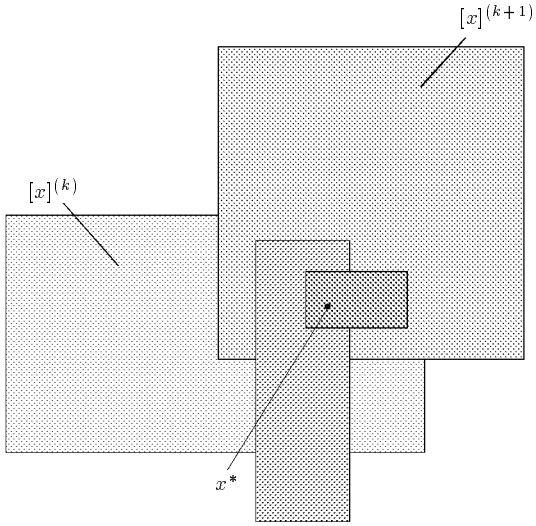
Institut für
Angewandte
Mathematik

Universität Karlsruhe (TH)
D-76128 Karlsruhe

Eine Fehlerfaktorarithmetik für zuverlässige a priori Fehlerabschätzungen

Walter Krämer

Forschungsschwerpunkt Computerarithmetik, Intervallrechnung und Numerische Algorithmen mit Ergebnisverifikation



Bericht 5/1997

Impressum

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

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

Walter Krämer
Institut für Wissenschaftliches Rechnen und
Mathematische Modellbildung (IWRMM)
Universität Karlsruhe (TH)
D-76128 Karlsruhe

E-Mail: Walter.Kraemer@math.uni-karlsruhe.de

Eine Fehlerfaktorarithmetik für zuverlässige a priori Fehlerabschätzungen

Walter Krämer

Inhaltsverzeichnis

Zusammenfassung

Eine Fehlerfaktorarithmetik: Es wird eine Arithmetik für Schranken von Fehlerfaktoren hergeleitet, welche mit Hilfe von Intervalloperationen und einer Programmiersprache mit Operator- und Funktionsüberladungsmöglichkeiten in leicht handhabbare automatische Softwarewerkzeuge umgesetzt werden kann. Der Einsatz dieser Werkzeuge gestattet es, worst-case Fehleranalysen von numerischen Berechnungsverfahren mit minimalem Aufwand durchzuführen. Das Einführen der sogenannten Fehlerfaktoren erlaubt die Angabe von Fehlerschranken in Einheiten der Rechengenauigkeit. Es können damit sowohl numerische Algorithmen in normaler Gleitkommaarithmetik als auch in verschiedenen Langzahlarithmetiken a priori untersucht werden.

Beispielhaft wird die neue Technik eingesetzt, um bei einem quadratisch konvergenten Iterationsverfahren zur Berechnung einer Näherung von π die Rundungsfehleranalyse a priori durchzuführen. Es läßt sich dann z. B. die Frage beantworten, wieviele Schutzziffern ausreichen, um eine Genauigkeit des Iterationsergebnisses auf eine bestimmte Stellenzahl garantieren zu können.

Abstract

An Error Factor Arithmetic: We derive an arithmetic which allows the computation of reliable bounds for error factors. The introduction of the so called error factors makes it possible to express error bounds in units of the machine precision epsilon (for multi-precision calculation epsilon denotes the precision of the arithmetic operations i. e. $0.5 \cdot b^{-k}$, k being the number of mantissa digits with repect to base b). Using interval operations in combination with a programming language providing an operator concept it is easy to develop software tools allowing the computation of a priory error bounds for the absolute/relative error of numerical algorithms using ordinary floating-point or multi-precision operations.

As an example the software tools are used to analyze a priory the error propagation of a quadratically convergent iteration scheme for the approximation of π. The results allow answering the question how many guard digits will be sufficient to get a certain number of correct digits in the final result.

1 Einleitung

Selbst wenn die einzelnen Grundoperationen (Addition, Multiplikation, Division, Wurzelbildung, Sinus, Exponentialfunktion usw.) maximal genau (semimorph) ausgeführt werden, ergeben sich auf Grund der endlichen Zahldarstellung bei der Berechnung von zusammengesetzten Ausdrücken mehr oder minder große Rechenfehler. Ein einfaches Beispiel soll dies verdeutlichen.

Die folgenden Maple[1]-Anweisungen berechnen den einfachen Ausdruck

$$a(x) := (\exp(\pi\sqrt{x}) - 640320) \cdot 10^{12} \tag{1}$$

mit verschiedenen Genauigkeiten an der Stelle $x = 169/3$. Der Wert der Variablen `Digits` steuert in den folgenden Schleifen die Anzahl der Mantissenstellen der verwendeten dezimalen Langzahlarithmetik.

[1] Es wurde Maple V, Release 4 unter Windows 95 verwendet.

```
> restart; Digits:=15;  a:= x-> ( exp(Pi*sqrt(x) )-640320 )*10^12;

                                  Digits := 15

          a := x -> 1000000000000 exp(Pi sqrt(x)) - 640320000000000000
> evalf(a(163.0/9));

                                     -4000.
------------------------------------------------------------------------
> for Digits from 14 to 19 do evalf( a(163.0/9) ) od;

                                    -320000.

                                     -4000.

                                     -3600.

                                     -2330.

                                     -2011.

                                     -2023.7

------------------------------------------------------------------------
> for Digits from 14 to 19 do evalf( a(163.0/9) ) od;

                                    -320000.

                                     -4000.

                                      2800.

                                      230.

                                      614.

                                      601.6
------------------------------------------------------------------------
```

Die erzeugten Ergebnisse[2] haben nicht nur verschiedene Größenordnungen, sondern sogar unterschiedliche Vorzeichen. Mit welcher Ziffernfolge beginnt der exakte Wert bzw. mit wievielen Mantissenstellen muß gerechnet werden, um z. B. fünf korrekte Ziffern im maschinell berechneten Ergebnis garantieren zu können?

Die Antworten auf solche und ähnliche Fragen wird die unten vorgestellte Fehlerfaktorarithmetik geben. Diese Arithmetik gibt zu einem maschinellen Berechnungsergebnis

[2] Daß man beim Ausführen der zweiten Schleife ganz andere Ergebnisse erhält als bei der Ausführung der ersten (diese ist offenbar mit der zweiten identisch), liegt vermutlich an der unausgegorenen Speicherung von bereits berechneten Zwischenergebnissen in Maple.

$\tilde{a}$ einen nichtnegativen Faktor $k(a)$ an, mit dem gilt

$$|a - \tilde{a}| \leq k(a) \cdot \varepsilon \,. \tag{2}$$

Dabei steht a für den exakten Wert und ε bezeichnet die relative Fehlerschranke der verwendeten Gleitkommaarithmetik. Ungleichung (2) besagt, daß der Betrag des absoluten Fehlers der berechneten Näherung $\tilde{a}$ kleiner $k(a) \cdot \varepsilon$ ist. Die Fehlerschranke wird in Einheiten der Rechengenauigkeit ausgedrückt.

Dieser kleine Trick erlaubt es, Schranken der Fehlerfaktoren zu bestimmen, ohne den numerischen Wert ε schon vorher zu kennen. Die gefundene Schranke des Fehlerfaktors gilt dann für beliebige Rechengenauigkeiten. Den konkreten Wert der absoluten Fehlerschranke findet man schließlich durch Multiplikation der Schranke des Fehlerfaktors $k(\cdot)$ mit dem aktuellen, der verwendeten Rechengenauigkeit entsprechenden Wert ε. Z. B. ist bei Berechnungen im IEEE-double-Format ε durch $\varepsilon = 0.5 \cdot 2^{1-53} = 1.116...E-16$ gegeben, dagegen ist bei einer dezimalen Langzahlarithmetik mit 1000 Stellen $\varepsilon = 0.5 \cdot 10^{1-1000}$.

