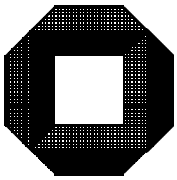


UNIVERSITÄT KARLSRUHE

FI_LIB,
eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-double-Format

W. Hofschuster und W. Krämer

Preprint Nr. 98/7

Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung

76128 Karlsruhe

Anschrift der Verfasser:

Werner Hofschuster

Walter Krämer
Institut für Wissenschaftliches Rechnen und
Mathematische Modellbildung (IWRMM)

Universität Karlsruhe
Postfach 6980
76128 Karlsruhe Bundesrepublik Deutschland

Das Postscript-File dieses Preprints ist über FTP unter der Adresse

`iamk4515.mathematik.uni-karlsruhe.de`

im Verzeichnis

`/pub/iwrmm/preprints`

abrufbar.

FI_LIB,

eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-double-Format

Projektdokumentation

Werner Hofschuster und Walter Krämer

Institut für Angewandte Mathematik (IAM)

Institut für Wissenschaftliches Rechnen
und Mathematische Modellbildung (IWRMM)

Universität Karlsruhe (TH)

Stand Juli 1998

Inhaltsverzeichnis

Kapitel 1

Vorwort

Die in dieser Projekt–Dokumentation beschriebene ANSI–C Unterprogrammbibliothek `fi_lib` für die Berechnung der elementaren mathematischen Funktionen exp, log, sin ... stellt einen guten Kompromiß zwischen hoher Ergebnisgenauigkeit mit gesicherten relativen Fehlerschranken (Intervallrechnung), kurzen Ausführungszeiten und breiter Verwendbarkeit (Portabilität) auf vielen Rechnerplattformen dar. Die vorliegende Dokumentation beschreibt im wesentlichen den aktuellen Stand des vom IWRMM geförderten Projekts *„Schnelle und verläßliche Funktionen“*. Sie ist vorwiegend als interne Arbeitsgrundlage für Weiterentwicklungen gedacht und ist insofern als Zwischenbericht (Stand Juli 1998) zu verstehen.

Der frei erhältliche Quelltext der Bibliothek befindet sich auf dem Server

`iamk4515.mathematik.uni-karlsruhe.de`

im Verzeichnis

`/pub/iwrmm/software` .

Die entsprechende `README`–Datei ist im Anhang A.4 abgedruckt.

1.1 Implementierte Funktionen

Der in Programmiersprachen wie C, Pascal, Fortran vorhandene Satz mathematischer Standardfunktionen[1] ist in der Regel eine Teilmenge der nachfolgen aufgelisteten Funktionen: exp, ln, sin, cos, tan, cot, arcsin, arccos, arctan, arccot, sinh, cosh, tanh, coth, arsinh, arcosh, artanh, arcoth, $\exp(x)-1$ und $\ln(1+x)$. Die realisierte Bibliothek umfaßt diese Funktionen. Dabei wurden die beiden zuletzt genannten Hilfsfunktionen (Gefahr der Auslöschung für $|x|$ klein) bereits zur Implementierung der anderen Funktionen verwendet, sind aber insbesondere auch bei der Realisierung von komplexwertigen Funktionen nützlich [11]. Im Hinblick auf den Einsatz im stark wachsenden Bereich der Numerik mit Ergebnisverifikation, wurden die angegebenen Routinen auch für Intervallargumente (Wertebereichseinschließungen) implementiert.

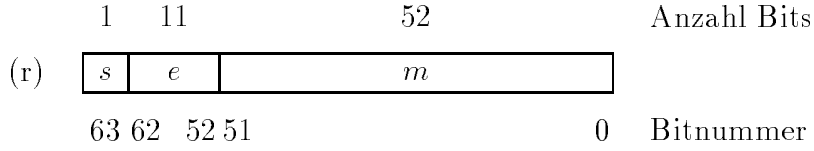
[1] Die Begriffe „Standardfunktionen“ und „elementare Funktionen“ werden in dieser Dokumentation synonym benutzt.

1.2 Der IEEE Standard 754

Der IEEE Standard 754 [4] für binäre Gleitkommaarithmetik zielte in erster Linie auf eine Vereinheitlichung der Gleitkommaarithmetik von Mikroprozessoren. Er wurde sehr schnell akzeptiert und ist inzwischen weit verbreitet. Auch im Bereich der Workstations und auf einigen Großrechnern hat er sich etabliert.

