Bergische Universität
Wuppertal

Sichere a priori Abschätzungen und Realisierung der Funktion $\sqrt{x^2-1}$ in C-XSC

Frithjof Blomquist, Werner Hofschuster, Walter Krämer

Preprint 2004/5

Wissenschaftliches Rechnen/
Softwaretechnologie

Impressum

Herausgeber:	Prof. Dr. W. Krämer, Dr. W. Hofschuster Wissenschaftliches Rechnen/Softwaretechnologie Fachbereich C (Mathematik und Naturwissenschaften) Bergische Universität Wuppertal Gaußstr. 20 D-42097 Wuppertal

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über die World Wide Web Seiten

`http://www.math.uni-wuppertal.de/wrswt/literatur.html`

Autoren-Kontaktadressen

Frithjof Blomquist
Adlerweg 6
D-66436 Püttlingen

E-mail: `blomquist@math.uni-wuppertal.de`

Werner Hofschuster
Bergische Universität Wuppertal
Gaußstr. 20
D-42097 Wuppertal

E-mail: `hofschuster@math.uni-wuppertal.de`

Walter Krämer
Bergische Universität Wuppertal
Gaußstr. 20
D-42097 Wuppertal

E-mail: `kraemer@math.uni-wuppertal.de`

Sichere a priori Abschätzungen und Realisierung der Funktion $\sqrt{x^2-1}$ in C-XSC

Frithjof Blomquist, Werner Hofschuster, Walter Krämer

Inhaltsverzeichnis

1 Zusammenfassung

In dieser Arbeit wird für die Funktion $f(x) = \sqrt{x^2 - 1}$ ein Algorithmus angegeben, mit dem zu vorgegebenen Maschinenargumenten $\widetilde{x}$ die i.a. fehlerbehafteten Maschinenwerte $\widetilde{f}(\widetilde{x})$ nahezu hochgenau berechnet werden. Nach geeigneter Umformung des Arguments $\widetilde{x}^2 - 1$ wird zunächst mit der in C-XSC implementierten Wurzelfunktion eine nullte Näherung $\widetilde{y}_0$ bestimmt, die mit einem nachfolgenden Newtonschritt weiter verbessert werden kann. Nach einer geeigneten Zerlegung von $\widetilde{y}_0$ in zwei Summanden wird dieser Newtonschritt in doppelter Genauigkeit simuliert. Die verlangte hohe Genauigkeit erreicht man dabei nur, wenn bei den auftretenden Additionen zu einem exakten ersten Summanden ein betragsmäß kleiner Summand addiert wird, der nur mit einem möglichst kleinen Fehler behaftet ist. Um dies zu erreichen, müssen in vier verschiedenen Teilbereichen des Definitionsbereiches geeignet angepasste Funktionsterme gewählt werden. Durch geeignete C-XSC Programme kann die Berechnung der erforderlichen a-priori Fehlerschranken in jedem Teilbereich automatisiert werden. Mit den gefundenen Fehlerschranken wird die Funktion zur Ergänzung der Verifikationsbibliothek C-XSC für Punkt- und Intervall-Argumente implementiert. Am Ende der Arbeit findet man einige numerische Beispiele und den Quelltext für Punkt- und Intervallargumente.

Die in dieser Arbeit dargestellten Methoden sowie die zur automatischen Fehlerabschätzung verwendeten Hilfsprogramme werden auch in [2] ausführlich besprochen. Die zugehörigen Programme stehen im Netz unter

`http://www.math.uni-wuppertal.de/wrswt/literatur/a_priori.tgz`

zur Verfügung.

Keywords: C-XSC, a priori Fehlerschranken, genaue Funktionsapproximationen, Einschließungsmethoden.

MSC (2000): 65G20, 65Y15

2 Einleitung

Für Algorithmen aus dem Bereich der Numerik mit Ergebnisverifikation werden so genannte Intervallfunktionen benötigt, deren Ergebnisintervall den exakten Wertebereich der betrachteten Funktion über Intervallargumenten mit Sicherheit einschließt. Die berechnete Einschließung soll dabei möglichst eng sein.

Zur Implementierung solcher Funktionen sind sichere a-priori Abschätzungen der Approximationsfehler in den verschiedenen Teilbereichen sowie a-priori worst-case Fehlerabschätzungen der durch Rundungsfehler verursachten Ungenauigkeiten notwendig. Für die Standardfunktionen wurden dazu spezielle Algorithmen mit Tabellenverfahren entwickelt, mit denen die Rundungsfehler bei den Polynomauswertungen auf ein Minimum reduziert werden konnten [6, 8, 9, 16, 17, 18].

Die genannten Tabellenverfahren und die damit oft gekoppelte Simulation einer doppelten Genauigkeit lassen sich in vielen Fällen auch dann anwenden, wenn Standardfunktionen für spezielle Terme auszuwerten sind [7]. Bei der Simulation einer doppelten

Genauigkeit wird ein Ergebnis als Summe zweier *double*-Zahlen y_0, η_N dargestellt, wobei y_0 als exakt angenommen wird und der betragsmäßig sehr kleine zweite Summand η_N mit nur einem möglichst kleinen Fehler behaftet sein darf. Bildet man dann auf der Maschine die Summe $y_0 \oplus \eta_N$, so erhält man einen relativen Fehler, der nur minimal größer ist als der relative Fehler der Maschinenaddition [2].

In dieser Arbeit wird gezeigt, wie diese Methode bei der Implementierung der Funktion $f(x) := \sqrt{x^2 - 1}$ erfolgreich angewendet werden kann. Dabei ist zu beachten, dass x als rundungsfehlerfreie Maschinenzahl zu betrachten ist, wobei man sich auf $x \geq 1$ beschränken kann. y_0 ist eine mit der in C-XSC implentierten Wurzelfunktion berechnete gute Näherung für $f(x)$, und η_N ist eine mit dem Newtonverfahren erhaltene sehr kleine Korrektur, die nach der beschriebenen Methode in hoher Genauigkeit bestimmt werden kann.

Die automatische und damit sichere numerische Berechnung der Rundungsfehlerschranken erfolgt mit C-XSC Programmen, die auch in [2] vollständig angegeben sind. Die in diesen Programmen benutzten Werkzeuge werden ebenfalls in [2] ausführlich beschrieben. Numerische Beispiele am Ende dieser Arbeit dokumentieren die Zuverlässigkeit und die hohe Genauigkeit der berechneten Funktionswerteinschließungen.

3 Zur Notation

$\mathbb{R}$	Menge der reellen Zahlen
IR	Menge der reellen Maschinenintervalle
$S(b,l,\underline{e},\overline{e}) = S(b,l)$	Gleitkommaraster mit Basis b, Mantissenlänge l und Exponent e, mit: $\underline{e} \le e \le \overline{e},\ \ e \in \mathbb{Z}$
$S(2,53,-1022,+1023) = S(2,53)$	IEEE-Datenformat *double*
x, y, y_0, y_1	reelle Größen; $x, y, y_0, y_1 \in \mathbb{R}$
$\widetilde{x}, \widetilde{y}, \widetilde{y}_0, \widetilde{y}_1$	Maschinenzahlen $\widetilde{x}, \widetilde{y}, \widetilde{y}_0, \widetilde{y}_1 \in S(2,53)$, die auch fehlerbehaftet sein können[1]
$\boldsymbol{x}$, $\mathtt{x}, \mathtt{W}, X, Y \in IR$	Maschinenintervalle vom Typ *interval*
$\circ \in \{+,-,\bullet,/\}$	exakte reelle Operatoren
$\{\oplus, \ominus, \odot, \oslash\}$	Gleitkomma-Operatoren, deren Rundung vom jeweils eingestellten Rundungsmodus abhängt
$\mathtt{Minreal} = 2^{-1022} = 2.225.. \cdot 10^{-308}$	kleinste positive, normalisierte *double* Zahl
$\mathtt{minreal} = 2^{-1074} = 4.940.. \cdot 10^{-324}$	kleinste positive, denormalisierte *double* Zahl
$\mathtt{Maxreal} < 2^{+1024} = 1.797.. \cdot 10^{+308}$	größte, normalisierte *double* Zahl
$\mathtt{Minreal} \le \lvert\widetilde{x}\rvert \le \mathtt{Maxreal}$	normalisierter Bereich im *double*-Format
$\mathtt{succ}(\widetilde{x})$	nächstgrößere Maschinenzahl bezüglich $\widetilde{x}$
$f(x) = \sqrt{x^2-1}$	die in diesem Preprint behandelte Funktion
$\widetilde{f}(\widetilde{x}) \approx f(\widetilde{x})$	$\widetilde{f}(\widetilde{x})$: auf der Maschine ausgewerteter i.a. fehlerbehafteter Funktionswert
$\mathtt{sqrt}(\widetilde{x}) \approx \sqrt{\widetilde{x}}$	$\mathtt{sqrt}(\widetilde{x})$ ist eine C-XSC Standardfunktion
$a := \widetilde{x}^2 - 1 \in \mathbb{R}$	exaktes Argument der Wurzelfunktion
ε_0	relativer Fehler der nullten Maschinennäherung $\widetilde{y}_0 = \sqrt{\widetilde{x}^2-1} \cdot (1+\varepsilon_0),\ \lvert\varepsilon_0\rvert \le \varepsilon(0)$
$\widetilde{y}_0 = s_1 + s_2$	Zerlegung in die Summanden $s_1, s_2 \in S(2,53)$
$y_1 \in \mathbb{R}$	exaktes Ergebnis nach Newtonkorrektur
$\widetilde{y}_1 \in S(2,53), \quad \widetilde{y}_1 \approx \widetilde{y}_0$	$\widetilde{y}_1$: Maschinenergebnis nach Newtonkorrektur
$\varepsilon(f, A_i),\ i = 1,2,3,4;$	relative Fehlerschranke bei Auswertung von $f(\widetilde{x}) = \sqrt{\widetilde{x}^2-1}$ über dem Teilbereich A_i
$\varepsilon(app)$	relativer Approximationsfehler
$\varepsilon(g)$	relativer Auswertefehler bezüglich $g(x)$
$\varepsilon(h) = 2^{-52} = 2.220446\ldots \cdot 10^{-16}$	relative Fehlerschranke der Grundoperationen bei nur hochgenauer Arithmetik.
$W = \{y \in \mathbb{R} \,\vert\, y = f(x),\ x \in \boldsymbol{x}\}$	Wertebereich von $f(x)$ über dem Intervall $\boldsymbol{x}$

