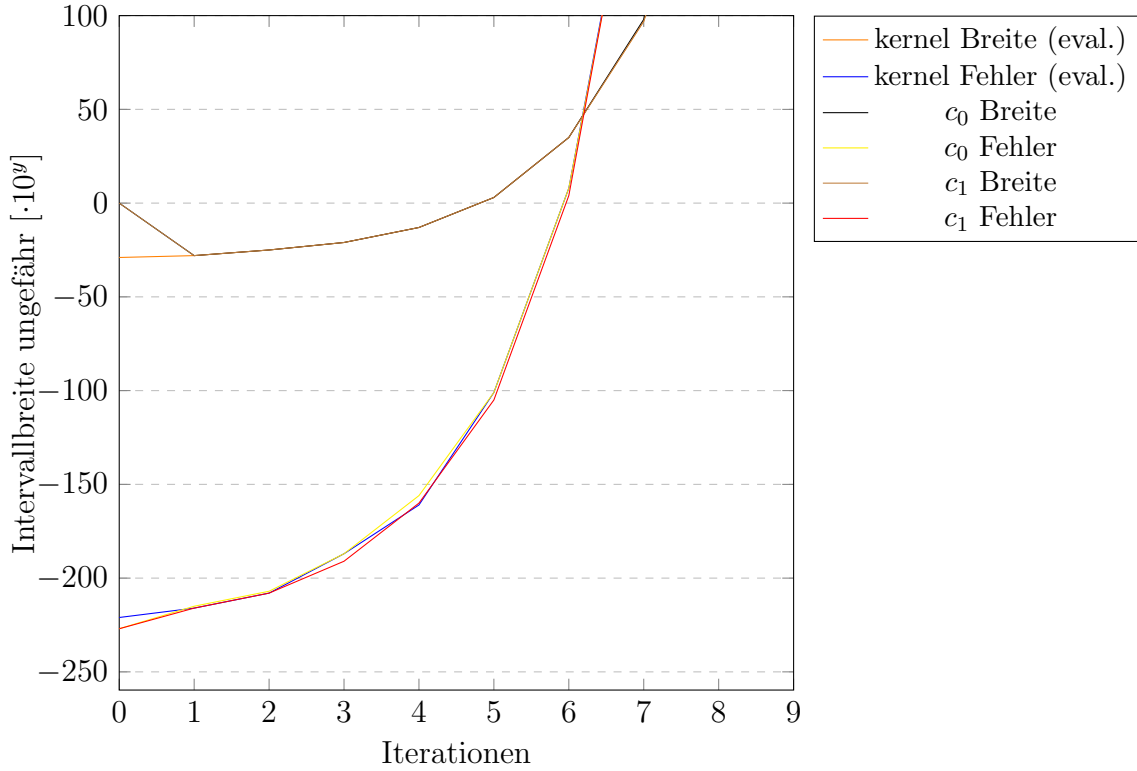
5.1 unterscheidet sich insofern von den restlichen Definitionen, in denen das λ ein nicht-Punktintervall ist, als dass das Intervall nicht um die 0 herumliegt.

5.2 und 5.3 haben bei der Verwendung der Sweepingstrategie `SQUARE_ONLY` den Vorteil, dass der zugelassene Fehler, der bei der Multiplikation des Fehlerintervalls hinzukommt lediglich halb so groß, wie das Intervall ist.

5.1 Linear vs Nicht-Linear

Ideen

- Formel von oben mit Iterationen und Startfehler e
- Untersuchung, nach wievielen Iterationen Linear nicht mehr ausreicht
- Untersuchung, wie sich die Gradzahl auf die Iterationen auswirkt, bei gleichbleibendem Startfehler
- Vergleich mit iRRAM Taylormodels und dergleichen

Abbildung 5.1: $x_0 = [0 \pm 0] + [1 \pm 0] \cdot \lambda$, $\lambda \in [0.5 \pm \varepsilon]$