Dieser Standard legt vor allem Datenformate und Eigenschaften der arithmetischen Grundoperationen $+$, $-$, $*$, / fest. Er verlangt, daß alle Gleitkommaoperationen so ausgeführt werden, als ob zunächst ein exaktes Zwischenergebnis mit unbeschränkter Genauigkeit und unbeschränktem Exponentenbereich berechnet würde, das anschließend gemäß der eingestellten Rundungsart zu einer Gleitkommazahl gerundet wird.

Für das sogenannte „double“ Format werden in einer Wortbreite von 64 Bit 1 Vorzeichenbit s sowie 11 Bit für die Darstellung des Exponenten e und 53 Bit für die (in der Regel normalisierte) Mantisse m vorgeschrieben (siehe Abbildung 1.1). Das führende 53-ste Mantissenbit ist im Fall einer normalisierten Zahl immer Eins und wird deshalb nicht gespeichert (hidden Bit). Das Datenformat entspricht in etwa einem 16-stelligen Dezimalformat.



$$
\mathrm{r} = \begin{cases}
(-1)^s \cdot 1.m \cdot 2^{e-1023} & 1 \le e \le 2046 & \text{normalisierte Zahl} \\
(-1)^s \cdot 0.m \cdot 2^{-1022} & e = 0, m \neq 0 & \text{denormalisierte Zahl} \\
(-1)^s \cdot 0 & e = 0, m = 0 & \text{Null, vorzeichenbehaftet} \\
(-1)^s \cdot \infty & e = 2047, m = 0 & \text{„Unendlich“, vorzeichenbehaftet} \\
\mathrm{NaN} & e = 2047, m \neq 0 & \text{„Not a Number“}
\end{cases}
$$

Abbildung 1.1: IEEE double Format

Leider stellt der zitierte Standard, mit Ausnahme der Wurzelfunktion (diese wird wie eine Grundoperation behandelt), keine Forderung an die Ergebnisgenauigkeit der mathematischen Funktionen.

1.3 Generelles Vorgehen bei der Funktionswerteberechnung

Die Funktionswertbestimmung geschieht üblicherweise in den drei Schritten Argumentreduktion, eigentliche Approximation und Ergebnisanpassung. Diese Schritte werden in der Regel in einem höhergenauen, internen Gleitkommasystem durchgeführt (Mitführung von

Schutzziffern). Abschließend wird das vorläufige Ergebnis in das 64 Bit Format zurückgerundet.

Mathematische Coprozessoren, wie z.B. die Prozessoren von Intel, Cyrix oder Motorola, verwenden intern ein 80 Bit Datenformat. Auch die Berechnung der hochgenauen Standardfunktionen in den bisherigen Implementierungen von PASCAL-XSC [23] und C-XSC [26] geschieht mit Hilfe eines 80 Bit Gleitkommadatenformates, welches allerdings aus Gründen der Portabilität dieser Sprachen bzw. deren Laufzeitsystemen durch ganzzahlige Arithmetik simuliert wird. Diese Simulation führt zu ganz erheblichen Laufzeiteinbußen.

Im Augenblick stehen sich sehr schnelle Standardfunktionen (insbesondere bei Verwendung von Coprozessoren) mit ungewisser Genauigkeit, und die um eine Größenordnung langsameren, nicht hardwareunterstützten, aber hochgenauen (durch nachvollziehbare a priori Fehlerabschätzungen abgesicherten) Funktionsunterprogramme der XSC-Sprachen gegenüber. Die Unsicherheit bezüglich der Genauigkeit der Coprozessorroutinen dokumentiert z.B. die Berechnung von $\sin(x)$ in BIAS [27] für das Punktargument x:=`400921FB54442D18`. Mit BIAS ergibt sich die sehr grobe Einschließung `[-3.3E-016, 5.7E-016]`, aus der nichteinmal das Vorzeichen des exakten Funktionswertes ablesbar ist, wohingegen die entsprechende PASCAL-XSC Intervallroutine, das maximal genaue Ergebnisintervall `[1.224646799147353E-016, 1.224646799147354E-016]` liefert.