[1] Berechnete Fehlerschranken von Funktionen $f(x)$ beziehen sich nur auf die Maschinenargumente $\widetilde{x}$ eines vorgegebenen Maschinenintervalls, wobei die $\widetilde{x}$ stets als rundungsfehlerfrei angenommen werden.

4 Aufgabenstellung

Wir betrachten die reelle Funktion

$$f : D_f = \left\{ x \in I\!R \,\middle|\, |x| \geq 1 \right\} \rightarrow I\!R, \quad \text{mit} \quad f(x) = \sqrt{x^2 - 1}$$

Für alle Maschinenzahlen[1] $\widetilde{x} \geq \texttt{succ}(1.0)$ ist eine garantierte Oberschranke $\varepsilon(f)$ des relativen Fehlers

$$\left| \frac{\sqrt{\widetilde{x}^2 - 1} - \widetilde{f}(\widetilde{x})}{\sqrt{\widetilde{x}^2 - 1}} \right| \leq \varepsilon(f), \quad \widetilde{x} \in S(2, 53), \quad \widetilde{x} \geq \texttt{succ}(1.0); \tag{1}$$

zu berechnen, wobei $\widetilde{f}(\widetilde{x})$ die auf der Maschine berechnete Näherung für den exakten Funktionswert $f(\widetilde{x})$ ist. Mit Hilfe einer solchen Schranke lässt sich dann zu einem vorgegebenen Maschinenintervall $X \in IR$ eine mathematisch garantierte und nahezu optimale Einschließung $Y \in IR$ aller zugehörigen Funktionswerte y berechnen:

$$\left\{ y \in I\!R \,\middle|\, y = \sqrt{x^2 - 1},\ x \in X \right\} \subseteq Y$$

Beachten Sie bitte, dass wir mit Y eine Einschließung der Funktionswerte $y = f(x)$ auch für diejenigen Argumente $x \in X$ erhalten, die keine Maschinenargumente sind, obwohl sich die Fehlerschranke $\varepsilon(f)$ nur auf die Maschinenargumente $\widetilde{x} \in S(2, 53)$ bezieht, da die Funktion f auf der Maschine nur für diese Argumente ausgewertet werden kann!

5 Punktargumente

Wir werden zunächst für alle Maschinenzahlen $\widetilde{x} \in D_f$ die obige relative Fehlerschranke $\varepsilon(f)$ bestimmen. Wegen der Symmetrie $f(-x) \equiv f(x)$ kann man sich dabei auf den Bereich $x \geq 1$ beschränken. Zu einem vorgegebenen Intervallargument $\boldsymbol{x} \in IR$ und mit Hilfe der Monotonie der Funktion f läßt sich dann eine garantierte und nahezu optimale Einschließung des Wertebereichs $W = \{ y \in I\!R \,|\, y = \sqrt{x^2 - 1},\ x \in \boldsymbol{x} \}$ berechnen.

5.1 Algorithmus

Für die Maschinenargumente $\widetilde{x} \in [\texttt{succ}(1.0), \texttt{MaxReal}]$ betrachten wir die folgenden vier Teilbereiche:

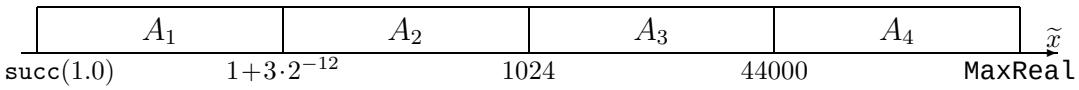

Abbildung 1: Teilintervalle A_i der Maschinenzahlen $\widetilde{x} \in S(2, 53)$

[1] Da $f(1) = 0$ auf der Maschine exakt ausgewertet wird, kann man sich bei der Fehlerabschätzung auf $\widetilde{x} \geq \texttt{succ}(1.0)$ beschränken.

In A_1, A_2, A_3 wird nach einer jeweils geeigneter Auswertung der Differenz $\widetilde{x}^2 - 1$ mit $\widetilde{y}_0 \approx \sqrt{\widetilde{x}^2 - 1}$ zunächst eine nullte Näherung berechnet. Eine weitere Verbesserung dieser Näherung $\widetilde{y}_0$ erhält man dann in einem nachfolgenden Newton-Schritt:

$$y_1 := \frac{1}{2} \cdot \left(\widetilde{y}_0 + \frac{\widetilde{x}^2 - 1}{\widetilde{y}_0} \right) \equiv \widetilde{y}_0 + \frac{1}{2} \cdot \frac{(\widetilde{x}^2 - 1) - \widetilde{y}_0^2}{\widetilde{y}_0} =: g(\widetilde{x}) \tag{2}$$

Eine wirkliche Verbesserung der Genauigkeit erhält man jedoch nur[2], wenn man in (2) bei der Auswertung des rechten Terms $g(\widetilde{x})$ darauf achtet, dass der Auswertefehler des zweiten Summanden mit dem Faktor $1/2$ hinreichend klein bleibt. In diesem Fall ist dann auf der Maschine eine Summe auszuwerten, deren erster Summand $\widetilde{y}_0$ als exakt aufzufassen ist und deren betragsmäßig sehr viel kleinerer zweiter Summand mit nur einem kleinen Fehler behaftet ist. Der durch die Gleichung

$$\widetilde{g}(\widetilde{x}) = g(\widetilde{x}) \cdot (1 + \varepsilon_g), \quad |\varepsilon_g| \leq \varepsilon(g)$$

definierte relative Auswertefehler ε_g ist dann fast optimal und betragsmäßig nur etwas größer als die Fehlerschranke der Grundoperationen bei vorausgesetzter hochgenauer Arithmetik: $\varepsilon(h) = 2^{-52} = 2.220446\ldots \cdot 10^{-16}$.

Bezeichnet man mit $a := \widetilde{x}^2 - 1$ das Argument der Wurzelfunktion und ist dabei $\widetilde{a} := \widetilde{x} \odot \widetilde{x} \ominus 1$ das auf der Maschine ausgewertete Argument, so ist der absolute Fehler δ_a und der relative Fehler ε_0 bei Auswertung der Wurzelfunktion definiert durch:

$$\begin{aligned} \widetilde{a} &= (\widetilde{x}^2 - 1) + \delta_a, \quad |\delta_a| \leq \Delta(a) \\ \widetilde{y}_0 := \texttt{sqrt}(\widetilde{a}) &= \sqrt{\widetilde{x}^2 - 1} \cdot (1 + \varepsilon_0), \quad |\varepsilon_0| \leq \varepsilon(0), \end{aligned} \tag{3}$$

wobei $\texttt{sqrt}(\ldots)$ die in C-XSC definierte Wurzelfunktion ist. Die Berechnung der relativen Fehlerschranke $\varepsilon(0)$ erfolgt dabei mit Hilfe der Funktion `rel_sqrt_abs(...)`, bei der berücksichtigt wird, dass das Argument $\widetilde{a}$ der Wurzelfunktion selbst mit einem absoluten Fehler behaftet ist [2]. Der Newtonschritt in (2) liefert mit (3) nach einigen Rechnungen:

$$\begin{aligned} y_1 &= \frac{1}{2} \cdot \left(\widetilde{y}_0 + \frac{a}{\widetilde{y}_0} \right) = \frac{1}{2} \cdot \left\{ \sqrt{a} \cdot (1 + \varepsilon_0) + \frac{a}{\sqrt{a} \cdot (1 + \varepsilon_0)} \right\} \in \mathbb{IR} \\ &= \sqrt{a} \cdot \left\{ 1 + \frac{\varepsilon_0^2}{2 \cdot (1 + \varepsilon_0)} \right\} = \sqrt{a} \cdot (1 + \varepsilon_{app}), \quad \varepsilon_{app} = \frac{\varepsilon_0^2}{2 \cdot (1 + \varepsilon_0)} \end{aligned} \tag{4}$$

Eine Oberschranke $\varepsilon(app)$ des relativen Approximationsfehlers erhält man dann durch Intervallauswertung des obigen Quotienten für ε_{app}, wobei gilt: $|\varepsilon_0| \leq \varepsilon(0)$. Ist dann $\widetilde{y}_1$ die auf der Maschine berechnete Näherung für y_1, so erhält man für den durch $\widetilde{y}_1 = \widetilde{g}(\widetilde{x}) = \sqrt{\widetilde{x}^2 - 1} \cdot (1 + \varepsilon_f)$ definierten relativen Gesamtfehler ε_f die Oberschranke, [2]:

$$\varepsilon(f) = \varepsilon(app) + \varepsilon(g) \cdot [1 + \varepsilon(app)], \quad |\varepsilon_f| \leq \varepsilon(f); \tag{5}$$

[2]Wertet man in (2) den ersten Term aus, so erhält man einen relativen Fehler $4.44\ldots \cdot 10^{-16}$, der sich bei geschickter Auswertung des rechten Terms $g(\widetilde{x})$ noch halbieren lässt.