2 Zur Notation

Es wird die folgende Notation verwendet:

$S = S(b, l, \underline{e}, \overline{e})$	Gleitkommaraster mit Basis b, Mantissenlänge l und Exponent e mit $\underline{e} \leq e \leq \overline{e}$
$S(2, 53, -1022, 1023)$	IEEE-Datenformat double
$\circ \in \{+, -, \bullet, /\}$	exakte reelle Operation
$\boxdot\,, \circ \in \{+, -, \bullet, /\}$	auf der Maschine ausgeführte Operation mit Rundung zur nächstgelegenen darstellbaren Zahl $\in S$
$\lvert a \circ b - a \boxdot b\rvert := \lvert(a \circ b) - (a \boxdot b)\rvert$	Implizite Klammerung beachten!
a	exaktes Argument (reelle Zahl)
$\tilde{a}$	auf der Maschine berechnete, i.a. fehlerbehaftete Größe, Approximation an a
$k(a)$	Fehlerfaktorschranke des absoluten Fehlers von $\tilde{a}$
k_a	Fehlerfaktor für die Paare $a, \tilde{a}$
$q(f)$	relative Fehlerschranke des Maschinenäquivalentes $\tilde{f}$ zur Funktion f, d. h. $\lvert f(x) - \tilde{f}(x)\rvert \leq q(f)\,\lvert f(x)\rvert$ für alle zulässigen, exakt darstellbaren Argumente x
ε	relative Rechengenauigkeit bzgl. S, d. h. $\varepsilon := \frac{1}{2} b^{1-l}$
$\overline{\varepsilon}$	Oberschranke für die Rechengenauigkeit ε , d. h. es wird $\varepsilon \leq \overline{\varepsilon}$ vorausgesetzt
`eps53`	Rechengenauigkeit (Maschinengenauigkeit) für IEEE-double Zahlformat, d. h. `eps53`$:= \frac{1}{2} 2^{1-53} = 1.11022\ldots \cdot 10^{-16} = \varepsilon$
$\mathbb{R}^+$	Menge der positiven reellen Zahlen

$I\!I\!R$	Menge der abgeschlossenen Intervalle über den reellen Zahlen
IS	Menge der Maschinenintervalle $IS = \{[\underline{a}, \overline{a}] \mid \underline{a}, \overline{a} \in S, \underline{a} \leq \overline{a}\}$
$\lvert A\rvert, A \in I\!I\!R$	$\lvert A\rvert := \max\limits_{a \in A} \lvert a\rvert$, Betragsmaximum
$<A>, A \in I\!I\!R$	$<A> := \min\limits_{a \in A} \lvert a\rvert$, Betragsminimum

3 Theoretische Ergebnisse

In den nachfolgenden Unterabschnitten werden Sätze formuliert, welche es erlauben, die Fehlerfortpflanzung bei ein- bzw. zweistelligen Grundoperationen sicher zu erfassen. Es wird davon ausgegangen, daß später Algorithmen untersucht werden sollen, die arithmetische Grundoperationen einer bestimmten (relativen) Genauigkeit

$$\varepsilon := \frac{1}{2} b^{1-l} \tag{3}$$

verwenden. Bei der Arithmetik kann es sich um die üblicherweise benutzte Gleitkommaarithmetik (in diesem Fall bezeichnet ε gerade die Maschinengenauigkeit), aber auch um eine Langzahlarithmetik mit l Mantissenziffern zur Basis b handeln. Das zugrunde liegende Gleitkommasystem ist $S(b, l)$.

Als Darstellung der numerischen Werte von Fehlern wird ein Produkt aus einem sogenannten Fehlerfaktor mit der Genauigkeitsschranke ε verwendet, d. h. die Fehler werden in Abhängigkeit von der zugrunde liegenden Genauigkeit der Arithmetik angegeben, ohne daß der numerische Wert der Genauigkeitsschranke ε tatsächlich bei den Abschätzungen verwendet wird. Durch diesen einfachen Trick kann man sich auf die Schrankenberechnung der Fehlerfaktoren zurückziehen. Die Berechnung von Fehlerschranken ist also vom numerischen Wert von ε entkoppelt.

Wie üblich können umfangreiche Berechnungsverfahren aus den Grundoperationen zusammengesetzt werden. Kennt man für ein solches numerisches Verfahren die zugehörige Schranke für den Betrag des maximalen Fehlerfaktors, so hat man für die unterschiedlichsten Arithmetiken die Möglichkeit, durch eine einzige Multiplikation mit dem aktuellen numerischen Wert ε eine worst-case-Schranke des Fehlers bzgl. der aktuell verwendeten Arithmetik zu erhalten. Allein die Kenntnis der Fehlerfaktorschranke für den Gesamtalgorithmus ist bereits ausreichend, um Aussagen darüber machen zu können, wieviele signifikante Stellen maximal verloren gehen. Man kann also unmittelbar sagen, wieviele Schutzziffern mit Sicherheit ausreichen werden, um eine gewünschte Ergebnisgenauigkeit für den Gesamtalgorithmus sicherzustellen.

In den folgenden Sätzen wird angenommen, daß die durch ε beschriebene Rechengenauigkeit nicht schlechter als eine 10stellige Dezimalarithmetik ist, d. h. , es wird

$$\varepsilon \leq \overline{\varepsilon} := \frac{1}{2} 10^{1-10} \tag{4}$$

gefordert. Die Größe $\overline{\varepsilon}$ wird immer dann verwendet, wenn der genaue Wert der tatsächlichen Rechengenauigkeit, für die die Fehlerfaktorabschätzungen durchgeführt werden

sollen, unerheblich für die ersten Ziffern der berechneten Fehlerfaktorschranken ist. Dabei wird durch die Forderung (4) die sichere Abschätzung der Fehlerfaktorschranken nach oben ermöglicht.

3.1 Einstellige Grundoperationen

Im folgenden werden zunächst einstellige Operationen, d. h. Funktionen behandelt. Es wird davon ausgegangen, daß eine relative Fehlerschranke $q(f)\cdot\varepsilon$ für die Funktion f und ihre Maschinenrealisierung $\tilde{f}$ bei Anwendung auf exakt darstellbare Argumente bekannt ist. Gesucht ist eine Oberschranke $k(f)$ für den maximalen Betrag der Fehlerfaktoren, die bei Anwendung von $\tilde{f}$ auf bereits gestörte Argumente $\tilde{a}$ entstehen können. Die folgenden Sätze erlauben die Bestimmung der Fehlerfaktorschranke $k(f)$.

