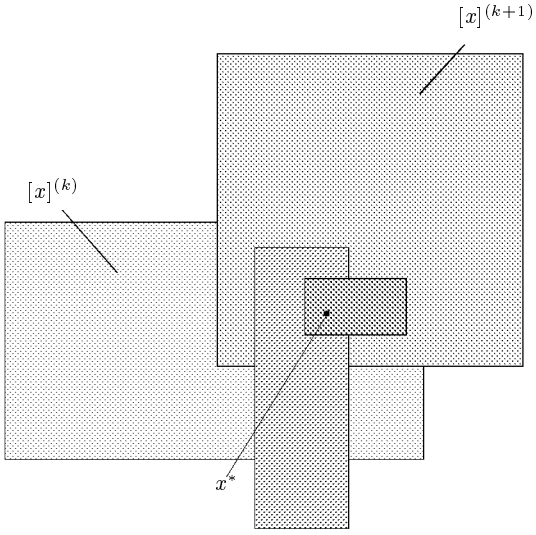
Institut für
Angewandte
Mathematik

Universität Karlsruhe (TH)
D-76128 Karlsruhe

Rückwärtsmethode zur automatischen Berechnung von worst-case Fehlerschranken

Michael Bräuer und Walter Krämer

Forschungsschwerpunkt
Computerarithmetik,
Intervallrechnung und
Numerische Algorithmen mit
Ergebnisverifikation



Bericht 3/1999

Impressum

Herausgeber:	Institut für Angewandte Mathematik Lehrstuhl Prof. Dr. Ulrich Kulisch Universität Karlsruhe (TH) D-76128 Karlsruhe
Redaktion:	Dr. Gerd Bohlender

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über

```
ftp://iamk4515.mathematik.uni-karlsruhe.de|
im Verzeichnis: /pub/documents/reports|
```

oder über die World Wide Web Seiten des Instituts

```
http://www.uni-karlsruhe.de/|iam|
```

Autoren-Kontaktadresse

Rückfragen zum Inhalt dieses Berichts bitte an

Michael Bräuer
Walter Krämer
Institut für Angewandte Mathematik
Universität Karlsruhe (TH)
D-76128 Karlsruhe

E-Mail: walter.kraemer@math.uni-karlsruhe.de

Rückwärtsmethode zur automatischen Berechnung von worst-case Fehlerschranken

Michael Bräuer und Walter Krämer

Zusammenfassung

Rückwärtsmethode zur automatischen Berechnung von worst-case Fehlerschranken:
Die insbesondere im Zusammenhang mit der Berechnung von Gradienten häufig verwendete schnelle Rückwärtsmethode wird eingesetzt, um worst-case Fehlerschranken für Gleitkommaalgorithmen automatisch und a priori zu bestimmen. Die numerischen Ergebnisse dieses Verfahrens sowie dessen Handhabung und die benötigten Laufzeiten werden mit den entsprechenden Größen eines Vorwärtsfehlerkalküls verglichen.

Abstract

Computaton of Worst Case Error Bounds in Reverse Mode:
A priori worst case error bounds for floating point algorithms are computed automatically using the so called reverse mode.

1 Einleitung

Bei der Verwendung numerischer Verfahren auf dem Rechner ist nicht nur das Laufzeitverhalten ein entscheidendes Einsatzkriterium. Ebenso wichtig ist das Verhalten dieser Methoden bezüglich der Verstärkung auftretender Rundungsfehler und möglicher Störungen in den Eingangsdaten. Deshalb ist man an der Angabe einer gleichmäßigen Fehlerschranke interessiert, welche eine obere Schranke für die maximale Abweichung eines auf der Maschine errechneten Wertes vom wahren (in $\mathbb{R}$ berechneten und allgemein unbekannten) Resultat für beliebige Argumente eines vorgegebenen Bereichs darstellt. Hat man einen Algorithmus in Gestalt einer faktorisierbaren Funktion vorliegen, so läßt sich eine a-priori Fehlerschranke für diese Funktion berechnen, falls die Größe der Eingangsdaten und deren Eingangsfehler bekannt sind.

Im folgenden Abschnitt 2 wird nach einigen Bemerkungen zur Rechnerarithmetik an Begriffe wie *faktorisierbare Funktion*, *Codeliste*, *Darstellungsgraph* erinnert. Diese werden bei der Darstellung des Prinzips der sogenannten Rückwärtsmethode bzw. dessen Realisierung auf einer Rechenanlage benötigt.

Abschnitt 3 erläutert das allgemeine Prinzip des Rückwärtsrechnens. Im zentralen Abschnitt 4 wird dieses Prinzip zur Berechnung von worst-case Fehlerschranken herangezogen. Dabei werden Vergleiche bzgl. numerischer Ergebnisse und Handhabbarkeit der Rückwärtsmethode mit einem Vorwärtskalkül angestellt.
Im Anhang sind beispielhaft der detaillierte Algorithmus zur Berechnung der arctan-Funktion und die Daten für die Fehlerabschätzung dieses Algorithmus angegeben.

2 Grundlegende Definitionen

2.1 Rechnerarithmetik

Bedingt durch die Endlichkeit des Speichers kann auf der Maschine nur mit einer endlichen Teilmenge S der reellen Zahlen gerechnet werden. Typischerweise bezeichnet S die Menge der sogenannten Gleitkommazahlen (vgl. [25]).

Definition 2.1 (Gleitkommasystem) *Unter einem Gleitkommasystem $S = S(b, n, e_1, e_2) \subseteq \mathbb{R}$ versteht man die Menge aller auf einer Rechenanlage darstellbaren normalisierten Gleitkommazahlen x mit den folgenden Eigenschaften:*

1. $x = m \cdot b^e$
2. $m := \pm 0.d_1 d_2 \ldots d_n$
3. $1 \leq d_1 \leq b-1$, $0 \leq d_i \leq b-1$ $(2 \leq i \leq n)$
4. $e_1 \leq e \leq e_2$, $e_1 \leq 0$ *und* $1 \leq e_2$, *wobei* e, e_1, e_2 *ganze Zahlen sind.*

Dabei heißt m die Mantisse mit Länge $n \in \mathbb{N}$, $b \in \mathbb{N}$ $(b > 1)$ die Basis und e der Exponent der Gleitkommazahl x. Ferner existiert eine Darstellung der 0, welche darin besteht, daß die Mantisse den Wert 0 und der Exponent den Wert e_1 hat[1].

Bemerkung 2.1 *Mit x_{min} wird die kleinste und mit x_{max} die größte positive normalisierte Gleitkommazahl bezeichnet. Der Unterlaufbereich ist die Menge $\{x \in \mathbb{R} \mid |x| \in (0, x_{min})\}$. Man definiert $\mathbb{R}_B := \{x \in \mathbb{R} \mid -x_{max} \leq x \leq x_{max}\}$. Gelegentlich werden auch denormalisierte Gleitkommazahlen betrachtet. Diese Zahlen sind dadurch gekennzeichnet, daß die Forderung $1 \leq d_1$ in obiger Definition abgeschwächt und durch $0 \leq d_1$ ersetzt wird.*

Reelle Zahlen müssen bei Berechnungen auf der Maschine durch Gleitkommazahlen ersetzt werden. Dazu sind sie durch eine geeignete Abbildungsvorschrift in die Menge der Gleitkommazahlen zu projizieren.

Definition 2.2 (Rundung) *Eine Abbildung $\square : \mathbb{R}_B \to S$ heißt Rundung, falls für alle $x \in S$ die Gleichung*

$$\square(x) = x$$

erfüllt ist.

[1] Man definiert $0 := +0.0 \ldots 0 \cdot b^{e_1}$.

Im allgemeinen erhält man beim Verknüpfen zweier Gleitkommazahlen durch Operationen aus $\{+,-,\cdot,/\}$ keine Gleitkommazahl. Vielmehr müssen die (exakten) Ergebnisse derartiger Operationen wieder ins Gleitkommasystem gerundet werden:

Definition 2.3 (Gleitkommaoperation) *Sei S ein Gleitkommasystem mit Rundung $\square : \mathbb{R}_B \to S$. Für $x, y \in S$ mit $x \circ y \in \mathbb{R}_B$ wird die zu $\circ \in \{+,-,\cdot,/\}$ gehörende Gleitkommaoperation $\boxed{\circ}$ durch*

$$x \boxed{\circ} y := \square(x \circ y)$$

definiert, falls kein Unterlauf entsteht.

Auf dem Rechner stehen verschiedene Gleitkommazahlensysteme zur Verfügung. Die wichtigste Rolle spielt das IEEE-`double`-Format $S = S(2, 53, -1022, 1023)$[2]. Es ist durch den IEEE-Standard 754 genormt. Jede Gleitkommazahl $x \in S$ läßt sich hier durch

$$x = (-1)^{s(x)} \cdot m(x) \cdot 2^{e(x)} \tag{1}$$

mit $s(x) \in \{0,1\}$, $m(x) = d_0.d_1 \ldots d_{52}$ und $-1022 \le e(x) \le 1023$ darstellen. Ist $d_0 = 1$, so heißt x normalisierte, ansonsten denormalisierte Zahl. Die kleinste darstellbare normalisierte positive Zahl wird hier mit `MinReal`, die größte mit `MaxReal` bezeichnet. Der Bereich N der normalisierten Gleitkommazahlen ist durch

$$N := [-\texttt{MaxReal}, -\texttt{MinReal}] \cup [\texttt{MinReal}, \texttt{MaxReal}]$$

erklärt. Der Unterlaufbereich ist die Menge aller Gleitkommazahlen aus

$$U := (-\texttt{MinReal}, \texttt{MinReal}).$$

Laut Standard müssen verschiedene Rundungen (Rundung zur nächstgelegenen Gleitkommazahl und gerichtete Rundungen) zur Verfügung stehen. Für Verknüpfungen $\circ \in \{+,-,\cdot,/\}$ werden an die zugehörigen Gleitkommaoperationen $\boxed{\circ}$ mit $x, y \in S$ die folgenden Anforderungen gestellt:

$$\left.\begin{array}{lll} \left|\frac{x \circ y - x \boxed{\circ} y}{x \circ y}\right| \le \varepsilon & , \text{falls} & x \circ y \in N \\ |x \circ y - x \boxed{\circ} y| \le \texttt{MinReal} & , \text{falls} & x \circ y \in U \\ x \boxed{\circ} y = x \circ y & , \text{falls} & x \circ y \in S. \end{array}\right\} \tag{2}$$

Die Größe ε entspricht dabei dem Maschinenepsilon und beträgt je nach Rundungsmodus $\varepsilon := 2^{-53}$ (Rundung zur nächstgelegenen Gleitkommazahl) oder $\varepsilon := 2^{-52}$ (gerichtete Rundung nach unten bzw. nach oben). Für Funktionen f, z.B. $\exp, \sin$, etc., wird eine Genauigkeitsforderung gemäß (2) nicht verlangt. In dieser Arbeit wird jedoch davon ausgegangen, daß für solche Funktionen Konstanten $\varepsilon(f)$ und $u(f)$ bekannt sind, welche die folgenden Eigenschaften erfüllen:

$$\left.\begin{array}{lll} |f(x) - \widetilde{f}(x)| \le \varepsilon(f) \cdot |f(x)| & , \text{für} & x \in S \text{ mit } f(x) \in N \\ |f(x) - \widetilde{f}(x)| \le u(f) & , \text{für} & x \in S \text{ mit } f(x) \in U\,. \end{array}\right\} \tag{3}$$

(vgl. [5], [12], [20]). Dabei ist $\widetilde{f} : S \to S$ eine zu f gehörende Gleitkommanäherung.