Die hier beschriebene Bibliothek stellt einen Kompromiß dar zwischen dem Wunsch nach maximaler Genauigkeit, dem Wunsch nach höchsten Ausführungsgeschwindigkeiten und dem Wunsch nach möglichst portablen Routinen. Dazu wird folgendermaßen vorgegangen:

1. Es wird keine Emulation mittels Integer-Arithmetik durchgefürt.
2. Es werden nur die Hardware-Grundoperationen des IEEE double Formates verwendet.
3. Die Approximationsgenauigkeit wird nur auf double Genauigkeit ausgelegt, was auf niederere Approximationsgrade führt.
4. Zentrale Routinen (z.B. exp und log) werden als schnelle Tabellenverfahren realisiert.
5. Die Rundungsart des Prozessors wird nicht geändert (u.U. zeitaufwendige Operation). Würde man den Rundungsmodus „round-to-nearest“ erzwingen, so könnten die worst-case-Fehlerschranken nahezu halbiert werden.
6. Zur Realisierung der Intervallversionen wird keine Maschinenintervallarithmetik verwendet, sondern es werden a priori Fehlerabschätzungen herangezogen (Laufzeitgewinn).
7. Die Portabilität wird durch Implementierung in ANSI-C gewährleistet.

Das beschriebene Vorgehen gewährleistet Ergebnisse mit verläßlichen Fehlerschranken, ermöglicht sichere Intervallrechnung und liefert Routinen mit recht guter relativer Genauigkeit bei gleichzeitig kurzen Ausführungszeiten. Weiterhin laufen diese Routinen sofort in jeder IEEE konformen Umgebung.

Leider verliert man gegenüber den bisherigen XSC-Realisierungen die sehr hohe bzw. maximale Genauigkeit. Auch kann man nicht mehr erwarten, daß man bei Verwendung dieser Routinen bitgleiche Ergebnisse auf allen Plattformen (ungewisser Rundungsmode!) erhält.

1.4 Projektziele

Die in dieser Dokumentation beschriebene Funktionsbibliothek entstand im Rahmen eines gemeinsamen Projektes des Instituts für Wissenschaftliches Rechnen und Mathematische Modellbildung (IWRMM) und des Instituts für Angewandte Mathematik (IAM) an der Universität Karlsruhe.

Die vorgegebenen Projektziele werden hier nochmals zusammenfassend genannt:

- Entwicklung und softwaremäßige Bereitstellung eines Fehlerkalküls zur (halb-)automatischen Fehlerabschätzung.
- Entwicklung schneller Standardfunktionen im IEEE-Datenformat double (siehe [4, 9]) für Punkt- und Intervallargumente mit mathematisch gesicherten Fehlerschranken.
- Implementierung dieser Funktionen aufgrund der gewünschten Portabilität in ANSI-C (Vorgegebene Benutzerplattformen: PC unter DOS, OS/2 und LINUX (jeweils mit GNU-C Compiler), SUN-Workstations, HP-Workstations, IBM-RS6000, IBM-SP/2).
- Bereitstellung dieser Funktionen in einer Bibliothek, die eigenständig unter ANSI-C verwendet werden kann.
- Implementierung von schnellen Intervallgrundoperationen zur Verwendung unter ANSI-C.
- Bereitstellung eines C++ Interfaces (Ausnutzung der in C++ vorhandenen Konzepte wie Funktionsüberladungen, Operatorvereinbarungen usw.)
- Die elementaren Funktionen der entstehenden Bibliothek sollen die bestehenden Standardfunktionen von Pascal-XSC und C-XSC ergänzen bzw. ersetzen, d.h. es müssen bestimmte Vorgaben und Namenskonventionen eingehalten werden.
- Eine spezielle Version der entstehenden Bibliothek soll in Fortran-XSC [2] (eine Erweiterung von Fortran-90 bzw. Fortran-95) integriert werden.
- Erstellung eines Testprogramms mit ausführlichen Testläufen.
- Beispielhafte Implementierung numerischer Verifikationsalgorithmen, wie z.B. des Intervall-Newton-Verfahrens.
- Dokumentation der Fehlerabschätzungen und der Implementierungen.

[2] gemeinsames Projekt mit dem Institut für Wissenschaftliches Rechnen der Universität Dresden

- Bereitstellung der gesamten Software auf einem FTP-Server.

Die Bearbeitung der obengenannten Projektziele ist mittlerweile erfolgreich abgeschlossen. Zur Zeit werden einige weiterführende Ideen und Verbesserungsmöglichkeiten untersucht (siehe hierzu die Bemerkungen im Anhang), die möglicherweise in eine zukünftige Version der Bibliothek eingearbeitet werden.