Satz 3.1 *Es seien* $a \in A \in I\!I\!R, \varepsilon, k(a) \in I\!R^+, \overline{\varepsilon} \geq \varepsilon, \tilde{a} = a + k_a\varepsilon \in S$ *mit* $|k_a| \leq k(a)$. *Weiter sei die Funktion*

$$f : I\!R \supseteq D_f \to I\!R \text{ auf dem Intervall } A + [-k(a)\cdot\overline{\varepsilon},\ k(a)\cdot\overline{\varepsilon}]$$

differenzierbar. Das Maschinenanalogon $\tilde{f}$ *zu* f *möge für alle exakt darstellbaren Argumente* $x \in A + [-k(a)\cdot\overline{\varepsilon},\ k(a)\cdot\overline{\varepsilon}] \cap S$ *die relative Fehlerschranke* $q(f)\cdot\varepsilon$ *einhalten, d. h., es möge für alle zulässigen* x *die Abschätzung*

$$|f(x) - \tilde{f}(x)| \leq |f(x)|q(f)\cdot\varepsilon \tag{5}$$

erfüllt sein. Dann gelten die Darstellungen
a)

$$f(\tilde{a}) \quad = \quad f(a) + k_a\varepsilon\, f'(a + \Theta k_a\varepsilon) \quad \textit{mit}\ \ \Theta = \Theta(a,\tilde{a}) \in (0,1) \tag{6}$$

und
b)

$$\tilde{f}(\tilde{a}) = f(a) + k_f\cdot\varepsilon \ \ \textit{mit}\ \ k_f := (1+q_f\varepsilon)k_a\, f'(a+\Theta k_a\varepsilon) + q_f f(a)$$
und geeignet gewähltem q_f *mit* $|q_f| \leq q(f)$.

Beweis:

a): Für die Funktion f sind bezüglich des Intervalls $A + [-k(a)\,\overline{\varepsilon},\ k(a)\,\overline{\varepsilon}]$ die Voraussetzungen des Mittelwertsatzes der Differentialrechnung erfüllt. Aus diesem folgt unmittelbar die Darstellung (6).

b): Mit (5) und Aussage a) ergibt sich

$$\begin{aligned}\tilde{f}(\tilde{a}) &= f(\tilde{a})\cdot(1+q_f\varepsilon)\\ &= f(a) + q_f\varepsilon f(a) + k_a\varepsilon f'(a+\Theta k_a\varepsilon)(1+q_f\varepsilon),\end{aligned}$$

womit bereits alles gezeigt ist. □

Satz 3.2 *Mit den Bezeichnungen von Satz 3.1 und den dort angegebenen Voraussetzungen gilt mit*

$$k(f) \;:=\; (1+\overline{\varepsilon}\cdot q(f))\cdot k(a)\cdot |\, f'(\, A+[-\overline{\varepsilon},\ \overline{\varepsilon}]\cdot k(a)\,)\,| \;+\; |f(A)|\cdot q(f)$$

die Abschätzung

$$\bigwedge_{a\,\in\, A}\ \bigwedge_{\substack{\tilde{a}\,\in\, S\\ |a-\tilde{a}|\,\le\, k(a)\cdot\varepsilon}} |f(a)-\tilde{f}(\tilde{a})| \le k(f)\cdot\varepsilon\,.$$

Die Größe $k(f)$ ist also eine Oberschranke für die Beträge der möglichen Fehlerfaktoren k_f. Diese Fehlerfaktorschranke gilt für beliebige Kombinationen $a, \tilde{a}$, wenn nur $a \in A$ und $\tilde{a} = a + k_a\,\varepsilon$ mit $|k_a| \le k(a)$ erfüllt ist.

Beweis: Nach Satz 3.1 b) folgt

$$\tilde{f}(\tilde{a}) \;=\; f(a) + k_f$$

mit

$$\begin{aligned} k_f \;&=\; (1+q_f\varepsilon)k_a\, f'(a+\Theta k_a\,\varepsilon) + q_f f(a)\\ &\in\; [-(1+\overline{\varepsilon}\cdot q(f)),\ (1+\overline{\varepsilon}\cdot q(f))]\cdot k(a)\cdot f'\,(A+[-\overline{\varepsilon},\ \overline{\varepsilon}]\cdot k(a))\\ &\qquad +\ [-q(f),\ q(f)]\cdot f(A)\,. \end{aligned}$$

In dieser Darstellung sind beide Summanden der rechten Seite symmetrische Intervalle, so daß sich

$$|k_f| \le |\,(1+\overline{\varepsilon}\cdot q(f))\cdot k(a)\cdot f'(A+[-\overline{\varepsilon},\ \overline{\varepsilon}]\cdot k(a))\,| + q(f)\cdot|f(A)| = k(f)$$

ergibt. □

Die in Satz 3.2 hergeleitete Darstellung der Fehlerfaktorschranke kann mittels automatischer Differentiation berechnet werden. Sollten sich zu große Überschätzungen ergeben, so kann direkt auf Satz 3.1 und die Verwendung der formelmäßigen Ableitung von f zurückgegriffen werden, um möglicherweise schärfere Aussagen zu erhalten.

Beispiel 3.3 Für die Funktion $f(x) = \mathrm{lnp1}(x) := \ln(1+x)$ soll die zugehörige Fehlerfaktorschranke $k(\mathrm{ln1p})$ bestimmt werden.
Mit $f'(x) = \dfrac{1}{1+x}$ findet man gemäß Satz 3.2

$$k(\mathrm{ln1p}) \;:=\; \frac{(1+\overline{\varepsilon}\,q(\mathrm{lnp1}))\,k(a)}{|A+[-\overline{\varepsilon},\overline{\varepsilon}]\,k(a)|} + |\ln(1+A)|\cdot q(\mathrm{ln1p}).$$

Mit dieser Fehlerfaktorschranke gilt dann

$$\bigwedge_{a\,\in\, A}\ \bigwedge_{\substack{\tilde{a}\,\in\, S\\ |a-\tilde{a}|\,\le\, k(a)\varepsilon}} |\mathrm{ln1p}(a) - \widetilde{\mathrm{ln1p}}(\tilde{a})| \le k(\mathrm{ln1p})\cdot\varepsilon\,.$$

Dabei ist angenommen, daß die Größen $A \in I\!I\!R, k(a), \overline{\varepsilon} \in I\!R^+$ den Voraussetzungen von Satz 1 entsprechen.

Beispiel 3.4 Für die Wurzelfunktion $f(x) := \sqrt{x}$ ergibt sich die gesuchte Fehlerfaktorschranke $k(\mathrm{sqrt})$ zu

$$k(\mathrm{sqrt}) \;:=\; \frac{(1+\overline{\varepsilon}\,q(\mathrm{sqrt}))\,k(a)}{2\sqrt{A+[-\overline{\varepsilon},\overline{\varepsilon}]k(a)}} + q(\mathrm{sqrt})\cdot\sqrt{A}\,.$$

Dabei wird vorausgesetzt, daß das Argument der Wurzel größer 0 ist.

Nimmt man an, daß die Wurzelfunktion auf der Maschine maximal genau berechnet wird (dies wird z. B. vom IEEE-Standard [10] für die dort definierten Datenformate verlangt), so findet man

$$k(\mathrm{sqrt}) \;:=\; \frac{(1+\overline{\varepsilon})\,k(a)}{2\sqrt{A+[-\overline{\varepsilon},\overline{\varepsilon}]k(a)}} + \sqrt{A}\,.$$

In diesem Fall gilt nämlich $q(\mathrm{sqrt}) = 1$, d. h. $|\sqrt{x} - \widetilde{\sqrt{x}}| \le \sqrt{x}\cdot\varepsilon$ für alle zulässigen und darstellbaren Argumente x.

3.2 Zweistellige Grundoperationen

Für die zweistelligen Grundoperationen Addition, Subtraktion, Multiplikation und Division gibt der folgende Satz Schranken für die maximalen Beträge der zugehörigen Fehlerfaktoren an. Man beachte, daß die Rechengenauigkeit ε, die Argumentintervalle A, B, die maximalen Faktoren der Störungen der Argumenete $k(a), k(b)$ sowie die Größe $\overline{\varepsilon} \ge \varepsilon$ als bekannt angenommen werden.