[2] Im folgenden sind Gleitkommazahlen immer als Maschinenzahlen im IEEE-`double`-Format anzusehen.

2.2 Faktorisierbarkeit von Funktionen

Bei der Lösung mathematischer Probleme durch eine Rechenanlage verwendet man Programmiersprachen wie z. B. Fortran, C oder C++. Dazu werden die benötigten Funktionen (Algorithmen) in einer dieser Sprachen kodiert und dann durch einen Compiler oder Interpreter in ein ausführbares Programm umgewandelt. Grundsätzlich können nur solche Funktionen ausgewertet werden, die mit Sprachkomponenten beschreibbar sind, welche die verwendete Programmiersprache bereitstellt, also z. B. durch Aufrufe von elementaren Operationen, Standardfunktionen, Bedingungsanweisungen und Schleifen. In diesem Zusammenhang definiert man eine eingeschränkte Klasse solcher Funktionen, nämlich die Klasse der faktorisierbaren Funktionen. Doch zunächst zwei weitere Definitionen.

Definition 2.4 (elementare Operationen $\mathcal{OP}$) *Die Menge aller unären und binären reellen elementaren Operationen* $\{-, +, -, \cdot, /\}$ *wird mit* $\mathcal{OP}$ *bezeichnet.*

Definition 2.5 (Funktionenmenge $\mathcal{F}$) *Das Symbol* $\mathcal{F}$ *bezeichnet die Menge der reellen Standardfunktionen* {sqr(.), sqrt(.), sqrt(. , .), power(. , .), sin(.), cos(.), tan(.), cot(.), arcsin(.), arccos(.), arctan(.), arccot(.), exp(.), ln(.), sinh(.), cosh(.), tanh(.), coth(.), arsinh(.), arcosh(.), artanh(.), arcoth(.)}.

Mit Hilfe dieser Definitionen wird nun die Klasse der faktorisierbaren Funktionen erklärt. Bei solchen Funktionen sind die Operanden nur durch Operationen aus $\mathcal{OP}$ und $\mathcal{F}$ zusammengesetzt. Die folgende Definition lehnt sich eng an die von Kedem [18] an.

Definition 2.6 (Faktorisierbare Funktion) *Sei* $\pi_i^n : \mathbb{R}^n \to \mathbb{R}$ *die Projektion eines reellen n-Tupels auf seine i-te Komponente, d.h.* $\pi_i^n(x_1, x_2, \ldots, x_n) = x_i$. *Eine Funktion* $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ *heißt faktorisierbar, falls es eine endliche Folge von Funktionen* $f_1, \ldots, f_k$ *gibt, für die gilt:*

(1) $f_j : D \to \mathbb{R}$ *für alle* $j \in \{1, \ldots, k\}$,
(2) $f = f_k$,
(3) $f_1 = \pi_1^n, f_2 = \pi_2^n, \ldots, f_n = \pi_n^n$ *und*
(4) für $n < j \leq k$: $f_j = g(f_l)$ *mit* $g \in \mathcal{F}$ *und* $l < j$ *oder*
$f_j = f_l \circ f_r$ *mit* $\circ \in \mathcal{OP}$[3] *und* $l, r < j$ *oder*
f_j *ist eine reelle Konstante.*

Ein Beispiel soll diese Definition verdeutlichen.

Beispiel 2.1 *Sei* $f : \mathbb{R}^3 \to \mathbb{R}$ *mit* $f(x_1, x_2, x_3) = 2 \cdot \sin(x_1 \cdot x_3) + \mathrm{e}^{{x_2}^2}$ *gegeben. Dann kann f mittels*

$$\begin{aligned} f_1 &= x_1 = \pi_1^3(x_1, x_2, x_3) \\ f_2 &= x_2 = \pi_2^3(x_1, x_2, x_3) \\ f_3 &= x_3 = \pi_3^3(x_1, x_2, x_3) \end{aligned}$$

[3] Entspricht $\circ$ dem unären $-$, so wird immer davon ausgegangen, daß nur ein Operand in die Rechnung eingeht.

$$\begin{aligned} f_4 &= f_1 \cdot f_3 \\ f_5 &= \sin(f_4) \\ f_6 &= 2 \\ f_7 &= f_6 \cdot f_5 \\ f_8 &= \mathrm{sqr}\,(f_2) \\ f_9 &= \exp(f_8) \\ f &= f_{10} = f_7 + f_9 \end{aligned}$$

zerlegt werden.

Bemerkung 2.2 *Eine solche Darstellung einer zerlegbaren Funktion wird Codeliste genannt. Die Funktionen $f_1, \ldots, f_n$ werden als Unabhängige oder Veränderliche von f bezeichnet. Die Ergebnisfunktion f_k heißt Abhängige von f. Alle anderen Funktionen der Funktionenfolge $f_1, \ldots, f_k$ heißen Zwischenergebnisse der Codeliste. Eine Codeliste ist im allgemeinen nicht eindeutig bestimmt. Für die Funktion des obigen Beispiels ist*

$$\begin{aligned} f_1 &= x_1 \\ f_2 &= x_2 \\ f_3 &= x_3 \\ f_4 &= f_1 \cdot f_3 \\ f_5 &= 2 \\ f_6 &= \sin(f_4) \\ f_7 &= \mathrm{sqr}\,(f_2) \\ f_8 &= \exp(f_7) \\ f_9 &= f_5 \cdot f_6 \\ f &= f_{10} = f_9 + f_8 \end{aligned}$$

eine weitere Darstellung. Die Auswertungsreihenfolge auf dem Rechner wird u.a. durch die Grammatik der benutzten Programmiersprache festgelegt. Vorsicht ist bei optimierenden Compilern geboten. Hier kann eine gewünschte Auswertungsreihenfolge durch entsprechende Klammerung erzwungen werden.

Bemerkung 2.3 *Funktionen der Form $g : \mathbb{R}^n \to \mathbb{R}^m$ heißen faktorisierbar, falls alle Komponentenfunktionen g_j $(1 \leq j \leq m)$ von g faktorisierbar sind.*

Bemerkung 2.4 *Funktionen, die Intervalle als Argumente haben, können in gleicher Art und Weise wie in Definition 2.6 als faktorisierbar erklärt werden. Dann sind die elementaren Operationen und Funktionen als Intervalloperationen bzw. Intervallfunktionen aufzufassen und werden ebenfalls mit $\mathcal{OP}$ bzw. $\mathcal{F}$ bezeichnet.*

2.3 Darstellungsgraph

Äquivalent zur Beschreibung faktorisierbarer Funktionen durch eine Codeliste ist die visuelle Darstellung mittels eines Graphen. Ein solcher Darstellungsgraph veranschaulicht die Codeliste in einem gerichteten, zyklenfreien Graphen[4]. Abbildung 1 (S.

[4]Eine genaue Definition ist z.B. in [27] zu finden.

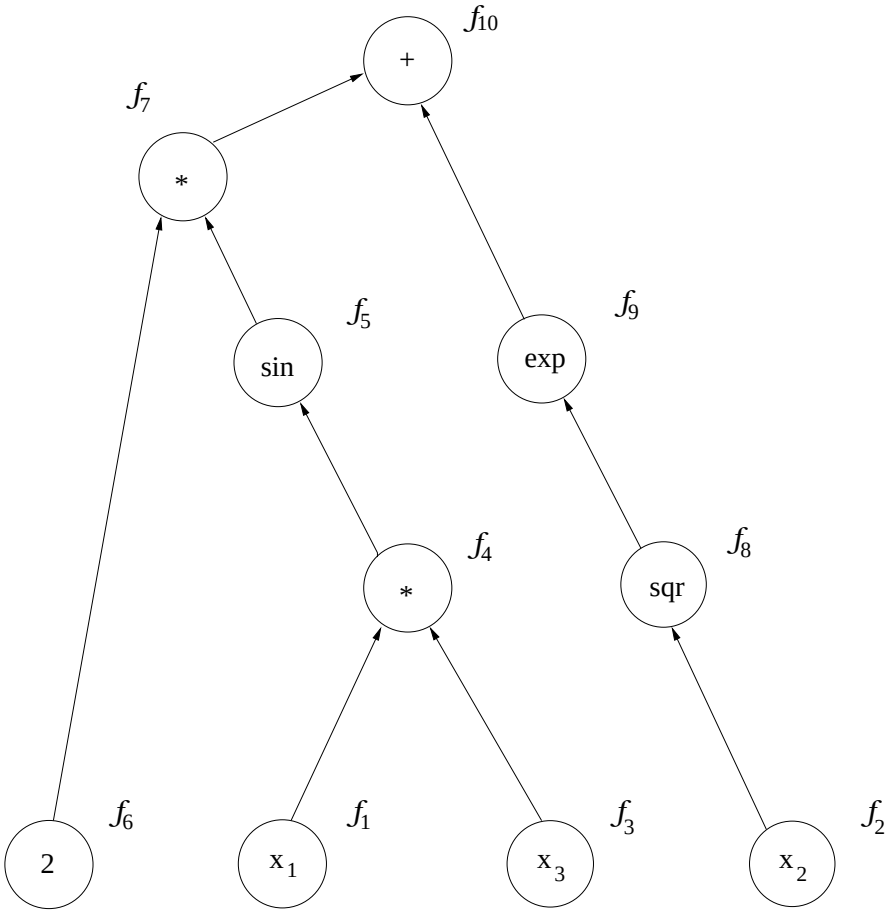


Abbildung 1: Darstellungsgraph der Funktion aus Bsp. 3.1.1

8) zeigt den Darstellungsgraphen der Funktion $f : \mathbb{R}^3 \to \mathbb{R}$ mit $f(x_1, x_2, x_3) = 2 \cdot \sin(x_1 \cdot x_3) + \mathrm{e}^{{x_2}^2}$ aus Beispiel 2.1. Wie man ihr entnehmen kann, ist jeder Operation aus $\mathcal{F}$ oder $\mathcal{OP}$ ein Knoten zugeordnet. Jeder dieser Knoten hat entweder einen oder zwei Vorgänger, abhängig davon, ob dieser Knoten eine unäre oder binäre Operation repräsentiert. Nur die Knoten der in der Funktion vorkommenden Konstanten und der Eingangsgrößen, welche die Unabhängigen darstellen, haben keine Vorgänger. Der Knoten ohne Nachfolger wird als Ergebnisknoten (Wurzel) bezeichnet. Seine Operation liefert den Wert von f. Offensichtlich gibt es zu jeder faktorisierbaren Funktion einen Darstellungsgraphen und umgekehrt.

Im allgemeinen treten in Computerprogrammen Funktionen nicht nur in Form zerlegbarer Funktionen auf. Wie man Definition 2.6 entnehmen kann, dürfen Bedingungsanweisungen und Schleifen[5] innerhalb faktorisierbarer Funktionen nicht vorkommen. Auf Verallgemeinerungen und dabei auftretende zusätzliche Probleme wird in [6] näher eingegangen.

[5]Damit sind Schleifen gemeint, bei denen die Anzahl der Schleifendurchläufe erst zur Laufzeit bekannt ist.