Im Bereich $A_4 = [44000, \texttt{MaxReal}]$ wird $f(x) = \sqrt{x^2-1}$ approximiert durch:

$$\sqrt{x^2-1} \approx g_4(x) := x - \frac{1}{2x}, \quad x \geq 44000 \tag{6}$$

Auch in diesem Fall wird die Auswertung von $g_4(x)$ nur einen sehr kleinen relativen Fehler verursachen, da von einem exakten Summanden x ein betragsmäßig sehr kleiner fehlerbehafteter Wert zu subtrahieren ist. Die Oberschranke $\varepsilon(g_4)$ dieses relativen Auswertefehlers wird daher nur minimal größer sein als die Fehlerschranke der Grundoperationen $\varepsilon(h) := 2^{-52}$ bei nur hochgenauer Arithmetik. Der auf der Maschine ausgewertete Term $\widetilde{g}_4(\widetilde{x})$ lautet:

$$\widetilde{g}_4(\widetilde{x}) := \widetilde{x} \ominus 0.5 \odot (1 \oslash \widetilde{x})$$

und der relative Auswertefehler ε_{g_4} ist definiert durch:

$$\widetilde{g}_4(\widetilde{x}) = g_4(\widetilde{x}) \cdot (1 + \varepsilon_{g_4}), \quad |\varepsilon_{g_4}| \leq \varepsilon(g_4);$$

Der relative Approximationsfehler ε_{app} ist gegeben durch $g_4(x) = \sqrt{x^2-1} \cdot (1 + \varepsilon_{app})$ und kann wie folgt abgeschätzt werden:

$$\begin{aligned}
|\varepsilon_{app}| &= \left| \frac{\sqrt{x^2-1} - \left(x - \frac{1}{2x}\right)}{\sqrt{x^2-1}} \right| \\
&= \left| \frac{\left[\sqrt{x^2-1} - \left(x - \frac{1}{2x}\right)\right] \cdot \left[\sqrt{x^2-1} + \left(x - \frac{1}{2x}\right)\right]}{\sqrt{x^2-1} \cdot \left[\sqrt{x^2-1} + \left(x - \frac{1}{2x}\right)\right]} \right| \\
&= \frac{\frac{1}{4x^2}}{\sqrt{x^2-1} \cdot \left[\sqrt{x^2-1} + \left(x - \frac{1}{2x}\right)\right]} \\
|\varepsilon_{app}| &< \frac{1}{4x^2 \cdot (x^2-1)} =: \varepsilon(app) \qquad (7)
\end{aligned}$$

Der relative Gesamtfehler ε_{f_4} ist definiert durch

$$\widetilde{g}_4(\widetilde{x}) = \sqrt{\widetilde{x}^2-1} \cdot (1 + \varepsilon_{f_4}), \quad |\varepsilon_{f_4}| \leq \varepsilon(f_4),$$

und für die Oberschranke $\varepsilon(f_4)$ gilt wieder analog zu (5) die bekannte Beziehung:

$$\varepsilon(f_4) = \varepsilon(app) + \varepsilon(g_4) \cdot [1 + \varepsilon(app)]; \tag{8}$$

Die Berechnung der Fehlerschranke $\varepsilon(f_4)$ erfolgt mit dem Programm `sqrtx2m1_A4` auf Seite 19.

5.2 Fehlerabschätzung

In den folgenden Unterabschnitten wird für jeden Teilbereich A_i die in (1) definierte Oberschranke des relativen Fehlers berechnet.

5.2.1 Fehlerabschätzung in $A_1 = [\mathbf{succ(1.0)}, \mathbf{1 + 3 \cdot 2^{-12}}]$

Bei der direkten Auswertung des Arguments $a = \widetilde{x}^2 - 1 \approx \widetilde{x} \odot \widetilde{x} \ominus 1$ entsteht wegen der bekannten Auslöschungseffekte ein relativer Fehler in der Größenordnung $1/2$, so dass damit die Wurzelfunktion extrem fehlerhaft berechnet wird. Abhilfe schafft jetzt die folgende Zerlegung von x:

$$\widetilde{x} = 1 + \delta, \text{ mit: } \quad \delta = \widetilde{x} - 1, \quad 0 < \delta \ll 1 \quad \text{und} \quad \widetilde{x}^2 - 1 = 2 \cdot \delta + \delta^2,$$

wobei die Differenz $\delta = \widetilde{x} - 1 = \widetilde{x} \ominus 1$ und damit auch $2 \cdot \delta$ jeweils exakt berechnet wird. Für die Fehlerabschätzung ist die gefundene Zerlegung

$$\widetilde{x}^2 - 1 = 2 \cdot \delta + \delta^2 =: a, \quad \widetilde{x}, \delta \in S(2, 53), \quad a \in I\!R; \tag{9}$$

jetzt wieder ideal, da zum exakten Summanden $2 \cdot \delta$ ein zwar fehlerbehafteter aber vergleichsweise nur kleiner Summand $\delta^2 \approx \delta \odot \delta \ll 2 \cdot \delta$ zu addieren ist. Bezeichnet man mit $\widetilde{a} = 2 \cdot \delta \oplus \delta \odot \delta$ wieder das Maschinenergebnis des Arguments a, so ist der durch $\widetilde{a} = a \cdot (1 + \varepsilon_a)$ definierte relative Fehler ε_a betragsmäßig wieder nur minimal größer als $\varepsilon(h) = 2^{-52}$. Wertet man nun mit dem fehlerbehafteten Argument $\widetilde{a}$ die selbst wieder fehlerbehaftete Wurzelfunktion `sqrt(...)` des C-XSC Systems aus, so erhält man für den durch $\widetilde{y}_0 := \texttt{sqrt}(\widetilde{a}) = \sqrt{\widetilde{x}^2 - 1} \cdot (1 + \varepsilon_0)$ definierten relativen Fehler ε_0 die Oberschranke $|\varepsilon_0| \leq \varepsilon(0) = 3.33\ldots \cdot 10^{-16}$. Im Vergleich zur Fehlerschranke $\varepsilon(h) = 2^{-52} = 2.220\ldots \cdot 10^{-16}$ der Grundoperationen haben wir damit eine schon recht kleine Fehlerschranke, die sich jedoch mit nur etwas Mehraufwand noch weiter verbessern lässt.

Führt man mit dem Startwert $\widetilde{y}_0$ nach (2) von Seite 8 einen Newtonschritt durch, so erhält man nach (4) die neue Näherung:

$$\begin{aligned} y_1 &= g_1(\delta) := \widetilde{y}_0 + \frac{1}{2} \cdot \frac{a - \widetilde{y}_0^2}{\widetilde{y}_0}, \quad a := \widetilde{x}^2 - 1 = 2 \cdot \delta + \delta^2, \\ &= \sqrt{a} \cdot (1 + \varepsilon_{app}), \quad \varepsilon_{app} = \frac{\varepsilon_0^2}{2 \cdot (1 + \varepsilon_0)}, \quad |\varepsilon_0| \leq \varepsilon(0), \end{aligned} \tag{10}$$

und eine wirkliche Verbesserung der Fehlerschranke wird man nur erhalten, wenn man den Auswertefehler von $g_1(\delta)$ hinreichend klein hält. Wertet man jetzt $g_1(\delta)$ auf der Maschine aus durch

$$\widetilde{g}_1(\delta) := \widetilde{y}_0 \oplus 0.5 \odot [(2 \cdot \delta \oplus \delta \odot \delta) - \widetilde{y}_0 \odot \widetilde{y}_0] \oslash \widetilde{y}_0 \in S(2, 53), \tag{11}$$

so erhält man für den durch

$$\widetilde{g}_1(\delta) := g_1(\delta) \cdot (1 + \varepsilon_{g_1}) \in S(2, 53)$$

definierten relativen Auswertefehler e_{g_1} die viel zu große Oberschranke

$$|\varepsilon_{g_1}| \leq \varepsilon(g_1) := 9.242608 \cdot 10^{-16},$$

d.h. man erhält bei der Durchführung eines zusätzlichen Newton-Schritts bei der Auswertung von $g_1(\delta)$ nach (11) einen im Vergleich zu $\varepsilon(0) = 3.33\ldots \cdot 10^{-16}$ sehr viel größeren Auswertefehler. Der Grund hierfür ist jedoch offensichtlich:

- Wegen der kleinen Fehlerschranke $\varepsilon(0) = 3.33\ldots\cdot 10^{-16}$ ist der Startwert $\widetilde{y}_0$ schon so gut, dass das Newton-Verfahren quadratisch konvergiert.
- Daher muss der zweite Summand in (10) mit dem Faktor $1/2$ als Korrekturglied im Vergleich zu $\widetilde{y}_0$ sehr klein ausfallen, so dass bei der Auswertung der Differenz $[(2 \cdot \delta \oplus \delta \odot \delta) - \widetilde{y}_0 \odot \widetilde{y}_0]$ nach (11) starke Auslöschung auftreten muss, die den großen relativen Fehler $\varepsilon(g_1)$ verursacht.