Satz 3.5 *Es seien $a \in A \in I\!I\!R, b \in B \in I\!I\!R, \varepsilon, k(a), k(b) \in I\!R^+, \overline{\varepsilon} \ge \varepsilon$. Weiter mögen die Grundoperationen $\circ \in \{+,-,\cdot,/\}$ und deren Maschinenäquivalent $\boxdot$ mit $\circ \in \{+,-,\cdot,/\}$ der Genauigkeitsanforderung*

$$\bigwedge_{\circ\,\in\,\{+,-,\cdot,/\}} \;\bigwedge_{a,b\,\in\,S} |a\circ b - a\,\boxdot\, b| \le |a\circ b|\cdot\varepsilon$$

genügen (bzgl. S handelt es sich also um maximal genaue Operationen).
Die angegebenen Voraussetzungen führen auf die folgenden Schranken für die Beträge der Fehlerfaktoren, die sich durch Anwendung von Grundoperationen auf fehlerbehaftete Argumente ergeben:

a) *Mit*

$$k(\mathrm{add}) \;:=\; |A+B| + (1+\overline{\varepsilon})(k(a)+k(b)) \text{ gilt}$$

$$\bigwedge_{a\,\in\,A} \;\bigwedge_{\substack{\tilde{a}\,\in\,S,\ |a-\tilde{a}|\,\le\,k(a)\,\varepsilon \\ \tilde{b}\,\in\,S,\ |b-\tilde{b}|\,\le\,k(b)\,\varepsilon}} |a+b-\tilde{a}\boxplus\tilde{b}| \le k(\mathrm{add})\cdot\varepsilon,$$
$$b\,\in\,B$$

d. h. die gesuchte Fehlerschranke der Addition ist durch $k(\mathrm{add})$ gegeben.

Entsprechend ergeben sich die Fehlerfaktorschranken

b) der Subtraktion zu

$$k(\mathrm{sub}) \;:=\; |A-B| + (1+\overline{\varepsilon})(k(a)+k(b)),$$

c) der Multiplikation zu

$$k(\mathrm{mul}) \;:=\; |A||B| + (1+\overline{\varepsilon})\,(|A|k(b) + |B|k(a) + k(a)k(b)\overline{\varepsilon})$$

d) und der Division zu

$$k(\mathrm{div}) \;:=\; \frac{1}{< B > - k(b)\overline{\varepsilon}} \quad (k(a) + (|A| + k(a)\overline{\varepsilon})(1 + h + 2h^2 \cdot \overline{\varepsilon})),$$

wobei h die Größe $h := \dfrac{k(b)}{< B >}$ bezeichnet. Hier wird zusätzlich vorausgesetzt,

daß $< B > - k(b)\,\overline{\varepsilon} > 0$ und $\dfrac{k(b)\overline{\varepsilon}}{< B >} < \dfrac{1}{2}$.

Diese Voraussetzungen sind in der Regel erfüllt, werden später aber zur Sicherheit in den Programmwerkzeugen automatisch mit abgeprüft. Gerade im Hinblick auf Langzahlrechnungen werden hier, im Gegensatz zu den Ausführungen in [8], nur Situationen betrachtet, in denen weder Unter- noch Überlauf eintritt.

Beweis: Die in [8] in den Sätzen 1 bis 4 bewiesenen Fehlerschranken für die Fehlerfortpflanzung bei Anwendung der Grundoperationen auf bereits fehlerbehaftete Argumente können hier als Ausgangspunkt verwendet werden. Die dort verwendeten Fehlergrößen $\triangle(\cdot)$ müssen entsprechend als Produkte $k(\cdot)\cdot\varepsilon$, bzw. weiter vergröbert als Produkte $k(\cdot)\cdot\overline{\varepsilon}$ interpretiert werden. Weiter ist zu berücksichtigen, daß bei den hier durchgeführten Fehlerabschätzungen im Gegensatz zu denen in [8] kein Ergebnisunterlauf erlaubt ist. □

4 Umsetzung in Software-Werkzeuge

Unter Verwendung von intervallarithmetischen Operationen können die im letzten Kapitel angegebenen Resultate in Software-Werkzeuge umgesetzt werden. Die entsprechenden Routinen `kLn1p, kSqrt, kAdd, kSub, kMul, kDiv` berechnen dann automatisch verläßliche Gleitkommaoberschranken für die Fehlerfaktoren der jeweiligen Grundoperation. Für die im Programm auftauchenden Größen `alpha`, `beta`, `rA` und `rB` handelt es sich um exakt darstellbare Größen, mit `alpha` $\supseteq A$, `beta` $\supseteq B$, `rA` $\geq r(a)$ und `rB` $\geq r(b)$. Für die berechneten Ergebnisse gilt `kAdd` $\geq k(\mathrm{add})$, `kSub` $\geq k(\mathrm{sub})$, usw. Genauer gilt nach einer Zuweisung der Form

```
kResult := kADD(alpha, beta, kA, kB);
```

mit der berechneten Gleitkommazahl `kResult`

$$\bigwedge_{\substack{a \in A \subseteq \texttt{alpha} \\ b \in B \subseteq \texttt{beta}}} \;\bigwedge_{\substack{\tilde{a} \in S, |a-\tilde{a}| \leq k(a)\,\varepsilon \\ \tilde{b} \in S, |b-\tilde{b}| \leq k(b)\,\varepsilon}} |a + b - \tilde{a} \boxplus \tilde{b}| \leq k(\mathrm{add}) \cdot \varepsilon \leq \texttt{kResult} \cdot \varepsilon\,.$$

Die Formeln der Sätze 3.2 und 3.5 müssen bei der programmtechnischen Umsetzung mit geeigneten gerichteten Rundungen bzw. mit Hilfe von Maschinenintervalloperationen berechnet werden. Z. B. bedeutet die Operation `+>` die nach oben gerichtete Addition auf der Maschine. Wertebereiche von Funktionen über Intervallen werden durch intervallmäßige Funktionsauswertungen sicher eingeschlossen.

Unter Verwendung eines Operatorkonzeptes kann die Handhabung der Fehlerfaktorarithmetik ganz erheblich erleichtert werden. Die Einführung des neuen Datentyps `Error` gemäß

```
global type Error = global record  { Neuer Datentyp                      }
              Value : interval;    { Einschliessung der korrekten Werte  }
              AbsErr: real;        { Zugehoeriger maximaler Fehlerfaktor }
            end;
```

erlaubt das Überladen der Grundoperationen und Funktionen für diesen Datentyp. Z. B. kann dies in PASCAL-XSC [11] für die Addition durch die Anweisungsfolge

```
global operator + (x, y: Error) erg: Error;
begin
  erg.AbsErr:= kAdd(x.Value, y.Value, x.AbsErr, y.AbsErr);
  erg.Value := x.Value + y.Value;
end;
```

geschehen. Mit den so überladenen Operatoren und Funktionen können arithmetische Ausdrücke wieder in ihrer gewohnten mathematischen Notation programmiert werden. Fehlerabschätzungen für ganze Programme können damit oft durch einfaches Ersetzen des Datentyps `real` durch `Error` und Einfügen einiger Schreibanweisungen durchgeführt werden. Als Beispiel betrachte man das Programm zur Berechnung von π auf große Stellenzahl, welches im Anhang ab Seite 19 abgedruckt ist. Das verwendete Iterationsverfahren wird im nächsten Abschnitt beschrieben.