5.2 GMP - iRRAM

5.3 hotm - iRRAM Taylormodels

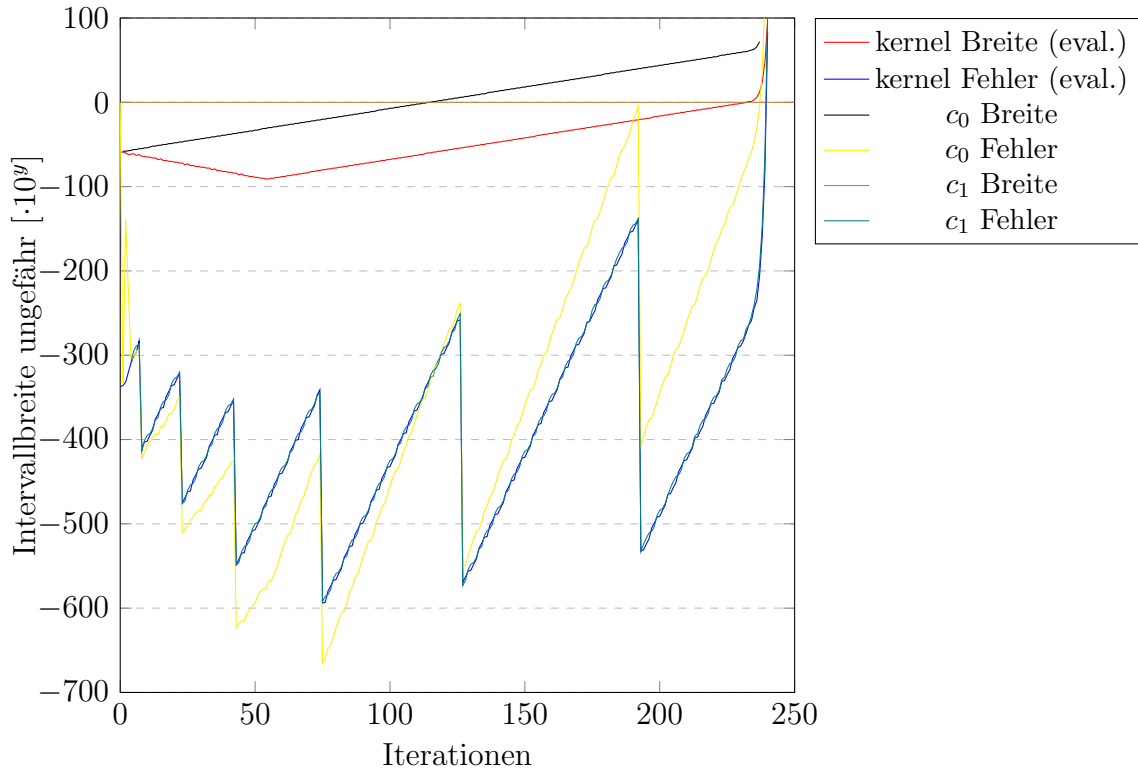
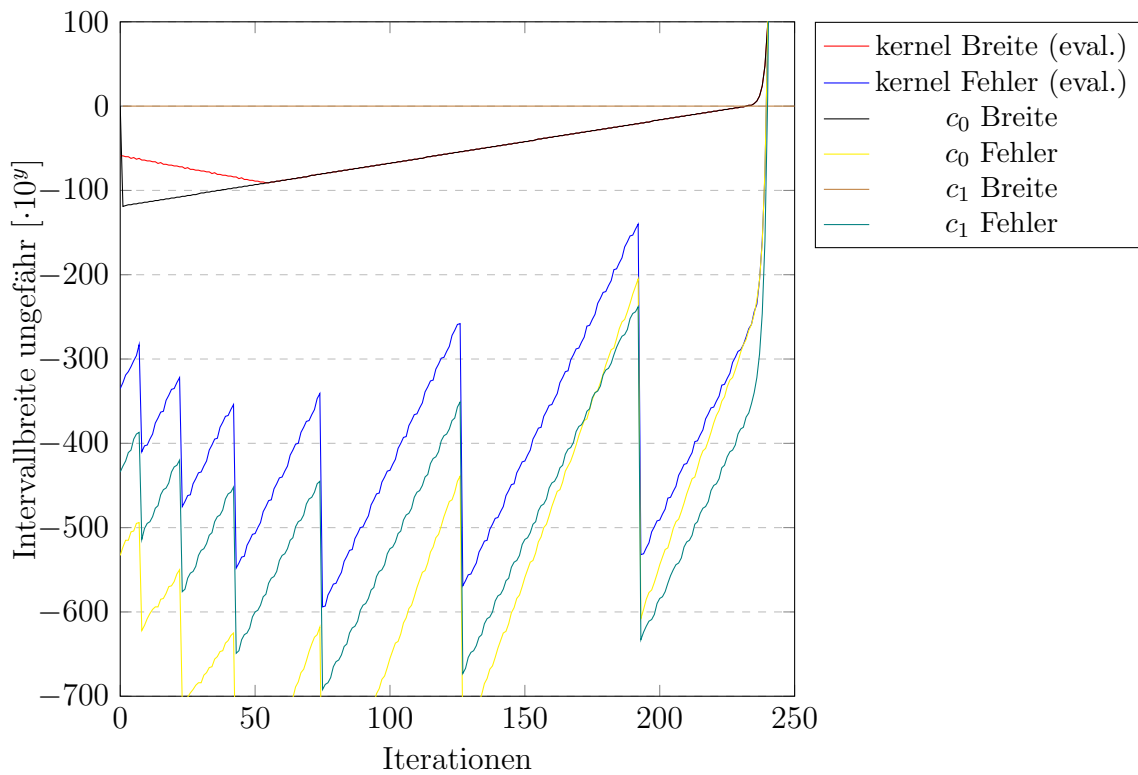
5.4 Logistic Map