Das Ziel muss daher sein, die Differenz $a - \widetilde{y}_0^2 = (2 \cdot \delta + \delta^2) - \widetilde{y}_0^2$ auf der Maschine ohne Auslöschung zu berechnen! Dazu zerlegen wir zunächst den Startwert $\widetilde{y}_0$ mit Hilfe der folgenden Anweisungen rundungsfehlerfrei in zwei `double`-Zahlen s_1, s_2:

$$s_1 = \texttt{Cut26}(\widetilde{y}_0), \quad s_2 = \widetilde{y}_0 - s_1 \quad \Longrightarrow \quad \widetilde{y}_0 = s_1 + s_2 = s_1 \oplus s_2, \quad s_1, s_2 \in S(2, 53);$$

Die C-XSC Funktion `Cut26` lässt die ersten 26 Mantissenbits unverändert und setzt die restlichen $53 - 26 = 27$ Bits auf Null, so dass das Produkt $s_1 \cdot s_1$ mit seinen maximal $2 \cdot 26 = 52 < 53$ von Null verschiedenen Mantissenbits auf der Maschine stets exakt ausgewertet wird. Die Differenz $a - \widetilde{y}_0^2$ kann jetzt geschrieben werden als:

$$a - \widetilde{y}_0^2 = (2 \cdot \delta + \delta^2) - \widetilde{y}_0^2 = (2 \cdot \delta - s_1^2) + [\delta^2 - s_2 \cdot (\widetilde{y}_0 + s_1)] \tag{12}$$

und wegen $2 \cdot \delta \approx a \approx \widetilde{y}_0^2 \approx s_1^2$ folgt damit $a - \widetilde{y}_0^2 \approx (2 \cdot \delta - s_1^2)$, so dass diese Differenz rundungsfehlerfrei berechnet werden kann[3]:

$$2 \cdot \delta - s_1^2 \equiv 2 \odot \delta \ominus s_1 \odot s_1$$

In (12) haben wir damit den für die Fehlerabschätzung gewünschten Idealfall, dass zu einem rundungsfehlerfreien ersten Summanden $(...)$ ein fehlerbehafteter aber betragsmäßig kleiner Summand $[....]$ zu addieren ist, so dass jetzt ein nur kleiner Auswertefehler für den vollständigen Term

$$\begin{aligned} \widetilde{g}_1(\delta) &:= \widetilde{y}_0 \oplus 0.5 \cdot [(2 \cdot \delta - s_1^2) \oplus \{\delta \odot \delta \ominus s_1 \odot (\widetilde{y}_0 \oplus s_1)\}] \oslash \widetilde{y}_0 \\ &= g_1(\delta) \cdot (1 + \varepsilon_{g_1}), \quad |\varepsilon_{g_1}| \leq \varepsilon(g_1) \end{aligned}$$

zur Berechnung von $\widetilde{y}_1 = \widetilde{g}_1(\delta)$ zu erwarten ist. Die relative Gesamtfehlerschranke ε_f ist definiert durch:

$$\begin{aligned} \widetilde{y}_1 &= \widetilde{g}_1(\delta) = g_1(\delta) \cdot (1 + \varepsilon_{g_1}) = \sqrt{a} \cdot (1 + \varepsilon_{app}) \cdot (1 + \varepsilon_{g_1}) \\ &= \sqrt{a} \cdot (1 + \varepsilon_f), \quad |\varepsilon_{app}| \leq \varepsilon(app), \quad |\varepsilon_{g_1}| \leq \varepsilon(g_1), \end{aligned} \tag{13}$$

und für $|\varepsilon_f|$ berechnet sich dann eine Oberschranke $\varepsilon(f)$ nach (5) zu:

$$|\varepsilon_f| \leq \varepsilon(f) := \varepsilon(app) + \varepsilon(g_1) \cdot [1 + \varepsilon(app)]; \tag{14}$$

Die relativen Fehlerschranken $e(app),\ \varepsilon(g_1)$ und damit $\varepsilon(f)$ werden berechnet mit dem folgenden Programm `sqrtx2m1_A1.cpp`

[3] Der erfahrene Numeriker wird dieses Ergebnis wegen $2 \cdot \delta \approx s_1^2$ zwar sofort akzeptieren, der eigentliche Beweis ergibt sich jedoch erst durch die gesicherte Fehlerabschätzung mit dem Programm `sqrtx2m1_A1.cpp` von Seite 11.

```
// Programm:  sqrtx2m1_A1.cpp;
// Berechnung der rel. Fehlerschranke eps(f) in A1:
// x = 1 + DEL;  --->  x^2-1 = 2*DEL + DEL^2;
// y0 = s1+s2; y0 = sqrt(x^2-1) = sqrt(2*DEL+DEL^2) ist Startwert
// x - y0^2 = (2*DEL - s1^2) + [DEL^2 - s2*(y0 + s1)]

#include "abs_relh.hpp" // Wegen abs_mulh1()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include "hilfe.hpp"    // Wegen CuT26()
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;

real T_x(const interval& Di, const real& delx)
{   // Di: Teilintervall von DEL
    interval Y0,B,C,D,E,S1,S2,G,H;
    real s1,s2,rel,app,delb,delc,dele,delh;
    int ex;

    G=Di;
    times2pown(G,1);  // G = 2*Di
    abs_mulh1(Di,delx,Di,delx,H,delh);  // H = Di*Di
    abs_addh1(G,0,H,delh,C,delc);  // C = 2*Di + Di*Di
                 // delc: abs. Fehlerschranke bez. 2*Di + Di*Di
    rel_sqrt_abs(C,delc,Y0,app);  // Y0 = sqrt(2*Di + Di*Di)
    // app: relative Fehlerschranke fuer den Startwert.
    D = interval(-app,+app);
    D = 0.5 * D*D/(1+D);
    app = Sup(D); // app: Schranke des rel. Approximationsfehlers

    s1 = Cut26(Inf(Y0));  s2 = Cut26(Sup(Y0));
    S1 = interval(s1,s2); // y0 = s1+s2; s1 = Cut26(y0)
    ex = expo(Sup(Y0));
    s1 = comp(0.5,ex-25);
    S2 = interval(0,s1);  // s1 in S1 und s2 in S2 enthalten!!

    B = S1*S1;  // s1*s1 wird stets rundungsfehlerfrei berechnet!
    abs_subh1(G,0.0,B,0.0,D,s1); // D = 2*Di-S1*S1; s1=0 erfuellt
    abs_addh1(Y0,0.0,S1,0.0,B,delb); // B = y0 + s1
    abs_mulh1(B,delb,S2,0.0,C,delc); // C = s2*(y0 + s1)
    abs_subh1(H,delh,C,delc,B,delb); // B = Di*Di - s2*(y0 + s1)
    abs_addh1(D,s1,B,delb,E,dele);   // E = (2Di + Di*Di) - y0^2
    abs_divh1(E,dele,Y0,0.0,C,delc); // C = [(2Di+Di*Di)-y0^2]/y0
    C = C/2;
    delc = delc/2;
    rel_addh1(Y0,0.0,C,delc,B,rel);   // rel: rel. Auswertefehler
```

```
    // Beruecksichtigung des relativen Approximationsfehlers:
    s1 = addu(1.0,app);
    s2 = mulu(s1,rel);
    rel = addu(app,s2);

    return rel;  // Schranke des relativen Gesamtfehlers.
}

int main()
{
    // 1.000732421875 = 1 + 3*2^(-12) wird exakt gespeichert!
    interval X = interval(succ(1.0),1.000732421875),DEL;
    DEL = X - 1;
    real bnd, diam = 1e-6, delx=0;
    Max_bnd_Xi(T_x,DEL,delx,diam,bnd);
    cout << RndUp << "Rel. Schranke eps(f) = " << bnd << endl;
}
```

Hinweise:

1. Die Funktion `Max_bnd_Xi(T_x,DEL,delx,diam,bnd)` unterteilt das übergebene Intervall `DEL` mittels `diam = 1e-6` in Teilintervalle `Di`, für die die Funktion `T_x(...)` die jeweilige relative Fehlerschranke bestimmt, deren Maximum als Gesamtfehlerschranke $\varepsilon(f) =$ `bnd` von `Max_bnd_Xi(...)` zurückgegeben wird. Weitere Informationen findet man in [2].

2. Für $\delta \in$ `Di` gilt für die entsprechenden Startwerte[4] $\widetilde{y}_0 \in$ `Y0`. Zerlegt man dann $\widetilde{y}_0 =$ `s1`$+$`s2` mittels `s1 = Cut26(y0); s2 = y0 - s1;` so werden die positiven `s1`-Werte eingeschlossen durch `s1` $\in [$`Cut26(Inf(Y0))`$,$ `Cut26(Sup(Y0))`$]$.