3 Allgemeines Prinzip der Rückwärtsrechnung

3.1 Ein einführendes Beispiel

Das Prinzip der Rückwärtsrechnung wird zunächst anhand der laufzeiteffizienten Berechnung des Gradienten einer Funktion veranschaulicht (Literatur zum Thema *Automatische Differentiation* [4, 8, 7, 9].) Dazu sei die in $\mathbb{R}^3$ beliebig oft differenzierbare und faktorisierbare Funktion $f : \mathbb{R}^3 \to \mathbb{R}$ mit $f(x_1, x_2, x_3) = x_3 + \sin(x_1 \cdot x_2)$ gegeben. f läßt sich zum Beispiel mittels der Codeliste

$$\begin{aligned} f_1 &= x_1 && (4)\\ f_2 &= x_2 && (5)\\ f_3 &= x_3 && (6)\\ f_4 &= f_1 \cdot f_2 && (7)\\ f_5 &= \sin(f_4) && (8)\\ f_6 &= f_3 + f_5\,. && (9) \end{aligned}$$

beschreiben. Differenziert man nun (4) bis (9) nach x_1, x_2 und x_3, so erhält man das Gleichungssystem

$$\left.\begin{array}{lll} \frac{\partial f_1}{\partial x_1} = 1 & \frac{\partial f_1}{\partial x_2} = 0 & \frac{\partial f_1}{\partial x_3} = 0 \\ \frac{\partial f_2}{\partial x_1} = 0 & \frac{\partial f_2}{\partial x_2} = 1 & \frac{\partial f_2}{\partial x_3} = 0 \\ \frac{\partial f_3}{\partial x_1} = 0 & \frac{\partial f_3}{\partial x_2} = 0 & \frac{\partial f_3}{\partial x_3} = 1 \\ \frac{\partial f_4}{\partial x_1} = \frac{\partial f_1}{\partial x_1} f_2 + \frac{\partial f_2}{\partial x_1} f_1 & \frac{\partial f_4}{\partial x_2} = \frac{\partial f_1}{\partial x_2} f_2 + \frac{\partial f_2}{\partial x_2} f_1 & \frac{\partial f_4}{\partial x_3} = \frac{\partial f_1}{\partial x_3} f_2 + \frac{\partial f_2}{\partial x_3} f_1 \\ \frac{\partial f_5}{\partial x_1} = \frac{\partial f_4}{\partial x_1}\cos(f_4) & \frac{\partial f_5}{\partial x_2} = \frac{\partial f_4}{\partial x_2}\cos(f_4) & \frac{\partial f_5}{\partial x_3} = \frac{\partial f_4}{\partial x_3}\cos(f_4) \\ \frac{\partial f_6}{\partial x_1} = \frac{\partial f_3}{\partial x_1} + \frac{\partial f_5}{\partial x_1} & \frac{\partial f_6}{\partial x_2} = \frac{\partial f_3}{\partial x_2} + \frac{\partial f_5}{\partial x_2} & \frac{\partial f_6}{\partial x_3} = \frac{\partial f_3}{\partial x_3} + \frac{\partial f_5}{\partial x_3} \end{array}\right\} \qquad (10)$$

durch Anwendung der elementaren Differentiationsregeln. (10) ist in Matrixform gleichbedeutend mit

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -f_2 & -f_1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\cos(f_4) & 1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 1 \end{pmatrix}}_{=I-\alpha=:A} \cdot \underbrace{\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} \\ \frac{\partial f_5}{\partial x_1} & \frac{\partial f_5}{\partial x_2} & \frac{\partial f_5}{\partial x_3} \\ \frac{\partial f_6}{\partial x_1} & \frac{\partial f_6}{\partial x_2} & \frac{\partial f_6}{\partial x_3} \end{pmatrix}}_{=:H=(h^1,\,h^2,\,h^3)} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{=:Z=(z^1,\,z^2,\,z^3)},$$

wobei

$$\alpha := \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ f_2 & f_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos(f_4) & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

eine untere Dreiecksmatrix darstellt, deren Diagonalelemente Null sind. Die Berechnung des Gradienten von f an einer bestimmten Stelle $x_0 \in \mathbb{R}^3$ entspricht somit dem Lösen der drei linearen Gleichungssysteme

$$\begin{aligned} A\,h^1 &= z^1 && (11)\\ A\,h^2 &= z^2 && (12)\\ A\,h^3 &= z^3. && (13) \end{aligned}$$

Diese Systeme können durch Vorwärtseinsetzen gelöst werden. Zum einen gilt für $1 \leq l \leq 3$:

$$\begin{aligned} h_i^l &= z_i^l = \delta_{li} \;\; (1 \leq i \leq 3) \text{ und} && (14)\\ h_i^l &= \sum_{j=1}^{i-1} \alpha_{ij} h_j^l + z_i^l && (15)\\ &= \sum_{j=1}^{i-1} \alpha_{ij} h_j^l \;\; (4 \leq i \leq 6). && (16) \end{aligned}$$

Zum anderen ist $\det(I-\alpha) \neq 0$. Für $D = (d_{ij})_{1\leq i,j\leq 6} := (I-\alpha)^{-1}$ ist wegen (11)-(13) $D\,z^l = h^l$ $(1 \leq l \leq 3)$. Es gilt:

$$h_i^l = \sum_{j=1}^{6} d_{ij} z_j^l \;\; (1 \leq i \leq 6)$$

und speziell

$$\frac{\partial f_6}{\partial x_l} = h_6^l = \sum_{j=1}^{6} d_{6j} z_j^l = d_{6j} \cdot \delta_{jl}, \qquad (17)$$

also

$$\text{grad}(f)(x_0) = (d_{61},\, d_{62},\, d_{63}).$$

Die Größen d_{ij} $(1 \leq i,j \leq 6)$ lassen sich wegen

$$\begin{aligned} D\,(I-\alpha) &= I\\ D &= I + D\,\alpha && (18) \end{aligned}$$

wieder rekursiv berechnen. (18) ist gleichbedeutend mit

$$d_{ij} = \begin{cases} 0 & , \quad i < j \\ 1 & , \quad i = j \\ \displaystyle\sum_{m=j+1}^{i} d_{im}\,\alpha_{mj} & , \quad j = i-1, \ldots, 1 \end{cases}$$

und speziell für $i = 6$ kann man die Größen d_{6j} $(j = 5, \ldots, 1)$ beginnend mit $d_{66} := 1$ und $d_{6j} := 0$ $(1 \leq j \leq 5)$ rekursiv ermitteln.

3.2 Der allgemeine Fall

Hat man einen Algorithmus gegeben, bei dem die Funktionswerte der Codeliste als Lösung eines linearen Gleichungssystem darstellbar sind und es sich bei der Systemmatrix um eine untere Dreiecksmatrix handelt, können die Aussagen des letzten Abschnitts generalisiert werden. Die folgenden Aussagen gehen auf [8] zurück. Gegeben sei das System $(\Gamma - \alpha)\, h = z$, welches folgende Gestalt besitzt:

$$\begin{pmatrix} \frac{1}{\gamma_1} & 0 & 0 & \cdots & 0 & 0 \\ -\alpha_{21} & \frac{1}{\gamma_2} & 0 & \cdots & 0 & 0 \\ -\alpha_{31} & -\alpha_{32} & \frac{1}{\gamma_3} & \cdots & 0 & 0 \\ \vdots & & \ddots & \ddots & \vdots & \vdots \\ \vdots & & & \ddots & \frac{1}{\gamma_{s-1}} & 0 \\ -\alpha_{s1} & \cdots & \cdots & \cdots & -\alpha_{ss-1} & \frac{1}{\gamma_s} \end{pmatrix} \cdot \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_{s-1} \\ h_s \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{s-1} \\ z_s \end{pmatrix} \tag{19}$$

Es ist $\Gamma := (\delta_{ij} \cdot \gamma_i^{-1})_{1 \le i,j \le s} \in I\!R^{s \times s}$, $\gamma_i \neq 0$ für $1 \le i \le s$ und $\alpha := (\alpha_{ij})_{1 \le i,j \le s}$ mit $\alpha \in I\!R^{s \times s}$, wobei $\alpha_{ij} = 0$ für $i \le j$. h und z sind Vektoren des $I\!R^s$ und h kann wieder wegen der speziellen Bauart der Matrix $\Gamma - \alpha$ durch Vorwärtsauflösen analog zu (14) und (15) bestimmt werden. Es gelten wegen (19) die Gleichungen:

$$\begin{aligned} h_1 &= \gamma_1\, z_1 \\ h_i &= \gamma_i \left(\sum_{j=1}^{i-1} \alpha_{ij} h_j + z_i \right) \quad (1 < i \le s). \end{aligned}$$

Wie im speziellen Fall der Gradientenberechnung kann man (19) wieder rückwärts auflösen. Sei dazu $D = (d_{ij})_{1 \le i,j \le s} := (\Gamma - \alpha)^{-1}$. Aufgrund von $\det(\Gamma - \alpha) = \det(\Gamma) \neq 0$ existieren D und Γ^{-1}. Es gilt:

$$\begin{aligned} D\,(\Gamma - \alpha) &= I \\ D\,\Gamma &= I + D\,\alpha \\ D &= (I + D\,\alpha)\,\Gamma^{-1}. \end{aligned}$$

Zusammenfassend kann der folgende Satz formuliert werden (entnommen aus [8]):

Satz 3.1 *Seien Γ, α, $D \in I\!R^{s \times s}$ $(s \in I\!N)$ wie oben definiert. Mit $z \in I\!R^s$ gilt für die Lösung $h \in I\!R^s$ des linearen Gleichungssystems $(\Gamma - \alpha)\, h = z$:*

$$\begin{aligned} h_1 &= \gamma_1\, z_1 \\ h_i &= \gamma_i \left(\sum_{j=1}^{i-1} \alpha_{ij} h_j + z_i \right) \quad (1 < i \le s) \\ &= \sum_{j=1}^{s} d_{ij}\; z_j, \end{aligned} \tag{20}$$

wobei die Berechnung der Elemente der Matrix D rekursiv durch

$$d_{ij} = 0 \;\; (i < j)\,,$$

$$d_{ij} = \gamma_j \ (i = j)\,, \tag{21}$$

$$d_{ij} = \gamma_j \sum_{m=j+1}^{i} d_{im}\alpha_{mj} \ (j = i-1,\ i-2, \ldots, 1) \tag{22}$$

erfolgen kann.

Hat man einmal alle d_{ij} $(1 \leq j \leq i \leq s)$ berechnet, so kann man mit (20) das Gleichungssystem (19) für verschiedene rechte Seiten z auswerten. Dies wurde bei der Berechnung des Gradienten im letzten Abschnitt benutzt (vgl. (17)) und begründet die Überlegenheit der Rückwärtsmethode gegenüber der Vorwärtsmethode hinsichtlich des Aufwandes zur Berechnung des Gradienten.

Setzt man in (21) und (22) speziell $i = s$, so läßt sich das folgende Korollar ableiten.

Korollar 3.1 (allgemeine Rückwärtsmethode) *Für $s \in \mathbb{N}$ sei γ_i, $z_i \in \mathbb{R}$ $(1 \leq i \leq s)$ und $\alpha_{ij} \in \mathbb{R}$ $(1 \leq i, j \leq s)$ mit $\alpha_{ij} = 0$ $(i \leq j)$, $\gamma_i \neq 0$ $(1 \leq i \leq s)$ gegeben. Ist dann die (endliche) Folge $h_1, \ldots, h_s$ durch*

$$h_1 := \gamma_1 z_1 \tag{23}$$

$$h_i := \gamma_i \left(\sum_{j=1}^{i-1} \alpha_{ij} h_j + z_i \right) \ (1 < i \leq s) \tag{24}$$

definiert, so gilt:

$$\begin{aligned} h_s &= \sum_{m=1}^{s} d_m z_m \ mit \\ d_s &= \gamma_s \ und \\ d_j &= \gamma_j \sum_{m=j+1}^{s} d_m \alpha_{mj} \ (j = s-1, \ldots, 1). \end{aligned}$$

Beweis: Siehe [8]. □

Mit Hilfe des Korollars hat man für Algorithmen, welche durch (23) und (24) beschreibbar sind, die Möglichkeit, die Berechnung durch Rückwärtseinsetzen durchzuführen. Als Anwendungen dieses Korollars wird im nächsten Abschnitt ein Verfahren zur automatischen Fehlerabschätzung von Gleitkommaalgorithmen vorgestellt. In [6] wird die Methodik auch zur schnellen Berechnung von Intervallsteigungen (siehe auch [28, 29]) herangezogen.