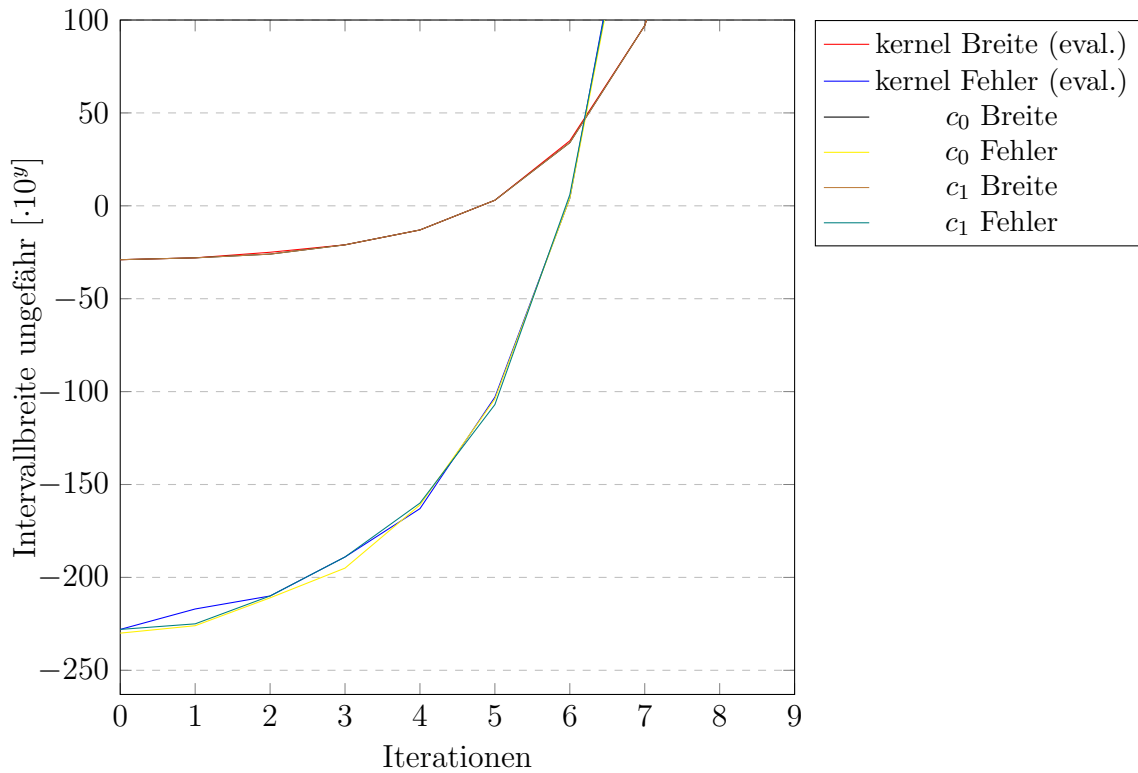
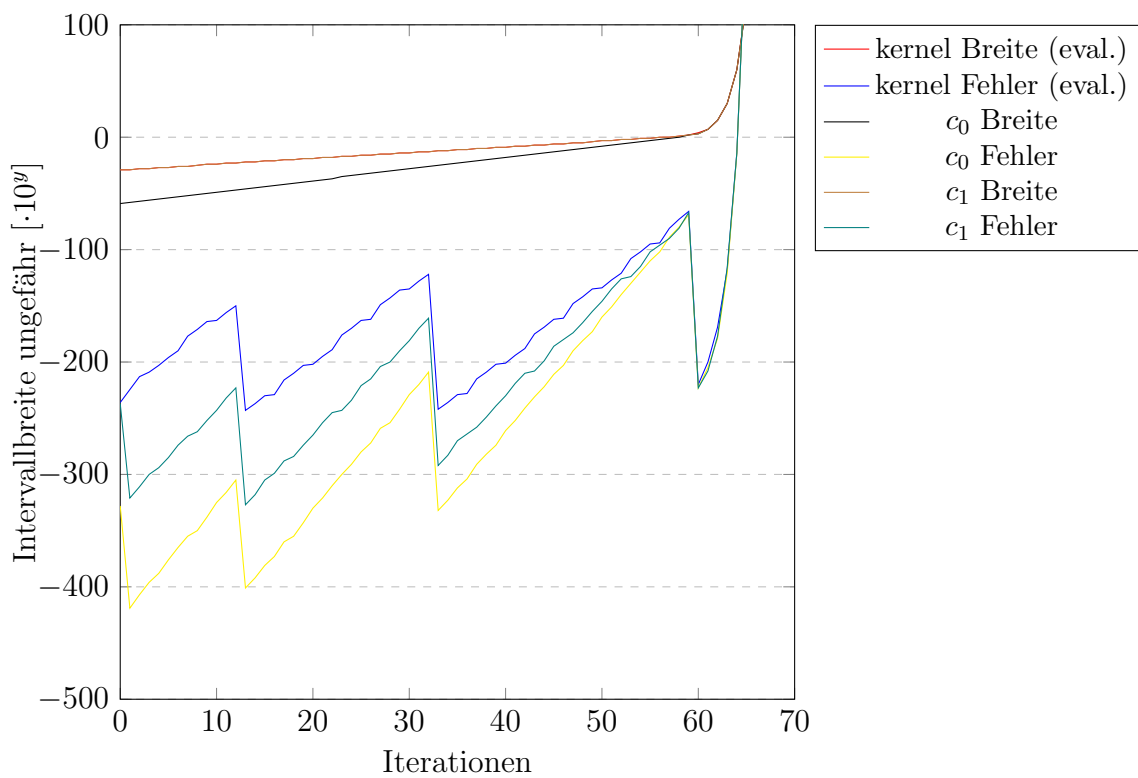
Um zu messen, welche der in 5 aufgezählten Definitionen des Startwertes für x_0 als Taylormodell am Besten zur Berechnung von Iterationen der Logistic Map eignet, wurden für die Verwendung der iRRAM-REALs die Anzahl der iRRAM-internen Iterationen und die benötigte Genauigkeit miteinander verglichen. Für die GMP-mpq_class wurde getestet, wieviele Iterationen möglich waren, bevor die Werte zu ungenau wurden.