3. Für die nichtnegativen `s2`-Werte gilt dann mit `ex = expo(Sup(Y0))` die Einschließung `s2` $\in [0,$ `comp(0.5, ex` $- 25)]$.

Ergebnis:

Für alle Maschinenzahlen $\widetilde{x} \in A_1 = [$`succ`$(1.0), 1+3\cdot 2^{-12}]$ gilt für den in (13) definierten relativen Gesamtfehler ε_f nach (14) die Abschätzung:

(15)
$$\boxed{\left|\frac{\widetilde{y}_1 - \sqrt{\widetilde{x}^2 - 1}}{\sqrt{\widetilde{x}^2 - 1}}\right| \leq \varepsilon(f, A_1) = 2.221261 \cdot 10^{-16}}$$

[4] Wir bezeichnen jetzt die Maschinenzahl $\widetilde{y}_0$ mit `y0`

5.2.2 Fehlerabschätzung in $A_2 = [1 + 3 \cdot 2^{-12}, 1024]$

In diesem Bereich liefert die Zerlegung $\widetilde{x}^2 - 1 = 2 \cdot \delta + \delta^2$ aus A_1 sehr viel größere Auswertefehler, da jetzt $\delta^2 \gg 2 \cdot \delta$ gilt und $\delta \odot \delta$ nicht rundungsfehlerfrei berechnet werden kann. Besser ist jetzt die rundungsfehlerfreie Zerlegung von $\widetilde{x} = x_1 + x_2$ durch die Anweisungen[5]: `x1 = Cut26(x); x2 = x - x1;` und man erhält:

$$a := \widetilde{x}^2 - 1 = (x_1^2 - 1) + x_2 \cdot (x + x_1) \in I\!R, \tag{16}$$

wobei jetzt der erste Summand $(x_1^2 - 1)$ rundungsfehlerfrei berechnet wird. Wegen der Beziehung $(x_1^2 - 1) \gg x_2 \cdot (x + x_1)$ wird dann bei der Auswertung der rechten Seite von (16) der relative Fehler nur wenig größer sein als die relative Schranke der Grundoperationen $\varepsilon(h) = 2^{-52} = 2.220.. \cdot 10^{-16}$ bei nur hochgenauer Arithmetik. Mit der Maschinenzahl

$$\widetilde{a} := (x_1 \odot x_1 \ominus 1) \oplus x_2 \odot (x \oplus x_1) = (x_1 \cdot x_1 - 1) \oplus x_2 \odot (x \oplus x_1)$$

liefert dann die Anweisung $\widetilde{y}_0 = \texttt{sqrt}(\widetilde{a})$ die Maschinenzahl $\widetilde{y}_0 \in S(2, 53)$ als Startwert für den nachfolgenden Newton-Schritt:

$$y_1 := \frac{1}{2} \cdot \left(\widetilde{y}_0 + \frac{\widetilde{x}^2 - 1}{\widetilde{y}_0} \right) \equiv \widetilde{y}_0 + \frac{1}{2} \cdot \frac{(\widetilde{x}^2 - 1) - \widetilde{y}_0^2}{\widetilde{y}_0} =: g_2(\widetilde{x}) \in I\!R \tag{17}$$

Wie im vorherigen Abschnitt müssen wir jetzt darauf achten, dass der obige Zähler $(\widetilde{x}^2 - 1) - \widetilde{y}_0^2$ ohne Auslöschung ausgewertet werden kann. Deshalb zerlegen wir den Startwert $\widetilde{y}_0 = s_1 + s_2$ durch die beiden Anweisungen $s_1 = \texttt{Cut26}(\widetilde{y}_0);\ s_2 = \widetilde{y}_0 - s_1;$ rundungsfehlerfrei in die beiden Summanden s_1, s_2, so dass der obige Zähler jetzt wie folgt geschrieben werden kann:

$$(\widetilde{x}^2 - 1) - \widetilde{y}_0^2 = [(x_1^2 - 1) - s_1^2] + \{x_2 \cdot (\widetilde{x} + x_1) - s_2 \cdot (\widetilde{y}_0 + s_1)\}, \tag{18}$$

wobei der erste Summand $[\,...\,]$ rundungsfehlerfrei berechnet wird und der betragsmäßig kleinere zweite Summand $\{\,...\,\}$ mit einem hinreichend kleinen Fehler behaftet ist. Wertet man jetzt mit (18) die Funktion $g_2(\widetilde{x})$ aus, so erhält man für den in $\widetilde{y}_1 := \widetilde{g}_2(\widetilde{x}) = g_2(\widetilde{x}) \cdot (1 + \varepsilon_{g_2})$ definierten relativen Fehler ε_{g_2} wie im vorhergehenden Abschnitt eine hinreichend kleine Oberschranke $\varepsilon(g_2)$.

Der relative Approximationsfehler ε_0 ist für den Startwert $\widetilde{y}_0$ definiert durch:

$$\widetilde{y}_0 = \texttt{sqrt}(\widetilde{a}) = \sqrt{\widetilde{x}^2 - 1} \cdot (1 + \varepsilon_0), \quad |\varepsilon_0| \leq \varepsilon(0)$$

Nach (10) ist dann der relative Approximationsfehler ε_{app} der verbesserten Näherung y_1 definiert durch:

$$y_1 = g_2(\widetilde{x}) = \sqrt{a} \cdot (1 + \varepsilon_{app}), \quad \varepsilon_{app} = \frac{\varepsilon_0^2}{2 \cdot (1 + \varepsilon_0)} \tag{19}$$

und eine Oberschranke $\varepsilon(app) \geq \varepsilon_{app}$ erhält man wieder durch Intervallauswertung des Quotienten für ε_{app} in (19), wobei $\varepsilon_0 \in [-\varepsilon(0), +\varepsilon(0)]$ zu berücksichtigen ist.

[5] Es gelten jetzt die Bezeichnungen: $\widetilde{x} = \texttt{x}, x_1 = \texttt{x1}$ und $x_2 = \texttt{x2}$.

Die relative Gesamtfehlerschranke ε_f ist jetzt wieder definiert durch:

$$\begin{aligned} \widetilde{y}_1 &= \widetilde{g}_2(\widetilde{x}) = g_2(\widetilde{x}) \cdot (1+\varepsilon_{g_2}) = \sqrt{a} \cdot (1+\varepsilon_{app}) \cdot (1+\varepsilon_{g_2}) \\ &= \sqrt{a} \cdot (1+\varepsilon_f), \quad |\varepsilon_{app}| \leq \varepsilon(app), \quad |\varepsilon_{g_2}| \leq \varepsilon(g_2), \end{aligned} \tag{20}$$

und für $|\varepsilon_f|$ berechnet sich dann wieder eine Oberschranke $\varepsilon(f)$ nach (5) zu:

$$|\varepsilon_f| \leq \varepsilon(f) := \varepsilon(app) + \varepsilon(g_2) \cdot [1+\varepsilon(app)]; \tag{21}$$

Die relativen Fehlerschranken $e(app),\ \varepsilon(g_2)$ und damit $\varepsilon(f)$ werden berechnet mit dem folgenden Programm `sqrtx2m1_A2.cpp`

```
// Programm:  sqrtx2m1_A2.cpp
// Berechnung des relativen Gesamtfehlers
// y0 +0.5*(x-y0^2)/y0;    x = x1+x2;   y0 = s1+s2;
// x-y0^2 = [(x1^2-1)-s1^2] + [x2*(x+x1)-s2*(y0+s1)]

#include "abs_relh.hpp" // Wegen abs_divh1()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include "hilfe.hpp"    // Wegen Cut26()
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;

real T_x(const interval& Xi, const real& delx)
{
    interval Y0,B,C,D,E,X1,X2,S1,S2,G,H,P,S;
    real r1,r2,rel,app,delb,delc,delh,dele,delg,dels,ep0;
    int ex;

    r1 = Cut26(Inf(Xi));  r2 = Cut26(Sup(Xi));
    X1 = interval(r1,r2); // x=x1+x2; x1=Cut26(x) enthalten in X1
    ex = expo(Sup(Xi));
    r1 = comp(0.5,ex-25);
    X2 = interval(0,r1);  // x1 in X1 und x2 in X2 enthalten!

    H = X1*X1; // Fuer alle x1 aus X1 wird x1*x1 exakt berechnet!
    abs_subh1(H,0.0,interval(1),0.0,G,delg); // G=X1*X1-1; delg=0
    abs_addh1(Xi,0.0,X1,0.0,E,dele); // E=x + x1
    abs_mulh1(X2,0.0,E,dele,S,dels); // S=x2*(x + x1)
    abs_addh1(G,delg,S,dels,C,delc);
                                // C=(x1^2-1) + x2*(x+x1) = x^2-1
    rel_sqrt_abs(C,delc,Y0,ep0);
                      // Startwert y0=sqrt(x^2-1) enthalten in Y0
    // ep0: relative Fehlerschranke fuer den Startwert.
    D = interval(-ep0,+ep0);
    D = 0.5 * D*D/(1+D);
    app = Sup(D);  // app: Relativer Approximationsfehler bez. y1
```

```
    r1 = Cut26(Inf(Y0));
    r2 = Cut26(Sup(Y0));
    S1 = interval(r1,r2); // y0 = s1+s2;  s1 = Cut26(y0) aus S1;
    ex = expo(Sup(S1));
    r1 = comp(0.5,ex-25);
    S2 = interval(0,r1);  // s1 in S1 und s2 in S2 enthalten!!

    B = S1*S1; // s1*s1 ist rundungsfehlerfrei, denn 2*26=52<53
    abs_subh1(G,delg,B,0.0,D,r1);     // D = (x1^2-1)-s1^2;
    abs_addh1(Y0,0.0,S1,0.0,B,delb);  // B = y0 + s1;
    abs_mulh1(S2,0.0,B,delb,G,delg);  // G = s2*(y0 + s1);
    abs_subh1(S,dels,G,delg,B,delb);  // B = x2*(x+x1)-s2*(y0+s1)
    abs_addh1(D,r1,B,delb,S,dels);    // S = x - y0^2
    abs_divh1(S,dels,Y0,0.0,C,delc);  // C = (x-y0^2)/y0

    C = C/2;
    delc = delc/2;
    rel_addh1(Y0,0.0,C,delc,B,rel);
    // Beruecksichtigung des Approximationsfehlers:
    r1 = addu(1.0,app);
    r2 = mulu(rel,r1);
    rel = addu(app,r2);

    return rel; // rel: Schranke des relativen Gesamtfehlers.
}

int main()
{
    interval X = interval(1.000732421875,1024);
    real bnd, diam = 1e-7, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = "
         << bnd << endl;
}
```