4 Automatische Fehlerkontrolle

4.1 Problemstellung

Hofschuster und Krämer leiten in [12] ein Fehlerkalkühl her, um maximale Fehler von auf der Maschine implementierten Standardfunktionen abzuschätzen. Dieses Verfahren

beruht auf dem Vorwärtsdurchlaufen des Darstellungsgraphen. Iri verwendet mittels Rückwärtsrechnung gewonnene Fehlerschranken, um optimale Abbruchkriterien für Iterationsverfahren zu ermitteln (vgl. [26]). Weiterführende Literatur zur automatischen Fehlerkontrolle findet man in [1], [2], [14], [15], [21], [22], [23] und [24].
In diesem Kapitel soll ein Verfahren zur automatischen Berechnung einer Gesamtfehlerschranke einer faktorisierbaren Funktion durch Rückwärtsrechnung vorgestellt werden. Um Korollar 3.1 benutzen zu können, wird zunächst in Analogie zu [8] ein Verfahren hergeleitet, welches den Gesamtfehler durch Vorwärtsrechnung berechnet. Aus diesem soll anschließend das gewünschte Verfahren abgeleitet werden. Desweiteren wird im letzten Abschnitt dieses Kapitels der gewonnene Algorithmus mit einem bereits implementierten Verfahren (siehe dazu [1], [2]) zur sicheren Abschätzung einer Gesamtfehlerschranke einer faktorisierbaren Funktion anhand einiger numerischer Beispiele verglichen.

4.2 Fehlerkontrolle durch Vorwärtsrechnung

Für die Herleitung sind einige Definitionen notwendig. Es wird ähnlich wie in [8] vorgegangen.

Definition 4.1 *Mit $\widetilde{\mathcal{OP}}$ und $\widetilde{\mathcal{F}}$ werden die zu $\mathcal{OP}$ und $\mathcal{F}$ entsprechenden Operationen auf der Maschine bezeichnet. Dabei entspricht $\widetilde{\mathcal{OP}}$ der in Definition 2.3 vorgestellten Menge der Gleitkommaoperationen (vgl. S. 5).*

Definition 4.2 (Datenfehler) *Seien $\widetilde{f_l} \in S$ und $\widetilde{f_r} \in S$ Näherungen von f_l und f_r. Dann ist der absolute Datenfehler $\triangle_{D,f}$ für die elementaren Operationen und Funktionen durch*

$$\begin{aligned} \triangle_{D,f} &:= (\widetilde{f_l} \circ \widetilde{f_r}) - (f_l \circ f_r), \quad f = f_l \circ f_r \\ \triangle_{D,f} &:= g(\widetilde{f_l}) - g(f_l), \quad f = g(f_l) \end{aligned}$$

erklärt.

Definition 4.3 (Rundungsfehler) *Gegeben seien die Gleitkommanäherungen $\widetilde{f_l}$, $\widetilde{f_r}$ von f_l und f_r. Für $f = f_l \circ f_r$ ist der absolute Rundungsfehler $\triangle_{R,f}$ durch*

$$\triangle_{R,f} := (\widetilde{f_l} \boxdot \widetilde{f_r}) - (\widetilde{f_l} \circ \widetilde{f_r}), \quad \circ \in \mathcal{OP}, \boxdot \in \widetilde{\mathcal{OP}}$$

bestimmt. Ist $f = g(f_l)$, so definiert man $\triangle_{R,f}$ mittels

$$\triangle_{R,f} := \widetilde{g}(\widetilde{f_l}) - g(\widetilde{f_l}), \quad g \in \mathcal{F}, \widetilde{g} \in \widetilde{\mathcal{F}}.$$

Definition 4.4 (absoluter Gesamtfehler) *Sei die faktorisierbare Funktion $f : \mathbb{R}^n \to \mathbb{R}$ mit einer Codeliste $f_1, \ldots, f_k$ und deren Näherung $\widetilde{f}$ mit Codeliste $\widetilde{f_1}, \ldots, \widetilde{f_k}$ gegeben. Weiterhin sei*

$$\begin{aligned} f_j = x_j \quad &bzw. \quad \widetilde{f_j} = \widetilde{x_j} \quad für \quad 1 \le j \le n \quad und\ o.B.d.A. \\ f_j = c_j \quad &bzw. \quad \widetilde{f_j} = \widetilde{c_j} \quad für \quad n+1 \le j \le n+m < k \end{aligned}$$

gesetzt. Dann ist der Gesamtfehler $\triangle_{G,f_j}$ für $1 \le j \le k$ wie folgt definiert:

$$\triangle_{G,f_j} := \begin{cases} \widetilde{f_j} - f_j & , \quad 1 \le j \le n+m \\ \widetilde{f_j}(\widetilde{x_1}, \ldots, \widetilde{x_n}) - f_j(x_1, \ldots, x_n) & , \quad n+m+1 \le j \le k. \end{cases}$$

Bemerkung 4.1 *Aus dem Betrag des absoluten Gesamtfehlers ergibt sich der Betrag des relativen Gesamtfehlers $\triangle^{rel}_{G,f_j}$ unter den Voraussetzungen von 4.4 durch*

$$|\triangle^{rel}_{G,f_j}| := \frac{|\triangle_{G,f_j}|}{|f_j|}, \quad f_j \neq 0,\ 1 \leq j \leq k.$$

Die beiden folgenden Lemmata und der darauffolgende Satz beantworten die Frage, wie der Datenfehler und der Gesamtfehler bei elementaren Operationen und Funktionen aus den Datenfehlern, Rundungsfehlern bzw. den Gesamtfehlern der Operanden entstehen.

Lemma 4.1 *Für $n+m+1 \leq j \leq k$ gilt mit den Voraussetzungen aus Definition 4.4:*

$$\triangle_{G,f_j} = \triangle_{D,f_j} + \triangle_{R,f_j}.$$

Beweis: Durch Anwendung der Definitionen folgt die Behauptung. □

Lemma 4.2 *Seien die Näherungen $\widetilde{f_l} \in S$ und $\widetilde{f_r} \in S$ von f_l und f_r und bekannte Fehlerschranken $\triangle_{G,f_l} = \widetilde{f_l} - f_l$ und $\triangle_{G,f_r} = \widetilde{f_r} - f_r$ gegeben. Dann gilt für den Datenfehler $\triangle_{D,f}$ bei Anwendung einer elementaren Operation:*

$$\triangle_{D,f} = -\triangle_{G,f_l}\ ,\ falls\ f = -f_l \tag{25}$$
$$\triangle_{D,f} = \triangle_{G,f_l} + \triangle_{G,f_r}\ ,\ falls\ f = f_l + f_r \tag{26}$$
$$\triangle_{D,f} = \triangle_{G,f_l} - \triangle_{G,f_r}\ ,\ falls\ f = f_l - f_r \tag{27}$$
$$\triangle_{D,f} = \widetilde{f_r}\triangle_{G,f_l} + f_l\triangle_{G,f_r}\ ,\ falls\ f = f_l \cdot f_r \tag{28}$$
$$\triangle_{D,f} = (\triangle_{G,f_l} - (f_l/f_r)\ \triangle_{G,f_r})/\widetilde{f_r}\ ,\ falls\ f = f_l/f_r,\ (\widetilde{f_r},\ f_r \neq 0). \tag{29}$$

Für den Datenfehler bei Anwendung elementarer Funktionen $g \in \mathcal{F}$ gilt:

$$\triangle_{D,f} = g'(\xi)\ \triangle_{G,f_l}\ ,\ falls\ f = g(f_l). \tag{30}$$

Dabei wird vorausgesetzt, daß $g'(\xi)$ für alle $\xi \in f_l \underline{\cup} \widetilde{f_l}$ existiert.

Beweis:
Die Formeln (25)-(30) lassen sich durch Anwendung der Definitionen zeigen. In (30) benötigt man zusätzlich den Mittelwertsatz der Differentialrechnung. Hier wird nur der Beweis von (28) durchgeführt. Wegen Definition 4.2 gilt:

$$\begin{aligned}
\triangle_{D,f} &= \widetilde{f_l}\ \widetilde{f_r} - f_l\ f_r \\
&= \widetilde{f_l}\ \widetilde{f_r} - \widetilde{f_r}\ f_l + \widetilde{f_r}\ f_l - f_l\ f_r \\
&= \widetilde{f_r}(\widetilde{f_l} - f_l) + (\widetilde{f_r} - f_r)f_l \\
&= \widetilde{f_r}\triangle_{G,f_l} + f_l\triangle_{G,f_r}.
\end{aligned}$$

Somit folgt die Behauptung. □

Aus den beiden Lemmata und Definition 4.4 ergibt sich

Satz 4.1 *Gelten die Voraussetzungen von Definition 4.4 und Lemma 4.2, so erhält man für den Gesamtfehler der j-ten Operation der Codeliste:*

$$\begin{aligned}
\triangle_{G,f_j} &= \widetilde{f_j} - f_j = \triangle_{R,f_j} \text{ , } falls\ 1 \le j \le n+m && (31)\\
\triangle_{G,f_j} &= -\triangle_{G,f_l} + \triangle_{R,f_j} \text{ , } falls\ f_j = -f_l && (32)\\
\triangle_{G,f_j} &= \triangle_{G,f_l} + \triangle_{G,f_r} + \triangle_{R,f_j} \text{ , } falls\ f_j = f_l + f_r && (33)\\
\triangle_{G,f_j} &= \triangle_{G,f_l} - \triangle_{G,f_r} + \triangle_{R,f_j} \text{ , } falls\ f_j = f_l - f_r && (34)\\
\triangle_{G,f_j} &= \widetilde{f_r}\triangle_{G,f_l} + f_l\triangle_{G,f_r} + \triangle_{R,f_j} \text{ , } falls\ f_j = f_l \cdot f_r && (35)\\
\triangle_{G,f_j} &= (\triangle_{G,f_l} - (f_l/f_r)\ \triangle_{G,f_r})/\widetilde{f_r} + \triangle_{R,f_j} \text{ , } falls\ f_j = f_l/f_r && (36)\\
\triangle_{G,f_j} &= g'(\xi)\ \triangle_{G,f_l} + \triangle_{R,f_j} \text{ , } falls\ f_j = g(f_l),\ \xi \in f_l \sqcup \widetilde{f}_l. && (37)
\end{aligned}$$

4.3 Fehlerkontrolle durch Rückwärtsrechnung

Die Gleichungen (31)-(37) bilden ein lineares Gleichungssystem mit einer unteren Dreiecksmatrix als Systemmatrix, wie in Abschnitt 3.2 (S. 11f) beschrieben. Die Größen $\triangle_{G,f_j}$ und $\triangle_{R,f_j}$ $(1 \le j \le k)$ entsprechen den Vektoren h bzw. z des dort beschriebenen Systems. Somit kann der Gesamtfehler $\triangle_{G,f_k}$ durch Rückwärtsrechnung bestimmt werden, da die Voraussetzungen von Korollar 3.1 (S. 12) erfüllt sind. Es gilt:

$$\triangle_{G,f} = \triangle_{G,f_k} = \sum_{j=1}^{k} d_j\ \triangle_{R,f_j}\ , \tag{38}$$

wobei die Größen d_j nach Korollar 3.1 durch die folgende Akkumulation, beginnend mit $d_k := 1$ und $d_j := 0$ $(1 \le j \le k-1)$, berechnet werden:

$$\begin{aligned}
d_l &:= d_l - d_j \text{ , falls } f_j = -f_l && (39)\\
d_l &:= d_l + d_j \text{ , } d_r := d_r + d_j \text{ , falls } f_j = f_l + f_r && (40)\\
d_l &:= d_l + d_j \text{ , } d_r := d_r - d_j \text{ , falls } f_j = f_l - f_r && (41)\\
d_l &:= d_l + \widetilde{f_r}\ d_j \text{ , } d_r := d_r + f_l\ d_j \text{ , falls } f_j = f_l \cdot f_r && (42)\\
d_l &:= d_l + d_j/\widetilde{f_r} \text{ , } d_r := d_r - f_j\ d_j/\widetilde{f_r} \text{ , falls } f_j = f_l/f_r && (43)\\
d_l &:= d_l + g'(\xi)\ d_j \text{ , falls } f_j = g(f_l),\ \xi \in f_l \sqcup \widetilde{f}_l && (44)
\end{aligned}$$