5 Fehleranalyse eines Iterationsverfahrens für π

Die folgende dreigliedrige Iterationsvorschrift zur näherungsweisen Berechnung von π beruht auf der Theorie der elliptischen Integrale und den Eigenschaften der arithmetisch-geometrischen Mittelbildung[3]:

Mit

$$a_0 := \sqrt{2},\ b_0 := 0,\ p_0 := 2 + \sqrt{2}$$

[3]Dieses Verfahren wurde Mitte der 80er Jahre zur Berechnung von 4 Millionen Stellen von π verwendet.

$$\left.\begin{aligned} a_{n+1} &:= \frac{1}{2}\left(\sqrt{a_n} + \frac{1}{\sqrt{a_n}}\right) \\ b_{n+1} &:= \sqrt{a_n}\,\frac{1+b_n}{a_n+b_n} \\ p_{n+1} &:= p_n\, b_{n+1}\,\frac{1+a_{n+1}}{1+b_{n+1}} \end{aligned}\right\} \quad n = 0, 1, 2, \ldots$$

gilt (bei Rechnung in $\mathbb{R}$) für die n-te Näherung p_n an π die relative Fehlerschranke

$$\left|\frac{\pi - p_n}{\pi}\right| \leq \frac{1}{2} 10^{1-2^n}. \tag{7}$$

Das Verfahren ist quadratisch konvergent, das $n-$te Glied enthält mindestens 2^n korrekte Ziffern [5].

Es soll nun untersucht werden, wieviele Stellen bei dieser Iteration möglicherweise durch Rundungsfehler verloren gehen. Genauer soll eine gewisse Anzahl von Schutzziffern bestimmt werden, so daß die durch das Verwenden einer endlichen Arithmetik auftretenden Fehler nicht über die angegebene Schranke (7) des maximalen Approximationsfehlers anwachsen.

Die Berechnung der Fehlerfaktorschranken geht davon aus, daß alle arithmetischen Operationen mit derselben Genauigkeit ausgeführt werden. Diese Annahme ist bei der vorliegenden Iteration gerechtfertigt, da das Verfahren, im Gegensatz z. B. zum Newtonverfahren, nicht selbstkorrigierend ist. Alle Berechnungen müssen mit der gewünschten Endergebnisgenauigkeit durchgeführt werden.

Mit dem in Anhang 8.1 angegebenen Programm zur Fehlerabschätzung des obigen Verfahrens findet man eine Fehlerschranke für den Betrag des absoluten Fehlers nach 32 Iterationsschritten kleiner $27559 \cdot \varepsilon$. Diese Schranke gilt auch für alle vorherigen Iterationen (die Fehlerfaktorschranken sind monoton wachsend, wie man Anhang 2 entnimmt).

Zur Bewertung des scheinbar großen Fehlerfaktors muß berücksichtigt werden, daß 32 Iterationsschritte bei einer exakten Rechnung im Körper der reellen Zahlen mindestens $2^{32} = 4294967296$ korrekte Ziffern generieren würden. Der erhaltene Fehlerfaktor von 27559 (bzw. dessen Zehnerlogarithmus) sagt nun aus, daß mit einer $(2^{32}+3)$-stelligen Dezimalarithmetik nach 32 Iterationsschritten ein Ergebnis erzielt wird, bei dem mindestens $2^{32}-1$ Dezimalen korrekt sind. Einer solchen Arithmetik entspricht die Wahl

$$\varepsilon := \frac{1}{2} 10^{1-(2^{32}+3)},$$

so daß wie folgt abgeschätzt werden kann:

$$\left|\frac{\pi - \tilde{p}_{32}}{\pi}\right| \quad \leq \quad \left|\frac{\pi - p_{32}}{\pi}\right| + \left|\frac{p_{32} - \tilde{p}_{32}}{\pi}\right|$$

$$\leq \frac{1}{2}\cdot 10^{1-2^{32}} + \frac{27559\cdot\varepsilon}{\pi}$$

$$\leq \frac{1}{2}\cdot 10^{1-2^{32}}\cdot(1+8.8) \leq \frac{1}{2}\cdot 10^{1-(2^{32}-1)}\ .$$

Hieraus folgt

$$\pi \in [\tilde{p}_{32} - \frac{1}{2}10^{2-2^{32}},\ \tilde{p}_{32} + \frac{1}{2}10^{2-2^{32}}]\ .$$

Drei Schutzziffern reichen also aus, um den durch die endliche Arithmetik eingeschleppten Fehler sicher unter das Neunfache des Verfahrensfehlers zu drücken. Man verliert dann gegenüber einer exakten Rechnung in $\mathbb{R}$ maximal eine Dezimale. Bei einer Langzahliteration mit über 4 Milliarden Dezimalstellen machen sich drei Schutzziffern laufzeitmäßig nicht bemerkbar.

Man beachte, daß ohne die Produktdarstellung (2) der Fehlerschranke obige Berechnungen nicht durchgeführt werden könnten. Der Wert von ε ist, falls er sich auf eine Langzahlarithmetik bezieht, in der Regel so klein, daß die Fehlerschranken im Unterlaufbereich der üblicherweise verwendeten Gleitkommaarithmetiken (z. B. IEEE-double-Format) liegen. Diese Schwierigkeit wird durch das alleinige Abschätzen der Fehlerfaktoren umgangen.

6 Zurück zum Beispiel aus der Einleitung

Es wird noch einmal die Auswertung des Ausdrucks

$$a(x) := \exp(\pi\sqrt{x}) - 640320 \tag{8}$$

an der Stelle $x := 163/9$ untersucht[4]. Das folgende Programmlisting zeigt die Berechnung der Fehlerfaktoren für alle Zwischenergebnisse des obigen Ausdrucks.

```
PROGRAM sqrt1639;
{------------------------------------------------------------------------------}
{ Fehlerfaktoren der absoluten Fehler bei der Berechnung des Ausdrucks       }
{                                                                            }
{          exp(pi*sqrt(x) - 640320)   fuer  x:= 163/9                        }
{                                                                            }
{------------------------------------------------------------------------------}
USE i_ari;                                    { Intervallarithmetik          }
USE abs_ari;                                  { Fehlerschrankenarithmetik    }

VAR
  x, pi, c163, c9, c640320: Error;          { Vereinbarung der Konstanten    }
BEGIN
  pi.Value := 4*arctan(intval(1));
  { Berechne den absoluten Fehler f a k t o r  fuer Konstante pi:            }
  pi.AbsErr:= diam(pi.Value)/Eps53; { IEEE: Epsilon= Eps53:= 0.5*2**(1-53) }

  c163   :=    163;
  c9     :=      9;
```

[4]Der Skalierungsfaktor 10^{12} aus Formel (1) wird hier nicht betrachtet. Bei Rechnung zur Basis 10 führt er zu keinem weiteren Fehler.

```
  c640320:= 640320;

  writeln('pi.AbsErr                      :', pi.AbsErr:12:2, '*Epsilon');
  x:= c163/c9;
  writeln('163/9.AbsErr                   :',  x.AbsErr:12:2, '*Epsilon');
  x:= sqrt(x);
  writeln('sqrt(163/9).AbsErr             :',  x.AbsErr:12:2, '*Epsilon');
  x:= pi*x;
  writeln('pi*sqrt(163/9).AbsErr          :',  x.AbsErr:12:2, '*Epsilon');
  x:= exp(x);
  writeln('exp(pi*sqrt(163/9)).AbsErr     :',  x.AbsErr:12:2, '*Epsilon');
  x:= x - c640320;
  writeln('exp(pi*sqrt(163/9))-640320.AbsErr:',  x.AbsErr:12:2, '*Epsilon');
  writeln('Einschliessung des Wertes des Gesamtausdrucks:');
  writeln('  ', x.Value);
END.
```