Ergebnis:

Für alle Maschinenzahlen $\widetilde{x} \in A_2 = [1 + 3 \cdot 2^{-12}, 1024]$ gilt für den in Gleichung (20) definierten relativen Gesamtfehler ε_f nach (21) die Abschätzung:

(22)
$$\boxed{\left|\frac{\widetilde{y}_1 - \sqrt{\widetilde{x}^2 - 1}}{\sqrt{\widetilde{x}^2 - 1}}\right| \leq \varepsilon(f, A_2) = 2.221305 \cdot 10^{-16}}$$

5.2.3 Fehlerabschätzung in $A_3 = [1024, 44000]$

In diesem Bereich A_3 benutzen wir fast den gleichen Algorithmus wie in A_2, lediglich der in (18) angegebene Term

$$(\widetilde{x}^2 - 1) - \widetilde{y}_0^2 = [(x_1^2 - 1) - s_1^2] + \{x_2 \cdot (\widetilde{x} + x_1) - s_2 \cdot (\widetilde{y}_0 + s_1)\}$$

wird jetzt wie folgt umgeschrieben:

$$(\widetilde{x}^2 - 1) - \widetilde{y}_0^2 = [(x_1^2 - s_1^2) - 1] + \{x_2 \cdot (\widetilde{x} + x_1) - s_2 \cdot (\widetilde{y}_0 + s_1)\}, \tag{23}$$

d.h. in den beiden Summanden $[\ldots]$ wird lediglich die Summationsreihenfolge geändert, um den relativen Auswertefehler dieses Terms zu minimieren. Ist $\widetilde{y}_1 \in S(2, 53)$ dann wieder die auf der Maschine ausgewertete Newton-Näherung, so wird der durch $\widetilde{y}_1 = \sqrt{\widetilde{x}^2 - 1} \cdot (1 + \varepsilon_f)$ definierte relative Gesamtfehler durch das folgende Programm `sqrtx2m1_A3.cpp` abgeschätzt:

```
// Programm:  sqrtx2m1_A3.cpp
// Berechnung des relativen Gesamtfehlers
// y0 +0.5*(x-y0^2)/y0
// x = x1+x2;   y0 = s1+s2;
// x-y0^2 = [(x1^2-s1^2)-1] + [x2*(x+x1)-s2*(y0+s1)]

#include "abs_relh.hpp" // Wegen abs_divh1()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include "hilfe.hpp"    // Wegen Cut26()
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;

real T_x(const interval& Xi, const real& delx)
{
    interval Y0,B,C,D,E,X1,X2,S1,S2,G,H,P,S;
    real r1,r2,rel,app,delb,delc,delh,dele,delg,dels,ep0;
    int ex;

    r1 = Cut26(Inf(Xi));  r2 = Cut26(Sup(Xi));
    X1 = interval(r1,r2); // x=x1+x2; x1=Cut26(x) enthalten in X1
    ex = expo(Sup(Xi));
    r1 = comp(0.5,ex-25);
    X2 = interval(0,r1);  // x1 in X1 und x2 in X2 enthalten!

    H = X1*X1; // Fuer alle x1 aus X1 wird x1*x1 exakt berechnet!
    abs_subh1(H,0.0,interval(1),0.0,G,delg);
                                        // G = X1*X1-1;  delg=0
    abs_addh1(Xi,0.0,X1,0.0,E,dele);  // E = x + x1
    abs_mulh1(X2,0.0,E,dele,S,dels);  // S = x2*(x + x1)
    abs_addh1(G,delg,S,dels,C,delc);
```

```
                                // C = (x1^2-1) + x2*(x+x1) = x^2-1
    rel_sqrt_abs(C,delc,Y0,ep0);
                    // Startwert y0 = sqrt(x^2-1) enthalten in Y0;
    // ep0: relative Fehlerschranke fuer den Startwert.
    D = interval(-ep0,+ep0);
    D = 0.5 * D*D/(1+D);
    app = Sup(D);  // app: Relativer Approximationsfehler bez. y1

    r1 = Cut26(Inf(Y0));  r2 = Cut26(Sup(Y0));
    S1 = interval(r1,r2); // y0 = s1+s2;  s1 = Cut26(y0) aus S1;
    ex = expo(Sup(S1));
    r1 = comp(0.5,ex-25);
    S2 = interval(0,r1);  // s1 in S1 und s2 in S2 enthalten!!

    B = S1*S1; // s1*s1 ist rundungsfehlerfrei, denn 2*26=52 < 53
    abs_subh1(H,0.0,B,0.0,P,r2);      // P = x1^2 - s1^2;
    abs_subh1(P,r2,interval(1),0.0,D,r1); // D = (x1^2 - s1^2)-1;
    abs_addh1(Y0,0.0,S1,0.0,B,delb);  // B = y0 + s1;
    abs_mulh1(S2,0.0,B,delb,G,delg);  // G = s2*(y0 + s1);
    abs_subh1(S,dels,G,delg,B,delb);  // B = x2*(x+x1)-s2*(y0+s1)
    abs_addh1(D,r1,B,delb,S,dels);    // S = x - y0^2
    abs_divh1(S,dels,Y0,0.0,C,delc);  // C = (x-y0^2)/y0
    C = C/2;  delc = delc/2;
    rel_addh1(Y0,0.0,C,delc,B,rel);

    // Beruecksichtigung des Approximationsfehlers:
    r1 = addu(1.0,app);
    r2 = mulu(rel,r1);
    rel = addu(app,r2);
    return rel; // rel: Schranke des relativen Gesamtfehlers.
}

int main()
{
    interval X = interval(1.5,2.5);
    real bnd, diam = 1e-6, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Ergebnis:

Für alle Maschinenzahlen $\widetilde{x} \in A_3 = [1024, 44000]$ gilt für den relativen Gesamtfehler ε_f die Abschätzung:

(24)
$$\boxed{\left|\frac{\widetilde{y}_1 - \sqrt{\widetilde{x}^2 - 1}}{\sqrt{\widetilde{x}^2 - 1}}\right| \leq \varepsilon(f, A_3) = 2.220461 \cdot 10^{-16}}$$

5.2.4 Fehlerabschätzung in $A_4 = [44000, \texttt{MaxReal}]$

In diesem Bereich benutzen wir die Approximation:

$$\sqrt{x^2-1} \approx g_4(x) := x - \frac{1}{2x}, \quad x \geq 44000; \tag{25}$$

Eine Oberschranke für den durch $g_4(x) = \sqrt{x^2-1} \cdot (1+\varepsilon_{app})$ definierten relativen Approximationsfehler ε_{app} findet man wie folgt:

$$\begin{aligned}
\left|\frac{\sqrt{x^2-1} - \left(x-\frac{1}{2x}\right)}{\sqrt{x^2-1}}\right| &= \left|\frac{\left[\sqrt{x^2-1} - \left(x-\frac{1}{2x}\right)\right]\cdot\left[\sqrt{x^2-1} + \left(x-\frac{1}{2x}\right)\right]}{\sqrt{x^2-1}\cdot\left[\sqrt{x^2-1} + \left(x-\frac{1}{2x}\right)\right]}\right| \\
&= \left|\frac{x^2-1-\left(x-\frac{1}{2x}\right)^2}{\sqrt{x^2-1}\cdot\left[\sqrt{x^2-1} + \left(x-\frac{1}{2x}\right)\right]}\right| \\
&< \frac{1}{4x^2\cdot(x^2-1)}
\end{aligned}$$

Da die rechte Seite monoton fällt, gilt die Abschätzung[6]:

$$\left|\frac{\sqrt{x^2-1} - \left(x-\frac{1}{2x}\right)}{\sqrt{x^2-1}}\right| < \frac{1}{4\cdot 44000^2\cdot(44000^2-1)} < 6.671\cdot 10^{-20} = \varepsilon(app)$$

Die Auswertung der Funktion $g_4(x)$ erfolgt für alle $\widetilde{x} \in A_4$ auf der Maschine durch:

$$\begin{aligned}
\widetilde{g}_4(\widetilde{x}) &:= \widetilde{x} \ominus 0.5 \odot (1 \oslash \widetilde{x}) = g_4(\widetilde{x})\cdot(1+\varepsilon_{g_4}) \quad |\varepsilon_{g_4}| \leq \varepsilon(g_4) \\
&= \sqrt{\widetilde{x}^2-1}\cdot(1+\varepsilon_{app})\cdot(1+\varepsilon_{g_4}) = \sqrt{\widetilde{x}^2-1}\cdot(1+\varepsilon_f),
\end{aligned}$$

und für den relativen Gesamtfehler ε_f gilt wieder die Abschätzung:

$$|\varepsilon_f| \leq \varepsilon(f) = \varepsilon(app) + \varepsilon(g_4)\cdot[1+\varepsilon(app)]$$

Die Berechnung einer Oberschranke $\varepsilon(f)$ des relativen Gesamtfehlers erfolgt mit dem nachfolgenden Programm `sqrtx2m1_A4.cpp`:

```
// Programm:  sqrtx2m1_A4.cpp
// Berechnung des relativen Gesamtfehlers von:
// sqrt(x^2-1) approx x-1/(2*x)  fuer x aus [44000,MaxReal]

#include "abs_relh.hpp" // Wegen abs_divh1()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;
```

[6] Im Programm `sqrtx2m1_A4.cpp` wird der Approximationsfehler in jedem Teilintervall einzeln abgeschätzt.

```
real T_x(const interval& Xi, const real& delx)
{
    interval A,B;
    real app,rel,rela,delb;

    abs_divh1(interval(1),0.0,Xi,delx,A,app);     // A = 1/Xi
    abs_divh1(A,app,interval(2),0.0,B,delb);      // B = [1/Xi]/2
    rel_subh1(Xi,0.0,B,delb,A,rela);         // A = Xi - [1/Xi]/2
    // rela: relativer Auswertefehler von x - [1/x]/2

    // Berechnung des relativen Approximationsfehlers:
    B = Xi*Xi;
    A = 1 / (4*B*(B-1));
    app = Sup(A);  // app:  relativer Approximationsfehler

    // Beruecksichtigung des Approximationsfehlers:
    rel  = addu(1.0,app);
    delb = mulu(rel,rela);
    rel  = addu(app,delb);

    return rel;     // Rueckgabe einer Schranke rel des
                    // relativen Gesamtfehlers.
}

int main()
{
    interval X = interval(44000,1e10);
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = "
         << bnd << endl;
}
```

Ergebnis:

Für alle Maschinenzahlen $\widetilde{x} \in A_4 = [44000, \texttt{MaxReal}]$ gilt für den relativen Gesamtfehler ε_f die Abschätzung:

(26)
$$\boxed{\left|\frac{\widetilde{g}_4(\widetilde{x}) - \sqrt{\widetilde{x}^2 - 1}}{\sqrt{\widetilde{x}^2 - 1}}\right| \leq \varepsilon(f, A_4) = 2.221114 \cdot 10^{-16}}$$

Zusammenfassung:

Bezeichnen wir in den vier Teilbereichen A_i mit $\widetilde{g}_i(\widetilde{x}) \in S(2,53)$ den auf der Maschine berechneten Näherungswert für den exakten Funktionswert $\sqrt{\widetilde{x}^2-1} \in I\!R$, so gilt für die durch $\widetilde{g}_i(\widetilde{x}) = \sqrt{\widetilde{x}^2-1} \cdot (1+\varepsilon_{f_i})$ definierten relativen Gesamtfehler ε_{f_i} im ganzen Definitionsbereich $|\widetilde{x}| \in [1, \texttt{MaxReal}]$ die Abschätzung:

$$|\varepsilon_{f_i}| = \left| \frac{\widetilde{g}_i(\widetilde{x}) - \sqrt{\widetilde{x}^2-1}}{\sqrt{\widetilde{x}^2-1}} \right| \leq \max_{i=1\ldots4}\{\varepsilon(f,A_i)\} = 2.221305 \cdot 10^{-16} =: \varepsilon(f)$$

Bei vorausgesetzter hochgenauer Arithmetik ist die relative Fehlerschranke der Grundoperationen gegeben durch $\varepsilon(h) := 2^{-52} = 2.2204\ldots \cdot 10^{-16}$. Da die obige Schranke $\varepsilon(f)$ nur minimal größer ist als $\varepsilon(h)$, habe wir für $f(x) = \sqrt{x^2-1}$ bezüglich der Fehlerschranke einen nahezu optimalen Algorithmus gewählt. Lässt man etwas größere Fehlerschranken zu, so kann die Laufzeit der jeweiligen Implementierung nur geringfügig verbessert werden. Die Funktion $f(x) = \sqrt{\widetilde{x}^2-1}$ wird daher in C-XSC durch den in diesem Abschnitt beschriebenen Algorithmus realisiert, vgl. dazu auch den Quelltext auf Seite 23.

6 Intervallargumente

Mit der Vereinbarung `interval x;` wird ein Maschinenintervall $\texttt{x} \subseteq D_f$ vorgegeben, und für alle reellen $x \in \texttt{x}$ ist eine Maschineneinschließung `W` des Wertebereichs

$$W_{\texttt{x}} := \left\{ y \in I\!R \,\middle|\, y = \sqrt{x^2-1} \ \wedge\ x \in \texttt{x} \right\} \subset \texttt{W} \in IR \tag{27}$$

gesucht. Wegen $f(x) \equiv f(|x|)$ kann man sich beschränken auf das Argumentintervall `z = abs(x)`, wobei $\texttt{abs}(\texttt{x}) := \{|r| \,|\, r \in x\}$ die Menge der Absolutbeträge ist. Es gilt also $W_{\texttt{x}} = W_{\texttt{z}}$. Da $f(x) = \sqrt{x^2-1}$ für alle $x \in \texttt{z}$ monoton wächst, ist die Berechnung der gesuchten Einschließung `W` recht einfach:

Zur Berechnung von `Inf(W)` bestimmt man zunächst `r1=Sqrtx2m1(Inf(z))` und muss dann diesen Maschinenwert noch mit `q_sqrtx2m1m` < 1 multiplizieren, um eine garantierte Unterschranke aller Funktionswerte $f(x)$, mit $x \in \texttt{z}$ zu erhalten. Mit Hilfe der obigen Fehlerschranke $\varepsilon(f)$ wird `q_sqrtx2m1m` vorher durch Aufruf der Funktion `eps2fractions(...)` aus dem Modul `bnd_util` berechnet. Weitere Einzelheite dazu findet man in [2].

Die Berechnung von `Sup(W)` erfolgt ganz analog. Man berechnet zunächst wieder `r2=Sqrtx2m1(Sup(z))` und muss dann noch abschließend mit der Konstanten `q_sqrtx2m1p` > 1 multiplizieren, um eine garantierte Oberschranke aller Funktionswerte $f(x)$, mit $x \in \texttt{z}$ zu erhalten. Im Falle $\texttt{expo}(\texttt{Sup}(\texttt{z})) >= 26$ ist jedoch `Sup(z)` wegen $\sqrt{x^2-1} < |x|$ die optimale Oberschranke, vgl. dazu Seite 23, ...

7 Numerische Ergebnisse

Punktargumente:

1. $\widetilde{x} = 1 \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{0.00000000000000000000E+000}$
2. $\widetilde{x} = \texttt{succ(1.0)} \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{2.10734242554470173340E-008}$
3. $\widetilde{x} = 1.0009765625 \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{4.42049621006104509480E-002}$
4. $\widetilde{x} = 1.03125 \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{2.51945554634329660360E-001}$
5. $\widetilde{x} = 2 \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{1.73205080756887719318E+000}$
6. $\widetilde{x} = 520 \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{5.19990384606494444460E+002}$
7. $\widetilde{x} = 1024 \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{1.02399951171863358468E+003}$
8. $\widetilde{x} = 1025 \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{1.02499951219500576372E+003}$
9. $\widetilde{x} = 44000 \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{4.39999999886363657424E+004}$
10. $\widetilde{x} = \texttt{MaxReal} \quad \rightsquigarrow \quad \texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{1.79769313486231570815E+308}$

Intervallargumente:

1. $\texttt{x} = [1, 1]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{0.00000000000000E+000},\texttt{0.00000000000000E+000}]$
2. $\texttt{x} = [\texttt{succ(1.0)},\texttt{succ(1.0)}]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{2.107342425544700E-008},\texttt{2.107342425544705E-008}]$
3. $\texttt{x} = [1, 2]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{0.00000000000000E+000},\texttt{1.732050807568880E+000}]$
4. $\texttt{x} = [1.03125, 1.03125]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{2.519455546343294E-001},\texttt{2.519455546343300E-001}]$
5. $\texttt{x} = [2, 2]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{1.732050807568876E+000},\texttt{1.732050807568880E+000}]$
6. $\texttt{x} = [520, 520]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{5.199990384606491E+002},\texttt{5.199990384606501E+002}]$
7. $\texttt{x} = [1025, 1025]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{1.024999512195005E+003},\texttt{1.024999512195007E+003}]$
8. $\texttt{x} = [32345678, 32345678]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{3.234567799999996E+007},\texttt{3.234567800000003E+007}]$
9. $\texttt{x} = [42345678, 42345678]$
   $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{4.234567799999995E+007},\texttt{4.234567800000000E+007}]$
10. $\texttt{x} = [44000, 44000]$
    $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{4.399999998863633E+004},\texttt{4.399999998863642E+004}]$
11. $\texttt{x} = [\texttt{MaxReal},\texttt{MaxReal}]$
    $\rightsquigarrow \quad W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{1.797693134862314E+308},\texttt{1.797693134862316E+308}]$

8 Quelltext für Punkt- und Intervallargumente