Bemerkung 4.2 *Beim Auftreten des unären Operators − wird auch auf der Maschine nicht gerundet. Es wird lediglich das Vorzeichenflag neu gesetzt. Der entsprechende Summand kann in (38) weggelassen werden.*

Bemerkung 4.3 *Die Größen d_j in (38) bestimmen die Größe des Gesamtfehlers. Sie werden als Verstärkungsfaktoren bezeichnet.*

Da die reellen Werte f_j $(1 \le j \le k)$ und ξ nicht bekannt sind, kann durch Einsatz der Intervallrechnung (Intervallauswertung von (39)-(44) und der Funktion f) eine obere Schranke für den Rundungsfehler hergeleitet werden. Dazu sei $F_j, D_j \in IS$ mit

$f_j, \widetilde{f_j} \in F_j$ und $d_j \in D_j$ $(1 \leq j \leq k)$ auf der Maschine berechnet. Dann gilt aufgrund von (38) für den Betrag des Gesamtfehlers die Ungleichung:

$$|\triangle_{G,f}| \leq \sum_{j=1}^{k} |D_j| \; |\triangle_{R,f_j}|. \tag{45}$$

Handelt es sich bei f_j um eine elementare Operation, so erhält man für den Betrag des Rundungsfehlers durch Anwendung von (2) (S. 5) die Abschätzung:

$$|\triangle_{R,f_j}| \leq \begin{cases} \varepsilon \; |F_j| & , \quad F_j \subseteq N \\ \texttt{MinReal} & , \quad F_j \subseteq U \\ \varepsilon \; |F_j| + \texttt{MinReal} & , \quad \text{sonst.} \end{cases} \tag{46}$$

Führt f_j eine elementare Funktion aus, so gilt wegen der Annahme (3) (S. 5) die folgende Ungleichung:

$$|\triangle_{R,f_j}| \leq \begin{cases} \varepsilon(f_j) \; |F_j| & , \quad F_j \subseteq N \\ u(f_j) & , \quad F_j \subseteq U \\ \varepsilon(f_j) \; |F_j| + u(f_j) & , \quad \text{sonst.} \end{cases} \tag{47}$$

Mit den Ungleichungen (45), (46) und (47) läßt sich eine obere Schranke für den Betrag des entstehenden Gesamtfehlers berechnen.

Wenn in der zugrundeliegenden Funktion binäre Operationen aus $\mathcal{OP}$ auftreten, so ist es möglich, zusätzliche Aussagen über die Größe der bei diesen Operationen auftretenden Rundungsfehler zu treffen. In [1] werden hinreichende Kriterien dafür angegeben, unter welchen Umständen diese Operationen rundungsfehlerfrei verlaufen. So erhält man zum Beispiel für die Addition zweier Gleitkommazahlen folgende Aussage:

Satz 4.2 *Sei die Anzahl der abschließenden Nullen der Mantisse einer Gleitkommazahl z durch die Funktion $z_{trail}(z)$ berechenbar. Wenn mindestens eine der Bedingungen*

1. $-2 \leq \frac{x}{y} \leq -\frac{1}{2}$
2. $e(x+y) \leq \min(e(x) + z_{trail}(x),\ e(y) + z_{trail}(y))$
3. $x = 0 \vee y = 0$

erfüllt ist und bei der Berechnung kein Unterlauf entsteht, so verläuft die Addition zweier Gleitkommazahlen $x, y \in S$ auf der Maschine rundungsfehlerfrei, d.h. es gilt $|(x \boxplus y) - (x + y)| = 0$.

Beweis: Siehe [1]. □

Ebenso können Aussagen für die verbleibenden elementaren Operationen formuliert werden (vgl. dazu [1]).

4.4 Implementierung

Durch die zur Klasse `rivall` (siehe [6]) gehörenden `friend`-Funktionen

```
real reverse_error(TAPE& wt, int y_c);
void reverse_error_j(TAPE& wt, int& j,
                     intvector& rnd_free_op,
                     rvector& delta);

real reverse_error(TAPE& wt, int y_c, rvector& stoerung);
void reverse_error_j(TAPE& wt, int& j,
                     intvector& rnd_free_op,
                     rvector& delta,
                     rvector& stoerung);
```

wurde (45) für alle Operationen aus $\mathcal{OP}$ in `reverse.cpp` implementiert. Um eine garantierte Oberschranke des Betrags des absoluten Gesamtfehlers auf der Maschine zu erhalten, müssen gerichtete Rundungen verwendet werden. Nachdem mit der Funktion

```
void forward_sweep(TAPE& working_tape,
                   rivall (*f)(TAPE& working_tape, rivall_vec& x),
                   ivector& d_vec,
                   interval& f_val)
```

alle Funktionswerteinschließungen F_j $(1 \leq j \leq k)$ berechnet wurden, kann anschließend der Betrag des absoluten Gesamtfehlers mit einer der beiden Funktionen `reverse_error(...)` bestimmt werden. Bei der zweiten dieser Funktionen ist es möglich, Eingangsstörungen direkt zu übergeben. Die Abschätzung des Rundungsfehlers der entsprechenden Operation erfolgt mittels (46)[6] unter Beachtung möglicher rundungsfehlerfrei verlaufender Operationen. Ob eine der binären Operationen aus $\widetilde{\mathcal{OP}}$ exakt verläuft, wird durch die Funktionen

```
int check_rnd_free(real& delta_l, real& delta_r,
                   interval& f_l,interval& f_r,
                   int _op );
```

und

```
int check_rnd_free_U(real& delta_l, real& delta_r,
                   interval& f_l,interval& f_r,
                   int _op );
```

festgestellt. Diese sind maschinenabhängig und entsprechenden den jeweiligen Funktionen aus der Implementierung des Kalküls in [1, 2].

Sind Eingangsgrößen oder Konstanten exakt darstellbar, so benutzt man, nachdem ihre Einschließungswerte den Variablen vom Typ `rivall` übergeben worden sind, die `friend`-Funktionen

[6] Falls auch elementare Funktionen in der Implementierung berücksichtigt werden sollen, so kann dies durch geeignete Fallunterscheidungen bei der Berechnung des Rundungsfehlers geschehen. Dann muß (47) benutzt werden.

```
void exactVar(rivall& s)
void exactConst(rivall& s).
```

Dabei wird an der Stelle, an der die Variable im Tape erscheint, eine Marke gesetzt, welche der reverse-sweep dazu benutzt, diese Variable als fehlerfreie Größe zu identifizieren. Analog ist vorzugehen, falls bekannt ist, daß eine eingehende Größe oder Konstante nach der nächstgelegenen Maschinenzahl gerundet wird. Es sind dann die beiden `friend`-Funktionen

```
void rnd_n_Var(rivall& s)
void rnd_n_Const(rivall& s)
```

zu verwenden.

4.5 Vergleich

An einigen einfachen Testfunktionen wird nun das Verhalten des durch obige Implementierung realisierten Algorithmus (Rückwärtsmethode (R)) mit der Implementierung des Fehlerkalküls von Hofschuster und Krämer (im folgenden immer als Vorwärtsmethode (V) bezeichnet) verglichen.

Beispiel 4.1 (Formelauswertung) *Gegeben sei die Formel*

$$y = a^2 + b^2 - c^2 + d^2 - e^2 \tag{48}$$

mit den Argumenten $a = 320000.0$, $b = 19.0/32768.0$, $c = 39.0/65536.0$, $d = 240000.0$ *und* $e = 400000.0$. *Diese sind so gewählt, daß bei Hintereinanderausführung der Operationen gemäß (48) starke Auslöschung auftritt. Man kann (48) auf verschiedene Weise auswerten, wie z.B.*

$$\begin{aligned} y_1 &= (((a^2 + b^2) - c^2) + d^2) - e^2 \\ y_2 &= ((a^2 + b^2) - c^2) + (d^2 - e^2) \\ y_3 &= a^2 + ((b^2 - c^2) + (d^2 - e^2)) \\ y_4 &= (b^2 - c^2) + (d^2 - e^2 + a^2). \end{aligned}$$

Je nach Auswertungsreihenfolge ergeben sich verschiedene Fehlerschranken für den Betrag des absoluten Fehlers, aus denen man erkennen kann, welche Reihenfolge optimal ist.
Bei der intervallmäßigen Auswertung treten bei Anwendung von y_1, y_2 *und* y_3 *starke Überschätzungen auf:*

$$\begin{aligned} y_1 &\in [-3.051758E-005, 3.051758E-005] \\ y_2 &\in [-1.525879E-005, 1.525879E-005] \\ y_3 &\in [-1.525879E-005, 0.000000E+000] \\ y_4 &\in [-1.792796E-008, -1.792795E-008] \end{aligned}$$

Wie Tabelle 1 zeigt, ist der absolute Fehler auch dort am kleinsten, wo die Intervallüberschätzung am geringsten ausfällt. Ebenso ist zu erkennen, daß die Fehlerschranken, die durch Rückwärtsrechnung bestimmt wurden, den durch Vorwärtsrechnung ermittelten entsprechen.

Auswertung	abs. Fehler (V)	abs. Fehler (R)
y_1	8.100187E-005	8.100187E-005
y_2	4.547474E-005	4.547474E-005
y_3	2.273737E-005	2.273737E-005
y_4	4.598607E-022	4.598607E-022

Tabelle 1: Absolute Fehler bei verschiedener Auswertungsreihenfolge in Bsp. 4.1

Das nächste Beispiel beschäftigt sich mit der Auswertung von Polynomen. Dabei ist zu erwarten, daß sich für verschiedene Auswertungsarten unterschiedliche Fehlerschranken ergeben, da zum einen die Anzahl der auszuführenden Operationen verschieden ist und zum anderen die Auswertungsreihenfolge der Operationen differiert.