Das Programm führt auf die folgende Ausgabe:

```
pi.AbsErr                       :        4.00*Epsilon
163/9.AbsErr                    :       18.11*Epsilon
sqrt(163/9).AbsErr              :        6.38*Epsilon
pi*sqrt(163/9).AbsErr           :       50.45*Epsilon
exp(pi*sqrt(163/9)).AbsErr      : 32942641.70*Epsilon
exp(pi*sqrt(163/9))-640320.AbsErr: 32942641.71*Epsilon
Einschliessung des Wertes des Gesamtausdrucks:
            [              -1.4E-009,               3.3E-009 ]
```

Der Programmausgabe kann man entnehmen, daß der tatsächliche Wert von $a(163/9)$ betragsmäßig kleiner $3.3 \cdot 10^{-9}$ ist. Dies führt zusammen mit der Fehlerschranke des absoluten Fehlers $32942641.71 \cdot \varepsilon$ zu der Aussage, daß man bei der Ausdrucksauswertung ca. 17 Stellen verlieren wird. Um diese Aussage zu präzisieren, wird obiges Programm noch einmal ausgeführt, jetzt allerdings unter Einbindung eines Moduls `abs_ari2`. In diesem Modul wird die begleitende Intervallauswertung (d. h. die Einschließungen der Teilausdrücke <Variablenname>`.Value`) mit Langzahlintervallrechnung des Multi-precision-Paketes `mp_ari` [14] berechnet. Es ergibt sich die folgende Programmausgabe:

```
pi                                 3.15*Epsilon
163/9                             18.12*Epsilon
sqrt(163/9)                        6.39*Epsilon
pi*sqrt(163/9)                    46.80*Epsilon
exp(pi*sqrt(163/9))         30603474.50*Epsilon
exp(pi*sqrt(163/9))-640320  30603474.51*Epsilon
Einschliessung fuer Wert des Gesamtausdrucks:
   6.048637350490160394691586211 9E-010
   6.048637350490160394691586212 0E-010
Relative Fehlerschranke:    5.0596E+016*Epsilon
```

Die hier angegebene relative Fehlerschranke berechnet sich aus dem absoluten Fehlerfaktor `30603474.51` und der genauen Einschließung des tatsächlichen Wertes $a(163/9) =$ `6.04863735...E-010` . Das Ergebnis besagt, daß bei einer approximativen Berechnung des Ausdrucks mit $17 + k$ Dezimalziffern, mindestens k führende Ziffern korrekt sind!

7 Schlußbetrachtung

Die vorgestellten Möglichkeiten zur a priori Fehlerkontrolle können in vielfältiger Weise erweitert werden. Zunächst sollten alle in höheren Programmiersprachen gängigen mathematischen Funktionen als Grundoperationen in das Modul `Abs_Ari` aufgenommen werden. Der Mittelwertsatz spielt hier die zentrale Rolle. Weiterhin sollten entsprechende Softwarewerkzeuge für *relative* Fehlerabschätzungen entwickelt werden.

Auch sollte versucht werden, Matrix/Vektor-Operationen, d. h. Skalarprodukte, als Grundoperationen mit Fehlerabschätzungsmöglichkeit zu behandeln. Mit einem solchen Werkzeug wäre es sehr einfach, Fehleruntersuchungen von Algorithmen mit Skalarproduktauswertungen für verschiedene Realisierungen der Skalarproduktoperation (semimorph, herkömmliche Schleifenprogrammierung) miteinander zu vergleichen.

Die Realisierung und der einfache Einsatz solcher Softwarewerkzeuge beruht auf dem Vorhandensein von Intervalloperationen sowie der Möglichkeit, Funktionen und Operationen für neue Datentypen zu überladen. Der Anwender der Fehlerabschätzungsroutinen kann dann durch Austausch von Datentypen und durch Einfügen von Befehlen zur Ausgabe der Fehlerfaktorschranken in seinem Originalprogramm eine verläßliche a priori Fehlerabschätzung seines Verfahrens automatisch mit dem Rechner durchführen.

Die beschriebene Methode wurde erfolgreich im Zusammenhang mit Funktionsimplementierungen [4, 8, 15] eingesetzt. Sichere a priori Fehlerabschätzungen sind besonders dann unerlässlich, wenn schnelle Routinen zur verifizierten Wertebereichseinschließung realisiert werden sollen. Auch bei den in [16] diskutierten Fragestellungen können Fehlerabschätzungen nach den vorgestellten Verfahren durchgeführt werden.

Vorstufen zu der hier diskutierten Methodik finden sich z. B. in [12, 6, 13, 3].