Die folgenden Funktionen werden derzeit in der Bibliothek `fi_lib` zur Verfügung gestellt: sqr, sqrt, exp, expm1, exp2, exp10, log, log1p, log2, log10, sin, cos, tan, cot, arcsin, arccos, arctan, arccot, sinh, cosh, tanh, coth, arsinh, arcosh, artanh, arcoth.

Weiterhin sind die Intervallgrundoperationen, sowie ein C++ – Interface implementiert.

Kapitel 2

Notation

In der Arbeit verwendete Notation:

$S = S(b, l, \underline{e}, \overline{e})$	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
S_N	Menge der normalisierten Zahlen des Gleitkommarasters S
S_D	Menge der denormalisierten Zahlen des Gleitkommarasters S
$\circ \in \{+, -, \bullet, /\}$	exakte reelle Operation
$\boxdot\ , \circ \in \{+, -, \bullet, /\}$	Gleitkommaoperator, Maschinenoperation mit Rundung zur nächstgelegenen Gleitkommazahl
$\triangledown\ , \circ \in \{+, -, \bullet, /\}$	Maschinenoperation mit Rundung nach unten
$\triangle\ , \circ \in \{+, -, \bullet, /\}$	Maschinenoperation mit Rundung nach oben
$\lvert a \circ b - a \boxdot b \rvert := \lvert (a \circ b) - (a \boxdot b) \rvert$	Implizite Klammerung beachten!
`MinReal`	kleinste positive normalisierte Gleitkommazahl, für IEEE-Datenformat gilt: `MinReal`:=$2.22\ldots \cdot 10^{-308}$
`MaxReal`	größte normalisierte Gleitkommazahl, für IEEE-Datenformat gilt: `MaxReal`:=$1.78\ldots \cdot 10^{308}$
$\tilde{a}$	auf der Maschine berechnete, i.a. fehlerbehaftete Größe
ulp	eine Einheit in der letzten Mantissenstelle (unit last place)
ε^*	relative Maschinengenauigkeit, $\varepsilon^* := \frac{1}{2}2^{1-l} = 2^{-l}$
`eps52`	`eps52`:= $2^{1-53} = 2.22044\ldots \cdot 10^{-16}$
`eps53`	`eps53`:= $\frac{1}{2}2^{1-53} = 1.11022\ldots \cdot 10^{-16} = \varepsilon^*$
`EpsQuer`	Aktuelle relative Genauigkeit einer Maschinenoperation
succ(x), $x \in S$	Gleitkommanachfolger von x
pred(x), $x \in S$	Gleitkommavorgänger von x
$\mathbb{R}^+$	Menge der positiven reellen Zahlen

$I\!I\!R$	Menge der abgeschlossenen Intervalle über den reellen Zahlen
IS	Menge der Maschinenintervalle $IS = \{[\underline{a}, \overline{a}] \mid \underline{a}, \overline{a} \in S, \underline{a} \leq \overline{a}\}$
$\lvert A\rvert, A \in I\!I\!R$	$\lvert A\rvert := \max_{a \in A} \lvert a\rvert$, Betragsmaximum
$\langle A\rangle, A \in I\!I\!R$	$\langle A\rangle >:= \min_{a \in A} \lvert a\rvert$, Betragsminimum
$\mathrm{diam}(A)$, $A \in I\!I\!R$	Durchmesser eines Intervalls, $\mathrm{diam}(A) := \sup(A) - \inf(A)$
$x\vert_n$, $x \in S$	Die führenden n Mantissenbits von x
$x\vert_{\mathrm{IEEE}}$, $x \in I\!R$	Die reelle Zahl x gerundet zur nächstgelegenen Gleitkommazahl des IEEE-double-Formates

Kapitel 3

Fehlerabschätzung

3.1 Fehlerkalkül

In diesem Abschnitt werden Techniken hergeleitet, welche es erlauben, Rundungsfehler in Gleitkommaalgorithmen sicher abzuschätzen. Auch die Fehlerfortpflanzung wird durch den hier vorgestellten Kalkül (siehe auch [20, 21]) mit erfaßt. Die a priori berechenbaren Fehlerschranken sind auf den vorgegebenen Argumentbereichen gleichmäßig gültige worst-case-Fehlerschranken.