Beispiel 4.2 (Polynomauswertung) *Es wird die Auswertung des in Abbildung 2 (S. 19) dargestellten Legendre-Polynoms* $L_{13}(x)$ *vom Grad* 13 *auf dem Intervall* $[-1, 1]$ *betrachtet. Die Koeffizienten lassen sich rekursiv berechnen (vgl. dazu [32]). Für* L_{13}

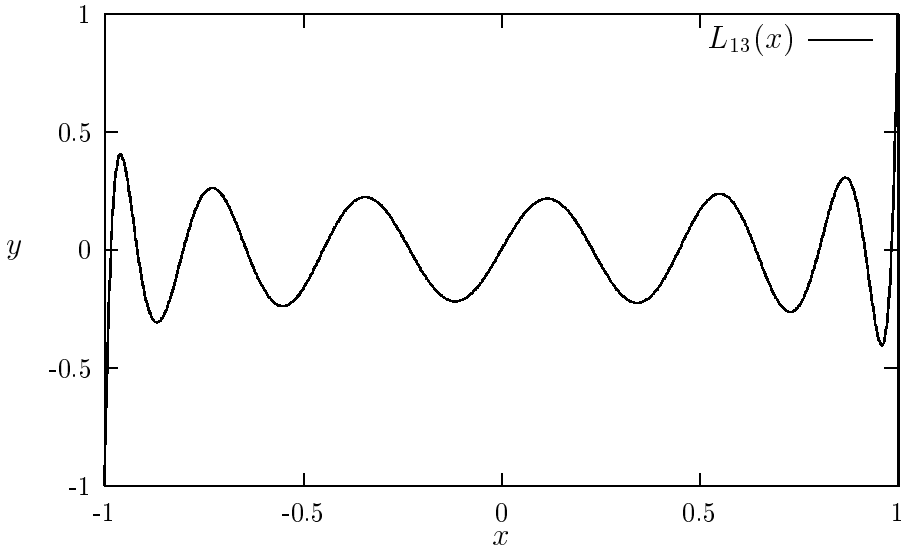


Abbildung 2: Legendre-Polynom $L_{13}(x)$ auf dem Bereich $[-1, 1]$

ergeben sich diese zu

$$a_1 = 3003/1024 \qquad a_3 = -90090/1024 \qquad a_5 = 765765/1024$$
$$a_7 = -2771340/1024 \qquad a_9 = 4849845/1024 \qquad a_{11} = -4056234/1024$$
$$a_{13} = 1300075/1024$$

und $a_i = 0$ *für* $i = 0, 2, \ldots, 12$. *Gesucht sei wieder eine gleichmäßig gültige Fehlerschranke auf dem vorgegebenen Intervall* $[-1, 1]$.

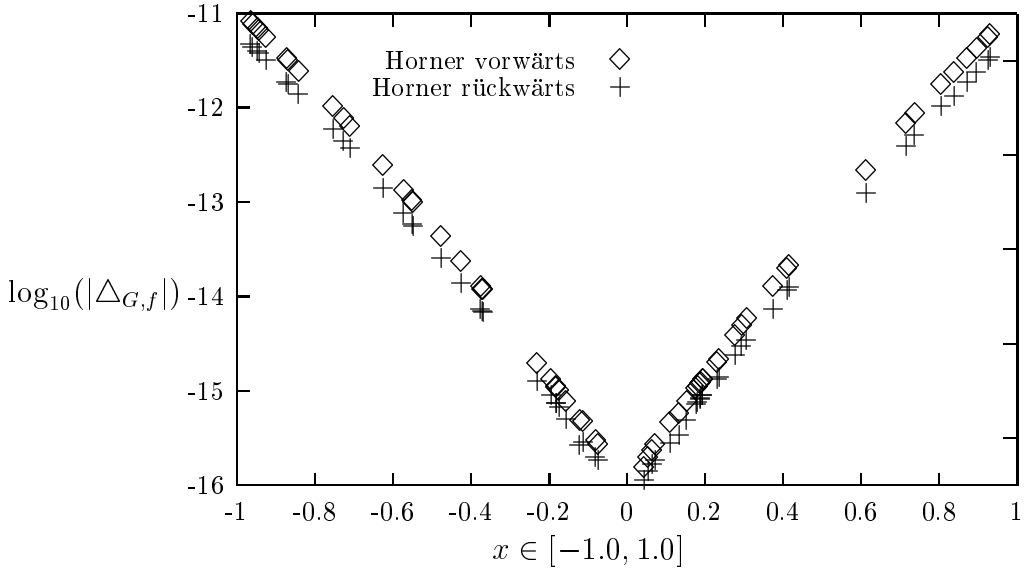


Abbildung 3: Absolute Fehler bei Auswertung durch das Hornerschema in Bsp. 4.2 mit zufällig erzeugten Argumenten aus $[-1, 1]$

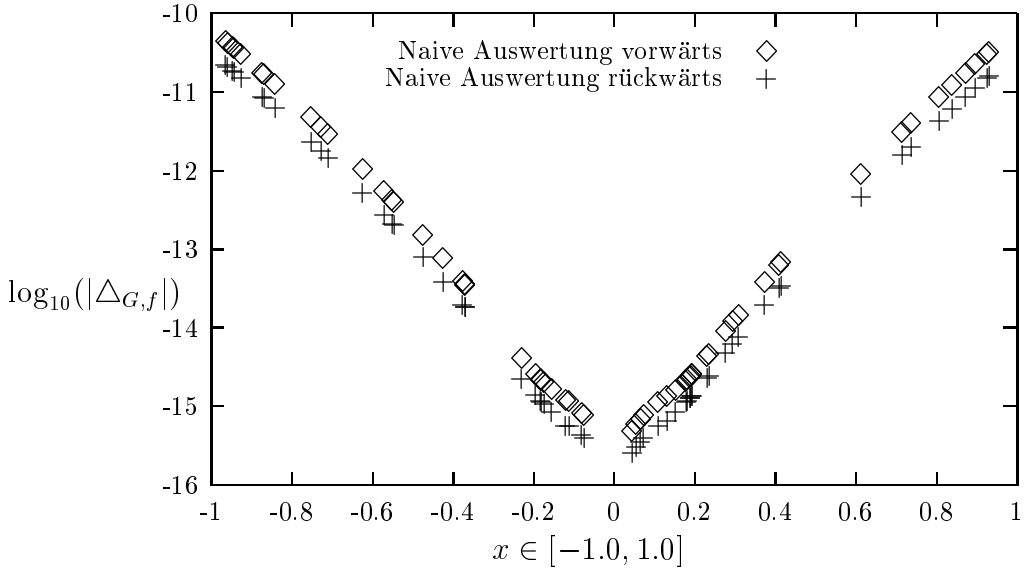


Abbildung 4: Absolute Fehler bei naiver Auswertung in Bsp. 4.2 mit zufällig erzeugten Argumenten aus $[-1, 1]$

Die Auswertung erfolgte sowohl durch naive Auswertung des Polynoms und Auswertung durch das Hornerschema im Vorwärts- als auch im Rückwärtsmodus. Mit zufällig erzeugten Punktintervallen wurden zunächst Unterschranken für die von beiden Auswertungsarten berechneten Fehlerschranken bestimmt. In den Abbildungen 3 und 4 (S. 20) ist der prinzipielle Verlauf der Fehlerfunktion zu sehen. Es ist zu erkennen, daß die Fehler stark anwachsen, sobald der Betrag der Argumente gegen Eins geht. Das Alternieren der Koeffizientenvorzeichen macht sich hier verstärkt bemerkbar.
Für die Berechnung des Gesamtfehlers wurde das zugrundeliegende Intervall in 5 gleichgroße Teilbereiche $B_1, \ldots, B_5$ *mit*

$$B_1 := [-1.0, -0.6] \quad B_2 := [-0.6, -0.2]$$
$$B_3 := [-0.2, 0.2] \quad B_4 := [0.2, 0.6]$$
$$B_5 := [0.6, 1.0]$$

unterteilt[7]*, die dann wiederum in n Intervalle aufgeteilt wurden. Aus den Abbildungen 3 und 4 (S. 20) sowie den Ergebnistabellen 2 bis 5 (S. 22f) ist zu erkennen, daß die Auswertung durch das Hornerschema sowohl für* $n = 1$ *als auch für* $n = 100$ *der naiven Polynomauswertung überlegen ist. Mit der Rückwärtsmethode werden in diesem Beispiel bessere absolute Fehlerschranken erzielt als mit der Vorwärtsmethode.*

Daß die durch Rückwärtsrechnung bestimmten Fehlerschranken nicht immer besser sein müssen, zeigt das nächste Beispiel.

Beispiel 4.3 (Fehlerabschätzung arctan-Funktion) *Eine Möglichkeit zur Berechnung der* arctan*-Funktion auf der Maschine stellt der Algorithmus* `q_atan` *aus [13] (vgl. auch S. 26) dar. Der Verlauf des relativen Fehlers für diesen Algorithmus bei sinnvoller Unterteilung des Argumentbereichs (*$|x| >$ `q_atnt`*, vgl. S. 26) ist in Abbildung 5 dargestellt*[8]*. Es fällt auf, daß beide Fehlerkurven prinzipiell gleich verlaufen, die Rückwärtsmethode in den Bereichen* B_2 *bis* B_6 *jedoch etwas größere Fehlerschranken liefert. Eine mögliche Ursache dafür ist die hier verwendete Methode zur Berücksichtigung des Approximationsfehlers (vgl. Abb. 6, S. 24), welcher bei der Polynomapproximation in Schritt 3b des Algorithmus* `q_atan` *auftritt. Dieser Approximationsfehler kann nicht wie bei der Vorwärtsmethode auf die bis dahin berechnete Fehlerschranke addiert werden. Vielmehr muß er als Störung in der weiteren Rechnung angesehen werden. Genauer: Es wird der Gesamtfehler bis einschließlich der Polynomapproximation berechnet, dann wird die Berechnung abgebrochen. Anschließend gehen Approximationsfehler und der bis dahin berechnete Gesamtfehler als Störung in eine erneute Rückwärtsrechnung ein*[9]*. Es muß beachtet werden, daß der dabei durchgeführte forward-sweep mit dem um die Störung aufgeblähten Funktionswert gestartet werden muß, um korrekte Einschließungen zu erhalten. Folglich können sich in den weiteren Berechnungen die Einschließungen* F_j *der in der Codeliste vorkommenden Funktionswerte* f_j *vergrößern und sich deshalb die Fehlerschranken verschlechtern. Durch weitere Unterteilung konnte dieser Effekt teilweise beseitigt werden.*

[7] Auf der Maschine müssen Einschließungen von B_1 bis B_5 benutzt werden, da die Zahlen -0.6, -0.2, 0.2 und 0.6 nicht exakt darstellbar sind.

[8] Die dabei zugrundeliegenden Werte sind im Abschnitt 6.2 (S. 27f) ausführlich angegeben.

[9] Dazu wird die im letzten Abschnitt vorgestellte Funktion `real reverse_error(TAPE& wt, int y_c, rvector& stoerung)` benutzt.