```
// Programm sqrtx2m1.cpp zur Auswertung der Funktion
// sqrt(x^2-1) fuer Punktargumente x aus [1,MaxReal]
// und fuer Intervallargumente.
#include <rmath.hpp>  // Wegen sqrt()
#include <interval.hpp>
#include "hilfe.hpp"  // Wegen Cut26()
#include <iostream>

using namespace std;
using namespace cxsc;

real q_sqrtx2m1p(4503599627370501.0/4503599627370496.0);
real q_sqrtx2m1m(9007199254740986.0/9007199254740992.0);

real Sqrtx2m1(const real& x)
{  // sqrt(x^2-1);  rel. Fehlerschranke:  eps = 2.221305E-16
    const real c1 = 1.000732421875,
               c2 = 44000.0,
               c3 = 1024.0;  // c1,c2,c3 werden exakt gespeichert
    real res,ep,ep2,s1,s2,x1,x2,arg=x;
    if (sign(arg)<0) arg = -arg; // arg = |x| >= 0
    if (arg <= c1) { // x = 1+ep; x^2-1 = 2*ep + ep^2;
        ep = x - 1; // Differenz rundungsfehlerfrei!
        ep2 = ep*ep;  // ep2 i.a. fehlerbehaftet!
        times2pown(ep,1);  // ep = 2*ep;
        res = sqrt(ep+ep2); // res=y0: Startwert;
        // x - y0^2 = (2*eps - s1^2) + [eps^2 - s2*(y0 + s1)]
        s1 = Cut26(res);  s2 = res - s1; // Startwert = s1 + s2;
        arg = ep - s1*s1;  // arg = 2*eps - s1^2;
        arg += (ep2 - s2*(res+s1));  // arg = x - y0^2
        if (sign(arg)>0) {
        arg = arg / res;
        times2pown(arg,-1);
        res += arg;  // 1. Newton-Schritt beendet;
        }            //    eps = 2.221261E-16
    } else if (arg<c2) {
        // x-y0^2 = [(x1^2-1)-s1^2] + [x2*(x+x1)-s2*(y0+s1)]
        x1 = Cut26(arg);  x2 = arg - x1;  // arg = x = x1 + x2;
        ep2 = x2*(arg+x1);  // ep2 ist fehlerbehaftet
        x2 = x1*x1;  ep = x2-1;
        res = sqrt(ep+ep2); // res ist Startwert fuer das
                            // folgende Newton-Verfahren.
        s1 = Cut26(res);  s2 = res - s1;  // Startwert = s1 + s2;
        ep2 = ep2 - s2 * (res+s1); // ep2=[x2*(x+x1)-s2*(y0+s1)]
        if (arg<c3) ep -= s1*s1;   // ep = (x1^2-1) - s1^2;
        else {
```

```
            x2 -= s1*s1;  // x2 = x1^2-s1^2
            ep = x2 - 1; }         // ep = (x1^2-s1^2) - 1;
        ep += ep2;        // ep = x - y0^2;
        ep /= res;
        times2pown(ep,-1);
        res = res + ep;  // 1. Newtonschritt in hoher Genauigkeit
                         //    beendet;  eps = 2.221305E-16
    } else { // arg = |x| >= 44000;
        res = -1/arg;
        times2pown(res,-1); // Multiplikation mit 0.5;
        res += arg;  // res = x - 1/(2*x);  eps = 2.221114E-16
    }

    return res;
} // Sqrtx2m1 (Punktargumente)

interval Sqrtx2m1(const interval& x)
{
    interval z = abs(x);
    real r1,r2;
    r1 = sqrtx2m1(Inf(z)) * q_sqrtx2m1m;
    r2 = Sup(z);
    if (expo(r2)<26)
        r2 = sqrtx2m1(r2) * q_sqrtx2m1p;
    // expo(r2) >= 26 --> r2 = Sup(z) ist optimale Oberschranke!

    return interval(r1,r2);
} // Sqrtx2m1 (Intervallargumente)

int main()
{
    interval x,y;

    while (1) {
        cout << "interval x = ?" << endl;
        cin >> x;
        y = Sqrtx2m1(x);
        cout << SetPrecision(20,20) << Scientific
             << "Sqrtx2m1(x) = " << y << endl << endl;
    };
}
```

Hinweis:

Zur Vermeidung von Namenskonflikten beim Übersetzen des obigen C-XSC Programms `sqrtx2m1.cpp` wurden beide Funktionen mit `Sqrtx2m1` bezeichnet. Die gleichen Funktionen sind in C-XSC unter dem Namen `sqrtx2m1` deklariert, für Punktargumente in `rmath.hpp` und für Intervallargumente in `imath.hpp`.

Literatur

[1] American National Standards Institute/Institute of Electrical and Electronics Engineers: “IEEE Standard for Binary Floating-Point Arithmetic”; ANSI/IEEE Std 754–1985, New York, 1985.

[2] Blomquist, F.: A priori Fehlerabschätzungen in C-XSC für Grundoperationen, Horner-Schema und Funktionen; Wissenschaftliches Rechnen / Softwaretechnologie, Universität Wuppertal, 2003. URL: `http://www.math.uni-wuppertal.de/wrswt/literatur/a_priori.pdf`

[3] Braune, K.; Krämer, W.: High Accuracy Standard Functions for Real and Complex Intervals, in Kaucher, E., Kulisch, U., Ullrich, Ch: *Computerarithmetic: Scientific Computation and Programming Languages*, B. G. Teubner, Stuttgart, pp. 81–114, 1987.

[4] Hammer, R.; Hocks, M.; Kulisch, U.; Ratz, D., C++ Toolbox for Verified Computing: Basic Numerical Problems. Springer-Verlag, Berlin / Heidelberg / New York, 1995.

[5] Herzberger, J. (Ed), Topics in Validated Computations. Proceedings of IMACS-GAMM International Workshop on Validated Numerics, Oldenburg, 1993. North Holland, 1994.

[6] Hofschuster, W.; Krämer, W.: Ein rechnergestützter Fehlerkalkül mit Anwendung auf ein genaues Tabellenverfahren. Preprint 96/5 des Instituts für Wissenschaftliches Rechnen und Mathematische Modellbildung, Universität Karlsruhe, 1996.

[7] Blomquist, F.; Hofschuster, W.; Krämer, W.: Sichere a priori Fehlerabschätzung und Implementierung der Funktion zweier Variabler $\log(\sqrt{x^2+y^2})$. Preprint 2004/1 des Instituts für Wissenschaftliches Rechnen und Sofwaretechnologie, Bergische Universität Wuppertal, 2004.

[8] Hofschuster, W.; Krämer, W.: FI_LIB, eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-*double*-Format. Preprint 98/7 des IWRMM, Universität Karlsruhe, 227 Seiten, 1998.

[9] Hofschuster, W.: Zur Berechnung von Funktionswerteinschließungen bei speziellen Funktionen der mathematischen Physik. Dissertation, Universität Karlsruhe, 2000.

[10] Hofschuster, W.; Krämer, W.; Wedner, S.; Wiethoff, A., C-XSC 2.0 – A C++ Class Library for Extended Scientific Computing. Preprint 2001/1, Wissenschaftliches Rechnen / Softwaretechnologie, Universität Wuppertal, 2001.

[11] Hofschuster, W., Krämer, W.: C-XSC – A C++ Class Library for Extended Scientific Computing. To appear in *Numerical Software with Result Verification*. R. Alt, A. Frommer, B. Kearfott, W. Luther (eds), Springer Lecture Notes in Computer Science, 2004.

[12] 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.

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

[14] Krämer, W., Wolff von Gudenberg, J. (eds): *Scientific Computing, Validated Numerics, Interval Methods*, Kluwer Academic Publishers Boston/Dordrecht/London, 398 pages, 2001.

[15] Krämer, W.; Blomquist, F.: Algorithms with Guaranteed Error Bounds for the Error Function and the Complementary Error Function. Eingereicht für Theoretical Computer Science (TCS), Special issue: Real Numbers and Computers, 2004.

[16] 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.

[17] Tang, P.T.P.: Table-Driven Implementation of the Logarithm Function in IEEE Floating-Point Arithmetic. ACM Trans. on Math. Software, Vol. **16**, No. 4, pp 378-400, 1990.

[18] Tang, P.T.P.: Table-Driven Implementation of the Expm1 Function in IEEE Floating-Point Arithmetic. ACM Trans. on Math. Software, Vol. **18**, No. 2, pp 211-222, 1992.