Für den Fehlerkalkül wird vorausgesetzt, daß die Operationen auf der Maschine dem IEEE-Standard [4, 9] entsprechen. Das bedeutet insbesondere, daß für die Grundoperation $\circ \in \{+, -, \bullet, /\}$ und deren Maschinenäquivalent $\boxdot$, $\circ \in \{+, -, \bullet, /\}$ die folgende Beziehung gilt:

$$\bigwedge_{\circ \in \{+,-,\bullet,/\}} \quad \bigwedge_{\substack{a,b \in S \text{ mit} \\ |a \circ b| \in [\texttt{MinReal},\texttt{MaxReal}]}} \quad \left| \frac{a \circ b - a \boxdot b}{a \circ b} \right| \leq \varepsilon^* \ . \tag{3.1}$$

Lemma 1 (Unterlaufbereich) Im Unterlaufbereich $U := (-\texttt{MinReal}, \texttt{MinReal})$ gilt die folgende Abschätzung:

$$\bigwedge_{\circ \in \{+,-,\bullet,/\}} \quad \bigwedge_{\substack{a,b \in S \\ |a \circ b| < \texttt{MinReal}}} \quad |a \boxdot b - a \circ b| \leq \texttt{diam}([0, \texttt{MinReal}]) = \texttt{MinReal} \tag{3.2}$$

.

Beweis:
Hier wird nur verwendet, daß die Vorzeichen von $a \boxdot b$ und $a \circ b$ übereinstimmen □

Die obigen Beziehungen sind richtig, falls als Rundungsmode „Rundung zur nächsten Maschinenzahl“ verwendet wird. Bei Verwendung von gerichtet gerundeten Operationen muß $\varepsilon^* = \frac{1}{2} b^{1-l}$ in Formel (3.1) durch $2\varepsilon^* = b^{1-l}$ ersetzt werden. Dies trifft auch dann zu, wenn man von der Maschinenarithmetik nur voraussetzt, daß sie „faithful“ ist (vgl. [17, 35]).

Im folgenden soll die Fehlerfortpflanzung für die Grundoperationen bei Anwendung auf bereits fehlerbehaftete Argumente untersucht werden.

Seien dazu $\circ \in \{+, -, \bullet, /\}$ und $a \in A \in I\!I\!R$, $b \in B \in I\!I\!R$,

$\tilde{a} = a + \triangle_a$ mit $|\triangle_a| \leq \triangle(a)$,

$\tilde{b} = a + \triangle_b$ mit $|\triangle_b| \leq \triangle(b)$.

Die Größen $A, B, \triangle(a)$ und $\triangle(b)$ seien dabei gegeben. Gesucht ist dann eine Schranke $\triangle(\circ) \in I\!R^+$, so daß

$$\bigwedge_{\substack{a \in A,\ b \in B \\ a \circ b, \tilde{a} \circ \tilde{b} \in [-\texttt{MaxReal}, \texttt{MaxReal}]}} \ \bigwedge_{\substack{|a - \tilde{a}| \leq \triangle(a) \\ |b - \tilde{b}| \leq \triangle(b)}} \left| a \circ b - \tilde{a} \boxdot \tilde{b} \right| \leq \triangle(\circ)$$

gilt.

Die folgenden Sätze beantworten diese Frage für die arithmetischen Grundoperationen. Generell sei dabei vorausgesetzt, daß $a \circ b, \tilde{a} \circ \tilde{b} \in [-\texttt{MaxReal}, \texttt{MaxReal}]$, d.h. daß kein Überlauf auftritt. Die später angegebenen Intervallroutinen brechen bei Überlauf mit einer entsprechenden Fehlermeldung ab.

Satz 1 (Addition) Für die Addition $\circ := +$ gilt die Fehlerfortpflanzungsabschätzung

$$\bigwedge_{\substack{a \in A \\ b \in B}} \ \bigwedge_{\substack{|a - \tilde{a}| \leq \triangle(a) \\ |b - \tilde{b}| \leq \triangle(b)}} \left| a + b - \tilde{a} \boxplus \tilde{b} \right| \leq \triangle(\text{add}) \quad \text{mit}$$

$$\triangle(\text{add}) := \varepsilon^*(|A + B|) + (1 + \varepsilon^*)(\triangle(a) + \triangle(b)) + \texttt{MinReal}$$

Beweis:
Es seien $a \in A, b \in B, \tilde{a} = a + \triangle_a \in A + [-\triangle(a), \triangle(a)], |\triangle_a| \leq \triangle(a)$ und $\tilde{b} = b + \triangle_b \in B + [-\triangle(b), \triangle(b)], |\triangle_b| \leq \triangle(b)$ beliebig aber fest gewählt.