Bereich	abs. Fehler (V)	abs. Fehler (R)
B_1	2.334474E-011	1.768227E-011
B_2	3.812549E-013	2.846412E-013
B_3	3.466742E-015	2.830968E-015
B_4	3.812549E-013	2.846412E-013
B_5	2.334474E-011	1.768227E-011

Tabelle 2: Absolute Fehler bei Auswertung durch das Hornerschema in Bsp. 4.2, $n = 1$

Bereich	abs. Fehler (V)	abs. Fehler (R)
B_1	6.720361E-011	5.296701E-011
B_2	9.064231E-013	7.393192E-013
B_3	6.142146E-015	6.011912E-015
B_4	9.064231E-013	7.393192E-013
B_5	6.720361E-011	5.296701E-011

Tabelle 3: Absolute Fehler bei naiver Auswertung in Bsp. 4.2, $n = 1$

Bereich	abs. Fehler (V)	abs. Fehler (R)
B_1	1.124859E-011	6.784536E-012
B_2	1.832245E-013	1.111242E-013
B_3	1.418401E-015	1.005887E-015
B_4	1.832245E-013	1.111242E-013
B_5	1.124859E-011	6.784536E-012

Tabelle 4: Absolute Fehler bei Auswertung durch das Hornerschema in Bsp. 4.2, $n = 100$

Bereich	abs. Fehler (V)	abs. Fehler (R)
B_1	6.160155E-011	3.263988E-011
B_2	7.509274E-013	4.020721E-013
B_3	2.766735E-015	1.571294E-015
B_4	7.509274E-013	4.020721E-013
B_5	6.160155E-011	3.263988E-011

Tabelle 5: Absolute Fehler bei naiver Auswertung in Bsp. 4.2, $n = 100$

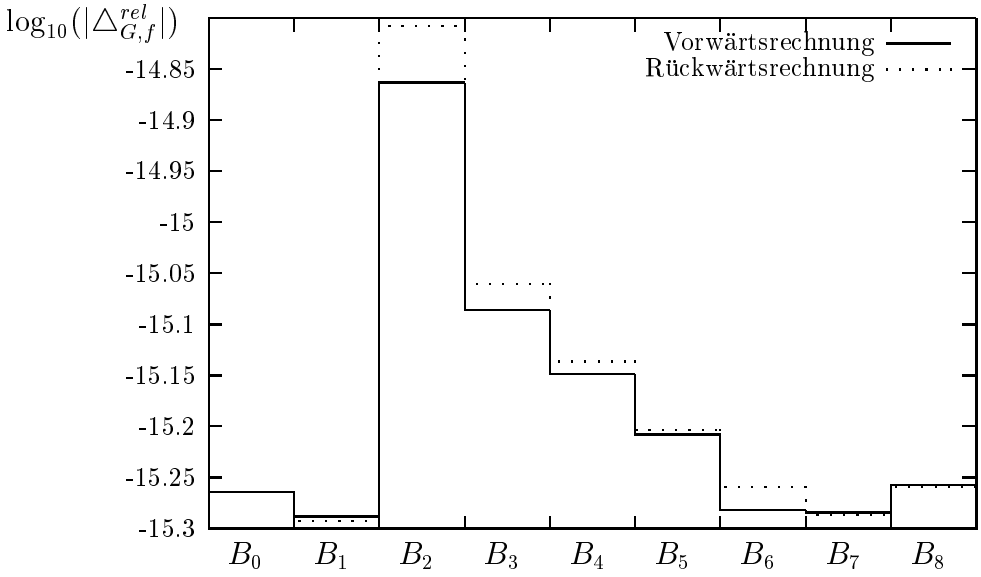


Abbildung 5: Relative Fehler bei Auswertung durch Vorwärts- und Rückwärtsrechnung von `q_atan`

Um eine möglichst kleine Gesamtfehlerschranke für den Algorithmus `q_atan` *zu erhalten, können die Ergebnisse beider Berechnungsmethoden benutzt werden. Man bildet das Maximum der Minima der durch beide Verfahren bestimmten Fehlerschranken auf allen Teilbereichen. Dieser Wert ist dann* $1.555012E-015$.

Bemerkung 4.4 (Fehlerabschätzung der exp-Funktion) *Für ein Tabellenverfahren, welches die Exponentialfunktion implementiert, wurde ebenfalls versucht, eine Fehlerschranke für den relativen Fehler zu berechnen. Dabei brach die Rückwärtsrechnung bei großen Argumenten ab. Die Ursache liegt darin, daß Einschließungen der Verstärkungsfaktoren, die im Bereich von* `MaxReal` *liegen, in Operationen eingehen, deren Ergebnis einen Überlauf liefert. Für die Berechnung einer Gesamtfehlerschranke ist hier die Rückwärtsmethode nicht geeignet.*

Ohne explizit die Laufzeiten zur Berechnung der Fehlerschranken in den Beispielen aufzulisten, ist zu bemerken, daß sich die Rückwärtsrechnung hinsichtlich des Laufzeitverhaltens schlechter verhält als die Vorwärtsrechnung. Größenordnungsmäßig

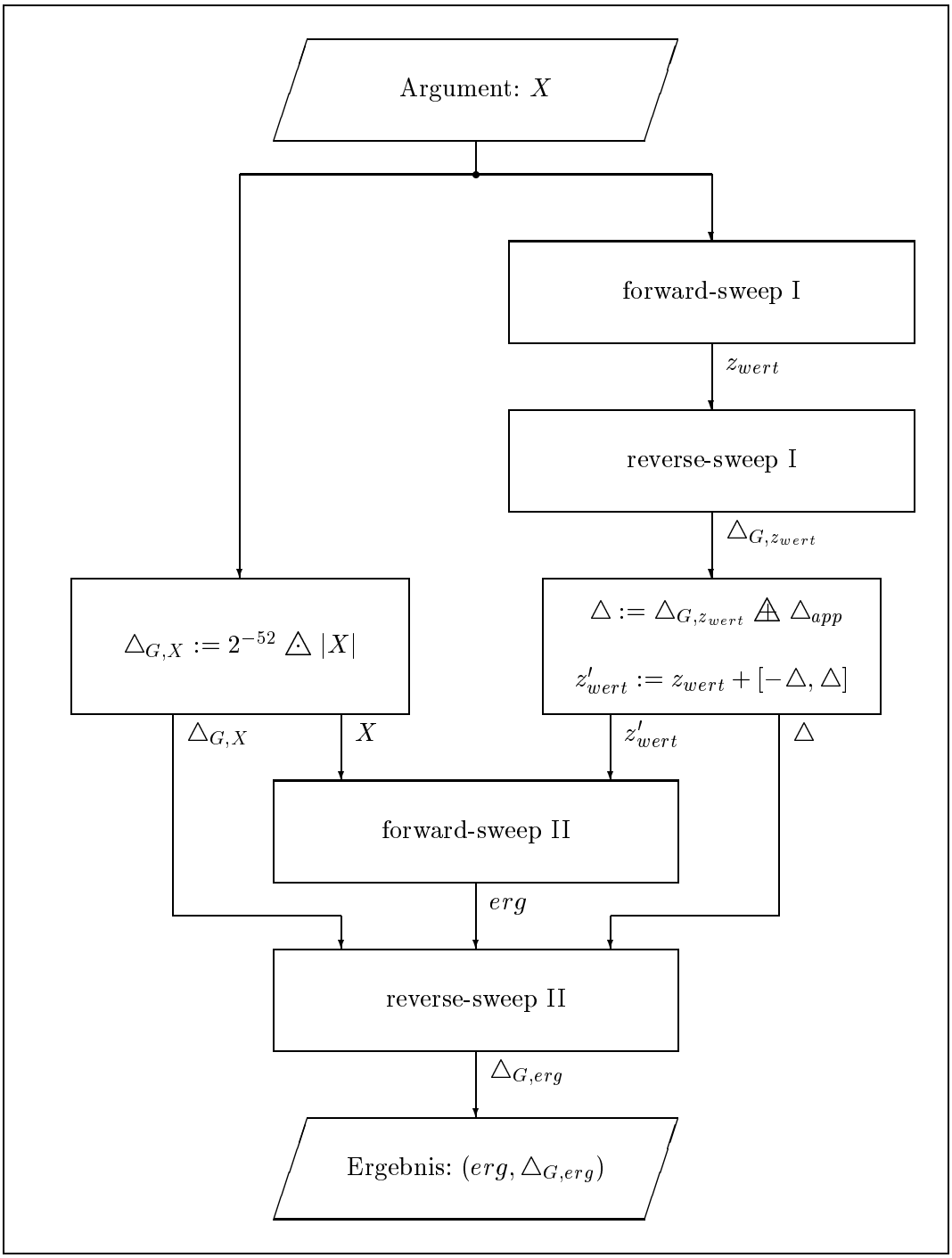


Abbildung 6: Prinzipielle Vorgehensweise bei Einbeziehung des Approximationsfehlers

entspricht die Anzahl der Operationen zur Berechnung der Verstärkungsfaktoren in (45) der Anzahl der Operationen zur Berechnung der absoluten Fehler der einzelnen Teilfunktionen der Codeliste in der Vorwärtsrechnung. Jedoch müssen in der Rückwärtsrechnung ebenfalls alle absoluten Fehler der einzelnen Teilfunktionen der Codeliste berechnet werden, um überprüfen zu können, ob rundungsfehlerfreie Operationen vorliegen. Das erhöht den Aufwand erheblich.

Zwar wurden in der Implementierung [2] noch keine Verzweigungen mit einbezogen, aber man kann auch hier eine Aussage über das Verhalten von Vorwärts- und Rückwärtsrechnung treffen. Treten Verzweigungen im zugrundeliegenden Algorithmus auf, so kann das Maximum der berechneten Fehlerschranken beider Zweige als Fehlerschranke dienen. Erweitert man $\mathcal{F}$ um die in [16, 17] (siehe auch [6, 28], Kap. 7) vorgestellte Verzweigungsfunktion χ, so werden durch (45) die Fehlerschranken beider Zweige addiert. Aufgrund der Ungleichung

$$\max\{x, y\} \leq x + y \quad x, y \in I\!R \text{ mit } x, y \geq 0$$

ist die mittels Vorwärtsrechnung bestimmte Fehlerschranke der durch Rückwärtsrechnung ermittelten Schranke im allgemeinen überlegen.

Abschließend ist zu bemerken, daß sich die Anwendung der Rückwärtsrechnung im Vergleich zur Vorwärtsrechnung mühsamer gestaltet, das Umschreiben der zu testenden Algorithmen ist *erheblich aufwendiger*. Bei der Vorwärtsrechnung müssen größtenteils nur Datentypen ausgetauscht werden.

5 Schlussbemerkung

Es können gleichmäßige Fehlerschranken numerischer Verfahren auf gegebenen Argumentbereichen durch Vorwärts- als auch durch Rückwärtsrechnung automatisch ermittelt werden. Solche Fehlerschranken geben Auskunft über die Qualität des betrachteten Verfahrens hinsichtlich der Verstärkung von Rundungsfehlern. Außerdem ist mit ihnen eine Sensitivitätsanalyse möglich, bei der die Auswirkung von Störungen in den Eingangsparametern auf die Ergebniswerte untersucht werden kann. Auch hier wird die Faktorisierbarkeit der Funktionen vorausgesetzt, für welche die Fehlerschranken ermittelt werden sollen. Durch Anwendung einer speziellen Verzweigungsfunktion ([6], Kap. 7) können aber zusätzlich Bedingungsanweisungen einbezogen werden. Dabei liefert die Vorwärtsmethode im allgemeinen bessere Fehlerschranken als die Rückwärtsrechnung. In der vorliegenden Arbeit wurden Fehlerschranken für Funktionen auf dem Rechner bestimmt, welche aus elementaren Operationen $\{-, +, -, \cdot, /\}$ zusammengesetzt sind. Treten in diesen Funktionen Approximationen auf, wie sie zum Beispiel in Verfahren vorkommen, welche Gleitkommanäherungen von Standardfunktionen auf der Maschine bestimmen, so konnte festgestellt werden, daß die Vorwärtsmethode kleinere Fehlerschranken ermittelt als die Rückwärtsrechnung. Ebenfalls zeigt die Vorwärtsmethode in den gerechneten Beispielen ein besseres Laufzeitverhalten, wenn man für alle Operationen aus $\{+, -, \cdot, /\}$ überprüft, ob diese möglicherweise rundungsfehlerfrei verlaufen.

Da in den XSC-Sprachen zusätzlich das hochgenaue Skalarprodukt als fünfte Grundoperation zur Verfügung steht, sollte in weiterführenden Arbeiten untersucht werden, wie man gleichmäßige Fehlerschranken erhält, wenn die Menge der elementaren Operationen um das Skalarprodukt erweitert wird.
Die vorgestellte Methodik wurde in [6] unter Verwendung der C++ Klassenbibliothek C-XSC [19, 11] in Softwarewerkzeuge umgesetzt. Quellcodeauszüge finden sich im Anhang A 3 in [6].