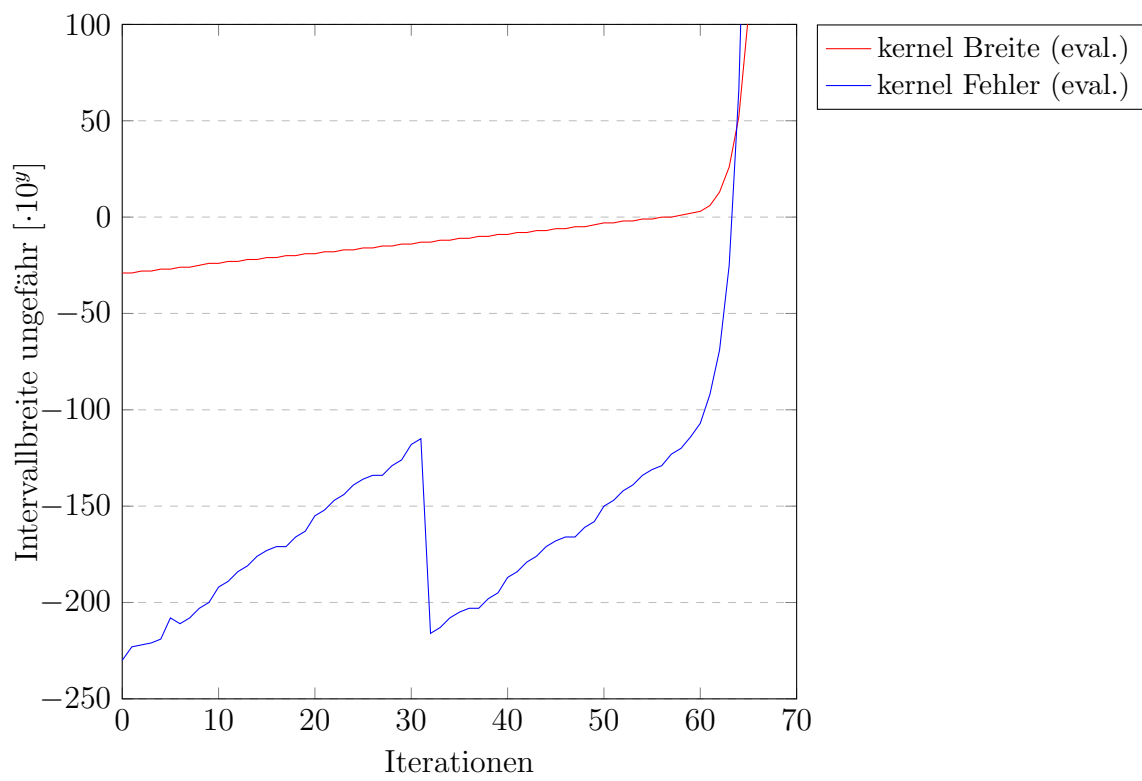
Es bieten sich viele Stellen zum Verändern der Testbedingungen:

- Verändern (Vergrößern oder Verkleinern) des Startfehlers ε
- Variation des Taylormodells für x_0 , also 5.1 oder 5.2
- Anpassung des Grades, auf dem das Taylormodell durch Sweeping während der Berechnung der Iterationen gehalten wird.

5.5 Zusammenfassung

Abbildung 5.2: $x_0 = [0.5 \pm 0] + [1 \pm 0] \cdot \lambda$, $\lambda \in [0 \pm \varepsilon]$ Abbildung 5.3: $x_0 = [0.5 \pm 0] + [\varepsilon \pm 0] \cdot \lambda$, $\lambda \in [0 \pm 1]$

Abbildung 5.4: $x_0 = [0 \pm 0] + [0.5 \pm \varepsilon] \cdot \lambda$, $\lambda \in [1 \pm 0]$ Abbildung 5.5: $x_0 = [0.5 \pm 0] + [0 \pm \varepsilon] \cdot \lambda$, $\lambda \in [1 \pm 0]$

Abbildung 5.6: $x_0 = [0.5 \pm \varepsilon]$

6. Diskussion und Ausblick

6.1 TODO

- 'Runden' für rationale Zahlen auf dyadische Zahlen: Abschneiden einer Dezimalzahl, wo die Genauigkeit zu groß wird und dann Einführen eines Taylormodells, dass den 'vorderen Teil der Zahl' als Kernel und die nun fehlende Genauigkeit als im Lambda umschließt.
- Finden oder Entwickeln von Heuristiken, die entscheiden, welche Fehlersymbole am besten gesweept werden sollten, wenn der Grad eines Monoms reduziert werden soll. Bei einer Reihe von Tests, wo bestimmte Konfigurationen zum Sweepen verwendet wurden, gab es Anzeichen darauf, dass die Qualität des Ergebnisses der Evaluation/Linearisierung von der Reihenfolge abhängt mit der die einzelnen Fehlersymbole entfernt werden.
- Lineare Lösung versuchen, bis der Fehler zu groß wird.

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.
- [Grth20] T. Granlund und the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.2.0. Auflage, 2020. <http://gmplib.org/>.
- [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.
- [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.
- [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.