8 Anhang

8.1 Programmlisting des Moduls `Abs_Ari`

Das folgende Programmlisting gibt einen Auszug des PASCAL-XSC Moduls `Abs_Ari` zur automatischen Berechnung von Oberschranken für die Beträge von Fehlerfaktoren bei Grundoperationen. Die Argumente sind mit bekannten absoluten Fehlern angenommen. Es werden Fehlerfaktorschranken für die absoluten Fehler der Grundoperationen berechnet.

```
MODULE abs_ari;
{------------------------------------------------------------------------}
{              F e h l e r -  f a k t o r -  a r i t h m e t i k          }
{                                                                        }
{         zur sicheren Abschaetzung von   a b s o l u t e n   Fehlern     }
{                                                                        }
{ Epsilon bezeichnet die Genauigkeit der arithmetischen (Langzahl-)      }
{ Operationen, fuer die die Fehlerabschaetzung durchgefuehrt werden soll.}
{ Dabei wird vorausgesetzt, dass die lokale Hilfsgroesse EpsQuer dieses  }
{ Moduls eine Oberschranke fuer Epsilon darstellt!                       }
{------------------------------------------------------------------------}

USE i_ari;  { Intervallarithmetik einbinden        }
USE iostd;  { Ermoeglicht sauberen Programmabbruch }
```

```
{-------------------------------------------------------------------------}
GLOBAL TYPE Error = GLOBAL RECORD    { Neuer Datentyp                     }
              Value : interval;      { Einschliessung der korrekten Werte }
              AbsErr: real;          { Zugehoeriger maximaler Fehlerfaktor }
            END;
{                                                                         }
{ Ist a eine Variable vom Typ error, so liegt die durch diese Variable    }
{ repraesentierte exakte Groesse im Intervall a.Value. Fuer die           }
{ gestoerten Groessen gilt die Fehlerschranke a.AbsErr*Epsilon.           }
{-------------------------------------------------------------------------}

CONST EpsQuer = 1e-10;     { Es muss Epsilon < EpsQuer  sein!!!           }

  . . .

{-------------------------------------------------------------------------}
{  Fehlerfaktorschranken der Funktionen Sqrt, Ln1p, Exp  bei deren        }
{  Anwendung auf ungestoerte Argumente:                                   }
{-------------------------------------------------------------------------}
CONST qSQRT =    1;
      qLn1p = 2.26;
      qExp  = 2.13;

. . .

{-------------------------------------------------------------------------}
{                                                                         }
{             F e h l e r f a k t o r a r i t h m e t i k               }
{                                                                         }
{                   fuer Grundoperationen +, -, *, /                      }
{                                                                         }
{ Absolute Fehlerschranken k(a)*Epsilon und k(b)*Epsilon der beiden        }
{ Operanden sind bekannt. Die exakten Werte der Operanden liegen in       }
{ den Intervallen alpha und beta. Epsilon bezeichnet die relative         }
{ Genauigkeit der (Langzahl-)Arithmetik. Bei den Operationen darf         }
{ kein Unterlauf eintreten, d.h., es wird davon ausgegangen, dass         }
{ in einem solchen Fall die Berechnung mit einer Unterlaufmeldung         }
{ abgebrochen wird. Weiterhin  muss Epsilon < EpsQuer ( siehe             }
{ obige Definition von EpsQuer) gelten!                                   }
{                                                                         }
{-------------------------------------------------------------------------}

GLOBAL FUNCTION kAdd(alpha, beta: interval; kA, kB: real ): real;
  {-----------------------------------------------------------------}
  { Berechnet wird der absolute Fehlerfaktor kAdd bei einer         }
  {   Addition von fehlerbehafteten Groessen.                       }
  {-----------------------------------------------------------------}
  { Die exakten Werte des 1. Summanden liegen im Intervall alpha.   }
  { Der absolute Fehler der fehlerbehafteten Werte ist durch        }
  { kA*Epsilon beschraenkt.                                         }
  { Die exakten Werte des 2. Summanden liegen im Intervall beta.    }
  { Der absolute Fehler der fehlerbehafteten Werte ist durch        }
  { kB*Epsilon beschraenkt.                                         }
  { Der absolute Fehler des Verknuepfungsergebnisses ist sicher     }
  { <= kAdd*Epsilon                                                 }
  {-----------------------------------------------------------------}
BEGIN
  IF (kA=0) AND (kB=0) AND ( (alpha=0) OR (beta=0) ) THEN
     kAdd:= 0
  ELSE
     kAdd:= (1 +> EpsQuer) *> (kA +> kB) +> MaxAbs(alpha + beta);
END;
```

```
   . . .

{-------------------------------------------------------------------------------}
{                                                                               }
{              F e h l e r f a k t o r a r i t h m e t i k                      }
{                                                                               }
{            fuer Funktionen  SQRT,  EXP  und  LN1P:= ln(1+x)                   }
{                                                                               }
{ Die absolute Fehlerschranke k(a)*Epsilon des Argumentes ist bekannt.          }
{ Der exakte Wert des Operanden liegt im Intervall alpha.                       }
{ Der absolute Fehler des berechneten Funktionsergebnisses ist                  }
{ sicher <= kFunktion*Epsilon                                                   }
{-------------------------------------------------------------------------------}

   . . .


GLOBAL FUNCTION kLn1p(Alpha: interval; kA: real): real;
{-------------------------------------------------------------------}
{ Berechnet wird absolute Fehlerfaktorschranke fuer Ln1p             }
{-------------------------------------------------------------------}
{ Fehlerschranke qLn1p fuer ln1p-Realisierung in ff_ari              }
BEGIN
   kLn1:= MaxAbs( ( 1 +> EpsQuer*>qLn1p)*intval(-kA, kA )
               / ( Alpha + intval(-kA*<EpsQuer, ka*>EpsQuer) )
                         + intval(-qLn1p, qLn1p) * ln(1+Alpha) );
END;

{-------------------------------------------------------------------------------}
{  F u n k t i o n s u e b e r l a d u n g e n   fuer neuen Datentyp Error }
{-------------------------------------------------------------------------------}

   . . .

GLOBAL FUNCTION ln1p(x: Error): Error;
VAR erg: Error;
BEGIN
   erg.AbsErr:= kLn1p(x.Value, x.AbsErr);
   erg.Value := ln(1+x.Value);
   ln1p:= erg;
END;


{-------------------------------------------------------------------------------}
{  O p e r a t o r d e f i n i t i o n e n  zur  Fehlerfaktorarithmetik    }
{-------------------------------------------------------------------------------}

GLOBAL OPERATOR * (x, y: Error) erg: Error;
BEGIN
  erg.AbsErr := kMul(x.Value, y.Value, x.AbsErr, y.AbsErr);
  erg.Value  := x.Value * y.Value;
END;

   . . .

BEGIN
END.  { module abs_ari }
{-------------------------------------------------------------------------------}
```

Der vollständige Quelltext, möglicherweise mit in der vorliegenden Arbeit noch nicht besprochenen Erweiterungen, kann über E-Mail angefordert werden.

8.2 Programm zur Analyse des Iterationsverfahrens für π

Es folgt der Ausdruck des PASCAL-XSC [11] Programms zur Fehleranalyse des angegebenen Iterationsverfahrens zur Berechnung von Langzahlnäherungen für π. Dieses Programm verwendet die im Modul `Abs_Ari` zur Verfügung gestellte Fehlerfaktorarithmetik. Durch den Einsatz des neuen Datentyps `Error` sowie der überladenen Funktionen und Operatoren unterscheidet sich das Programm nur geringfügig von einem Programm, welches tatsächlich für die Berechnung der Näherungen geschrieben werden muß.

```
PROGRAM pi3_abs;
{------------------------------------------------------------------------------}
{ Bestimmung der Fehlerfaktoren fuer eine quadratisch konvergente              }
{ Iteration zur Berechnung von Pi                                              }
{------------------------------------------------------------------------------}
USE i_ari;       { Intervallarithmetik einbinden                               }
USE abs_ari;     { Absolute Fehlerfaktorarithmetik einbinden                   }
CONST test = false;     { Keine zusaetzlichen Testausgaben                     }

VAR
  An, Bn, Pn      : Error;          { Alte Generation der Iterationswerte }
  Anp1, Bnp1, Pnp1: Error;          { Neue Generation der Iterierten      }
  zp5, two        : Error;          { Konstanten 0.5 und 2                }
  n, nMax         : integer;        { Iterationszaehler mit Obergrenze    }
  Eps             : string;         { Textkonstante '*Epsilon'            }
BEGIN
  eps:= '*Epsilon ';
  two:= 2;     { Groessen des Datentyps Error; sowohl 2 als auch 0.5 sind  }
  zp5:= 0.5;   { exakt darstellbar, so dass ueberladene Zuweisung          }
               { verwendet werden darf.                                    }

  { Initialisierung der drei Folgen der Iteration:                         }
  Anp1:= sqrt(two);   { = a0 }
  IF test THEN writeln('Anp1.abserr: ', anp1.abserr);

  Bnp1:= 0;           { = b0 }

  Pnp1:= 2 + anp1;    { = p0 }
  IF test THEN writeln('Pnp1.abserr: ', pnp1.abserr);

  writeln;
  writeln('   n         Absolute Fehlerschranken fuer An, Bn und Pn');
  writeln;
  nMax:= 32;
  FOR n:= 1 TO nMax DO BEGIN
     An:= Anp1;
     Bn:= Bnp1;
     Pn:= Pnp1;
     { Berechne die neuen Iterierten: }
     Anp1:= zp5*(sqrt(An) + sqrt(1/An));
     Bnp1:= sqrt(An)*(1 +Bn)/(An + Bn);
     Pnp1:= Pn*Bnp1*(1 + Anp1)/(1 + Bnp1);

     writeln( n: 4, '  ',
       Anp1.abserr:9:1:1, eps, Bnp1.abserr:9:1:1, eps, Pnp1.abserr:9:1:1, eps);
     IF test AND (n < 5) THEN
        writeln( 'n,  Pi-Pn.inf ', n:4,  '  ',  4*arctan(1) - inf(Pnp1.value) );
  END;
  writeln;
  writeln('Korrekte Ziffernzahl ca. 2**', nMax, ' = ', power(2,nMax):12:1 );
END.  { Programm pi3_abs.p }
```

```
{------------------------------------------------------------------------}

$off  { Ab hier alles Kommentar }

Ausgabe dieses Programms:

   n          Absolute Fehlerschranken fuer An, Bn und Pn

   1        3.8*Epsilon       3.8*Epsilon      46.7*Epsilon
   2        5.2*Epsilon      13.0*Epsilon     131.5*Epsilon
   3        5.9*Epsilon      23.1*Epsilon     265.1*Epsilon
   4        6.2*Epsilon      33.9*Epsilon     450.1*Epsilon
   5        6.4*Epsilon      45.1*Epsilon     687.9*Epsilon
   6        6.5*Epsilon      56.4*Epsilon     979.2*Epsilon
   7        6.5*Epsilon      67.8*Epsilon    1324.4*Epsilon
   8        6.5*Epsilon      79.3*Epsilon    1723.6*Epsilon
   9        6.5*Epsilon      90.7*Epsilon    2176.9*Epsilon
  10        6.5*Epsilon     102.2*Epsilon    2684.3*Epsilon
  11        6.5*Epsilon     113.7*Epsilon    3246.0*Epsilon
  12        6.5*Epsilon     125.2*Epsilon    3861.8*Epsilon
  13        6.5*Epsilon     136.7*Epsilon    4531.7*Epsilon
  14        6.5*Epsilon     148.2*Epsilon    5255.9*Epsilon
  15        6.5*Epsilon     159.7*Epsilon    6034.3*Epsilon
  16        6.5*Epsilon     171.2*Epsilon    6866.9*Epsilon
  17        6.5*Epsilon     182.7*Epsilon    7753.6*Epsilon
  18        6.5*Epsilon     194.2*Epsilon    8694.6*Epsilon
  19        6.5*Epsilon     205.7*Epsilon    9689.7*Epsilon
  20        6.5*Epsilon     217.2*Epsilon   10739.0*Epsilon
  21        6.5*Epsilon     228.7*Epsilon   11842.6*Epsilon
  22        6.5*Epsilon     240.2*Epsilon   13000.3*Epsilon
  23        6.5*Epsilon     251.7*Epsilon   14212.2*Epsilon
  24        6.5*Epsilon     263.2*Epsilon   15478.3*Epsilon
  25        6.5*Epsilon     274.7*Epsilon   16798.6*Epsilon
  26        6.5*Epsilon     286.2*Epsilon   18173.1*Epsilon
  27        6.5*Epsilon     297.7*Epsilon   19601.8*Epsilon
  28        6.5*Epsilon     309.2*Epsilon   21084.6*Epsilon
  29        6.6*Epsilon     320.7*Epsilon   22621.7*Epsilon
  30        6.6*Epsilon     332.2*Epsilon   24213.0*Epsilon
  31        6.6*Epsilon     343.7*Epsilon   25858.4*Epsilon
  32        6.6*Epsilon     355.2*Epsilon   27558.1*Epsilon

Korrekte Ziffernzahl ca. 2**32 = 4294967296.0
```