6 Anhang: Fehlerabschätzung der arctan-Funktion

6.1 Algorithmus `q_atan` aus Beispiel 4.3

Für die Berechnung der Funktion arctan(.) auf der Maschine kann der Algorithmus `q_atan` aus [13] benutzt werden. Hier findet man auch eine ausführliche Erklärung dieses Verfahrens und eine Beschreibung aller verwendeter Konstanten. Der Algorithmus `q_atan` ist in [13] durch folgende vier Hauptschritte beschrieben[10]:

1. Abprüfen von Spezialfällen und deren Behandlung
   (a) x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
   (b) $|x| \leq$ `q_atnt` $= 1.807032\ldots \cdot 10^{-8} \Longrightarrow res := x$
2. Initialisierung und Argumentreduktion
   (a) Falls $x < 0 : y := -x$, $neg :=$ `true`
   sonst: $y := x$, $neg :=$ `false`
   (b) Falls $y < 8 : vzi := 1$, $y_m := 0$
   sonst: $vzi := -1$, $y_m := \frac{\pi}{2}\ |_{IEEE}$, $y := 1 \boxslash y$
   (c) Bestimme $i := \max_{k=1,2,\ldots,7}\{k | b_k \leq y\}$
   (d) Berechne $y := (y \boxminus c_i) \boxslash (1.0 \boxplus y \circledast c_i)$
3. Approximation: $y + y \cdot (\sum_{i=0}^{5} d_i \cdot y^2)$, d.h. auf dem Rechner
   (a) $y_{sq} := y \circledast y$
   (b) $z_{wert} := ((((d_5 \circledast y_{sq} \boxplus d_4) \circledast y_{sq} \boxplus d_3) \circledast y_{sq} \boxplus d_2) \circledast y_{sq} \boxplus d_1) \circledast y_{sq} \boxplus d_0)$
   (c) $res := y \circledast z_{wert} \boxplus y$
4. Ergebnisanpassung
   (a) $res := res \boxplus a_i$
   (b) $res := vzi \cdot res \boxplus y_m$
   (c) Falls $neg =$ `true` : $res := -res$

Nun gilt: $res \approx \arctan(x)$.

[10] Die Bezeichnungen wurden von dort übernommen.

6.2 Ergebnisse der Fehlerabschätzung

Die in Abbildung 5 gezeigten Fehlerschranken ergeben sich aus den folgenden Ergebnissen. Die Größe n gibt dabei die Anzahl nochmaliger Unterteilungen der Bereiche $B_0 - B_8$ an. Bei Anwendung der Vorwärtsrechnung erhält man:

```
B0 [1.807031E-008,1.000000E-007] n: 40    4.944260E-016
B0 [9.999999E-008,1.000000E-006] n: 40    5.440093E-016
B0 [9.999999E-007,1.000001E-005] n: 40    5.440093E-016
B0 [1.000000E-005,1.000001E-004] n: 40    5.440093E-016
B0 [1.000000E-004,1.000001E-003] n: 40    5.440093E-016
B0 [1.000000E-003,1.000001E-002] n: 40    5.440114E-016
B0 [1.000000E-002,1.000001E-001] n: 40    5.442172E-016
B1 [1.000000E-001,1.250000E-001] n: 2     5.153864E-016
B2 [1.250000E-001,3.913935E-001] n: 10000 1.369861E-015
B3 [3.913934E-001,7.165921E-001] n: 1000  8.212020E-016
B4 [7.165920E-001,1.186492E+000] n: 10    7.105762E-016
B5 [1.186491E+000,2.061722E+000] n: 10    6.197640E-016
B6 [2.061721E+000,4.860929E+000] n: 10    5.223889E-016
B7 [4.860928E+000,8.000000E+000] n: 1     5.197107E-016
B8 [8.000000E+000,1.000000E+001] n: 10    4.221772E-016

B8 [1.000000E+i,1.000000E+(i+1)] n: je 10
                            mit i=1(1)307:  gesamt   5.528541E-016

B8 [1.000000E+308,1.797694E+308] n: 1000  3.330669E-016

Gesamtfehler: 1.369861E-015
```

Die Rückwärtsrechnung ergibt bei gleicher Unterteilung:

```
B0 [1.807031E-008,1.000000E-007] n: 40    4.944260E-016
B0 [9.999999E-008,1.000000E-006] n: 40    5.440093E-016
B0 [9.999999E-007,1.000001E-005] n: 40    5.440093E-016
B0 [1.000000E-005,1.000001E-004] n: 40    5.440093E-016
B0 [1.000000E-004,1.000001E-003] n: 40    5.440093E-016
B0 [1.000000E-003,1.000001E-002] n: 40    5.440107E-016
B0 [1.000000E-002,1.000001E-001] n: 40    5.441492E-016
B1 [1.000000E-001,1.250000E-001] n: 2     5.097375E-016
B2 [1.250000E-001,3.913935E-001] n: 10000 1.555012E-015
B3 [3.913934E-001,7.165921E-001] n: 1000  8.702787E-016
B4 [7.165920E-001,1.186492E+000] n: 10    7.300869E-016
B5 [1.186491E+000,2.061722E+000] n: 10    6.256731E-016
B6 [2.061721E+000,4.860929E+000] n: 10    5.504527E-016
B7 [4.860928E+000,8.000000E+000] n: 1     5.175578E-016
B8 [8.000000E+000,1.000000E+001] n: 10    4.204228E-016

B8 [1.000000E+i,1.000000E+(i+1)] n: je 10
                            mit i=1(1)307:  gesamt   5.510245E-016

B8 [1.000000E+308,1.797694E+308] n: 1000  3.330669E-016

Gesamtfehler: 1.555012E-015
```

und bei Verfeinerung der Teilbereiche $B_4 - B_6$:

```
B4 [7.165920E-001,1.186492E+000] n: 100  7.057976E-016
B5 [1.186491E+000,2.061722E+000] n: 100  6.046695E-016
B6 [2.061721E+000,4.860929E+000] n: 100  5.255126E-016
```

Literatur

[1] Bantle, A., Krämer, W.: *Ein Kalkül für verläßliche absolute und relative Fehlerabschätzungen* Preprint 98/5 des IWRMM, Universität Karlsruhe, 1998

[2] Bantle, A., Krämer, W.: *Implementierung, Handhabung und Beispielanwendungen eines verlässlichen Vorwärtsfehlerkalküls* Forschungsschwerpunkt Computerarithmetik, Intervallrechnung und Numerische Algorithmen mit Ergebnisverifikation, Bericht 2/1999, Karlsruhe, 1999

[3] Beck, T.: *Automatisches Differenzieren von Algorithmen* Dissertation, Technische Universität München, 1991

[4] Berz, M., Bischof, C., Corliss, G. and Griewank, A. (eds.): *Computational Differentiation: Techniques, Applications, and Tools* SIAM, Philadelphia, Penn., 1996

[5] Braune, K., Krämer, W.: *High-Accuracy Standard Functions for Intervals* in: [30], pp. 341-347

[6] Bräuer, M.: *Berechnungsmethoden für Ableitungen und Steigungen und deren Realisierung in C-XSC* Diplomarbeit, Universität Karlsruhe, 1999.

[7] Fischer, H.: *Special Problems in Automatic Differentiation* in: [9], pp. 43-50

[8] Fischer, H.-C.: *Schnelle Automatische Differentiation, Einschließungsmethoden und Anwendungen* Dissertation, Institut für Angewandte Mathematik, Karlsruhe, 1990

[9] Griewank, A., Corliss, G. F. (eds.): *Automatic Differentiation of Algorithms: Theory, Implementation, and Application* SIAM, Philadelphia, Penn., 1991

[10] Griewank, A., Juedes, D. and Utke, Jean: *ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C++* ACM Trans. Math. Software, 22 (1996), pp. 131-167

[11] Hammer, R., Hocks, M., Kulisch, U. und Ratz, D.: *C++ Toolbox for Verified Computing I - Basic Numerical Problems* Springer Verlag, Berlin, 1995

[12] Hofschuster, W. und Krämer, W.: *Ein rechnergestützter Fehlerkalkül mit Anwendung auf ein genaues Tabellenverfahren* Preprint Nr. 96/5, Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung, Universität Karlsruhe, 1996

[13] Hofschuster, W. und Krämer, W.: *FI_LIB, eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-double-Format* Preprint Nr. 98/7, Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung, Universität Karlsruhe, 1998

[14] Iri, M.: *Simultaneous Computation of Functions, Partial Derivatives and Estimates of Rounding Errors* Japan Journal of Applied Mathematics, 1 (1994), pp. 223-252

[15] Iri, M., Tsuchiya, T. and Hoshi, M.: *Automatic computation of partial derivatives and rounding error estimates with application to large-scale systems of nonlinear equations* Journal of Computational and Applied Mathematics 24, 1988, pp. 365-392

[16] Kearfott, R.: *Interval Extensions of Non-Smooth Functions for Global Optimization and Nonlinear System Solvers* Computing 57 (1996), pp. 149-162

[17] Kearfott, R.: *Automatic Differentiation of Conditional Branches in an Operator Overloading Context* in: [7], pp. 75-81

[18] Kedem, G.: *Automatic Differentiation of Computer Programs* ACM Trans. Math. Software, 6 (1980), pp. 150-165

[19] Klatte, R., Kulisch, U., Lawo, C., Rauch, M., Wiethoff, A.: *C-XSC, A C++ Class Library for Extended Scientific Computing* Springer-Verlag Berlin, Heidelberg, New York 1993

[20] Krämer, W.: *Inverse Standard Functions for Real and Complex Point and Interval Arguments with Dynamic Accuracy* Computing, Suppl. 6, 1988, pp. 185–212

[21] Krämer, W.: *Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen* Forschungsschwerpunkt Computerarithmetik, Intervallrechnung und Numerische Algorithmen mit Ergebnisverifikation, Bericht 3/1996, Karlsruhe, 1996

[22] Krämer, W.: *Eine Fehlerfaktorarithmetik für zuverlässige a priori Fehlerabschätzungen* Forschungsschwerpunkt Computerarithmetik, Intervallrechnung und Numerische Algorithmen mit Ergebnisverifikation, Bericht 5/1997, 21 Seiten, Karlsruhe, 1997

[23] Krämer, W.: *Constructive Error Analysis* Journal of Universal Computer Science (JUCS), Vol. 4, No. 2, 1998, pp. 147-163

[24] Krämer, W.: *A priori Worst Case Error Bounds for Floating-Point Computations* IEEE Transactions on Computers, Vol. 47, No. 7, July 1998

[25] Kulisch, U.: *Grundlagen des Numerischen Rechnens* Bibliographisches Institut, Mannheim, Wien, Zürich 1976

[26] Kubota, K., Iri, M.: *Estimates of Roundings Errors with Fast Automatic Differentiation and Interval Analysis* Technical Report METR-88-15, Dep. of Mathematical Engineering and Information Physics, University of Tokyo, October 1988

[27] Neumann, K., Morlock M.: *Operations research* Carl Hanser Verlag, München, Wien, 1993

[28] Ratz, D.: *Automatic Slope Computation and its Application in Nonsmooth Global Optimization* Habilitationsschrift, Institut für Angewandte Mathematik, Karlsruhe, 1998

[29] Rump, S. M.: *Expansion and Estimation of the Range of Nonlinear Functions* Mathematics of Computation, Vol. 65, No. 216, 1996, pp. 1503-1512

[30] Ruschitzka, M. (editor): *Computer Systems: Performance and Simulation* Elsevier Science Publishers B.V. (North-Holland), IMACS, 1986

[31] Sander, P., Stucky, W., Herschel, R.: *Automaten, Sprachen, Berechenbarkeit* B.G.Teubner, Stuttgart, 1991

[32] Schwarz, H. R.: *Numerische Mathematik* B.G.Teubner, Stuttgart, 1993

[33] Stroustrup, B.: *Die C++ Programmiersprache* Addison-Wesley-Longman, Bonn, 1998