Die letzte Ergebniszeile in der Tabelle (n=32) zeigt, daß der absolute Fehler nach 32 Iterationsschritten, welcher durch Verwendung einer endlichen Arithmetik eingeschleppt wird, $27559 \cdot \varepsilon$ nicht übersteigt. Diese Aussage wird in Abschnitt 5 verwendet und näher erläutert.

Literatur

[1] Abramowitz, M., Stegun, I. A.: *Handbook of Mathematical Functions*. Nat. Bur. Standards, Appl. Math. Series, No. **55**, U.S. Government Printing Office, Washington, D.C. 1964.

[2] Black, Ch. M., Burton, R. B., Miller, T. H.: *The Need for an Industry Standard of Accuracy for Elementary-Function Programs*. ACM Trans. on Math. Software, Vol. **10**, No. 4, pp 361–366, 1984.

[3] Blomquist, F.: *Fehlerkalkül mit Hilfe eines Rechners.* Manuskript, 1996.

[4] Blomquist, F., Krämer, W.: *Die Fehler- und komplementäre Fehlerfunktion für Intervallargumente im IEEE-double Format*, Preprint 97/2 des IWRMM, Universität Karlsruhe, 1997.

[5] Borwein, J.M., Borwein, P.B.: *The arithmetic-geometric mean and fast computation of elementary functions*, SIAM Review 26, 1984, 351-366.

[6] Braune, K.: *Hochgenaue Standardfunktionen für reelle und komplexe Punkte und Intervalle in beliebigen Gleitpunktrastern.* Dissertation, Universität Karlsruhe, 1987.

[7] Hart, J. F. et al.: *Computer Approximations.* Wiley, New York / London / Sydney, 1968. Springer–Verlag, Berlin / Heidelberg / New York, 1993.

[8] Hofschuster, W., Krämer, W.: *Ein rechnergestützter Fehlerkalkül mit Anwendungen auf ein genaues Tabellenverfahren*, Preprint 96/5 des IWRMM, Universität Karlsruhe, 1996.

[9] Hofschuster, W., Krämer, W.: *A computer Oriented Approach to Get Sharp Reliable Error Bounds*, to appear in: Reliable Computing, Issue 3, Volume 3, 1997.

[10] American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic.* ANSI/IEEE Std. 754-1985, New York, 1985 (reprinted in SIGPLAN **22**, 2, pp 9–25, 1987).

[11] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: *PASCAL–XSC — Language Reference with Examples.* Springer-Verlag, Berlin/Heidelberg/New York, 1992.

[12] Krämer, W.: *Inverse Standardfunktionen für reelle und komplexe Intervallargumente mit a priori Fehlerabschätzungen für beliebige Datenformate.* Dissertation, Universität Karlsruhe, 1987

[13] Krämer, W.: *Fehlerschranken für häufig auftretende Approximationsausdrücke.* ZAMM **69**, pp T44–T47, 1989.

[14] Krämer, W.: *Multiple-Precision Computations with Result Verification*, in: *Scientific Computing with Automatic Result Verification*, Adams, E., Kulisch, U.(editors), Academic Press, pp. 311–343, 1992.

[15] Tang, P. T. P.: *Table-Driven Implementation of the Exponential Function in IEEE Floating-Point Arithmetic.* ACM Trans. on Math. Software, Vol. **15**, No. 2, pp 144–157, 1989.

[16] Ziv A.: *Fast Evaluation of Elementary Mathematical Functions with Correctly Rounded Last Bit.* ACM Trans. on Math. Software, Vol. 17, N0. 3, pp 410-423, 1991.

In dieser Reihe sind bisher die folgenden Arbeiten erschienen:

1/1996 Ulrich Kulisch: *Memorandum über Computer, Arithmetik und Numerik.*

2/1996 Andreas Wiethoff: *C–XSC — A C++ Class Library for Extended Scientific Computing.*

3/1996 Walter Krämer: *Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen.*

4/1996 Dietmar Ratz: *An Optimized Interval Slope Arithmetic and its Application.*

5/1996 Dietmar Ratz: *Inclusion Isotone Extended Interval Arithmetic.*

1/1997 Astrid Goos, Dietmar Ratz: *Praktische Realisierung und Test eines Verifikationsverfahrens zur Lösung globaler Optimierungsprobleme mit Ungleichungsnebenbedingungen.*

2/1997 Stefan Herbort, Dietmar Ratz: *Improving the Efficiency of a Nonlinear-System-Solver Using a Componentwise Newton Method.*

3/1997 Ulrich Kulisch: *Die fünfte Gleitkommaoperation für top-performance Computer — oder — Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik.*

4/1997 Ulrich Kulisch: *The Fifth Floating-Point Operation for Top-Performance Computers — or — Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic.*

5/1997 Walter Krämer: *Eine Fehlerfaktorarithmetik für zuverlässige a priori Fehlerabschätzungen.*