I) Fehlerschranke für $|\tilde{a} + \tilde{b} - \tilde{a} \boxplus \tilde{b}|$:
In dieser Abschätzung werden nur gestörte Argumente berücksichtigt.

a) $\tilde{a} + \tilde{b} \in U$, d.h. die exakte Summe der gestörten Argumente liegt im Unterlaufbereich.
$\tilde{a} + \tilde{b} \in U \Longrightarrow \tilde{a} \boxplus \tilde{b} \in \overline{U} := U \cup \{-\texttt{MinReal}, \texttt{MinReal}\}$
$\overset{\text{Lemma 1}}{\Longrightarrow} |\tilde{a} + \tilde{b} - \tilde{a} \boxplus \tilde{b}| \leq \texttt{MinReal}$

b) $\tilde{a} + \tilde{b} \notin U$, d.h. der Betrag der exakten Summe der gestörten Argumente liegt in $|\tilde{a} + \tilde{b}| \in [\texttt{MinReal}, \texttt{MaxReal}]$:
$\tilde{a} + \tilde{b} \notin U \Longrightarrow \tilde{a} \boxplus \tilde{b} \notin U$
$\Longrightarrow |\tilde{a} + \tilde{b} - \tilde{a} \boxplus \tilde{b}| \leq \varepsilon^* |\tilde{a} + \tilde{b}| \leq \varepsilon^* \Big(|A + B| + \triangle(a) + \triangle(b) \Big)$.
Dabei wird z.B. verwendet, daß $\tilde{a} \in A + [-\triangle(a), \triangle(a)]$ und damit $|\tilde{a}| \leq |A| + \triangle(a)$ gilt.

II) Fehlerschranke für $|a + b - (\tilde{a} + \tilde{b})|$, d.h. Fehlerschranke bei Verwendung der exakten Operation $+$:
$|a + b - (\tilde{a} + \tilde{b})| \leq |a - \tilde{a}| + |b - \tilde{b}| \leq \triangle(a) + \triangle(b)$

Gesamtfehlerabschätzung:

$$
\begin{aligned}
& |a+b-\tilde{a}\boxplus\tilde{b}| \le |a+b-(\tilde{a}+\tilde{b})| + |\tilde{a}+\tilde{b}-\tilde{a}\boxplus\tilde{b}| \\
\overset{\text{I),II)}{\le}\ & \triangle(a)+\triangle(b)+\begin{cases} \texttt{MinReal}, & \text{Fall I) a)} \\ \varepsilon^*\big(|A+B|+\triangle(a)+\triangle(b)\big), & \text{Fall I) b)} \end{cases} \\
\le\ & \begin{cases} \triangle(a)+\triangle(b)+\texttt{MinReal}, & \text{Fall I) a)} \\ \varepsilon^*(|A+B|)+(1+\varepsilon^*)(\triangle(a)+\triangle(b)), & \text{Fall I) b)} \end{cases} \\
\le\ & \varepsilon^*(|A+B|)+(1+\varepsilon^*)(\triangle(a)+\triangle(b))+\texttt{MinReal} \\
=\ & \triangle(\text{add}).
\end{aligned}
$$

Die Größen $a, b, \tilde{a}, \tilde{b}$ waren beliebig gewählt, so daß die Behauptung von Satz 1 unmittelbar folgt □

Satz 2 (Subtraktion) Für die Subtraktion gilt die folgende Fehlerschranke $\triangle(\text{sub})$:

$$\triangle(\text{sub}) := \varepsilon^*(|A-B|)+(1+\varepsilon^*)(\triangle(a)+\triangle(b))+\texttt{MinReal}.$$

Beweis:
Der Beweis wird analog zum Beweis von Satz 1 geführt.

Satz 3 (Multiplikation) Für die Multiplikation $\circ := \bullet$ gilt

$$\bigwedge_{\substack{a\in A\\ b\in B}} \bigwedge_{\substack{|a-\tilde{a}|\le\triangle(a)\\ |b-\tilde{b}|\le\triangle(b)}} \left|a\cdot b-\tilde{a}\boxdot\tilde{b}\right| \le \triangle(\text{mul}) \quad \text{mit}$$

$$\triangle(\text{mul}) := |A||B|\varepsilon^* + (|A|\,\triangle\,(b)+|B|\,\triangle\,(a)+\triangle(a)\,\triangle\,(b))\cdot(1+\varepsilon^*)+\texttt{MinReal}$$

Beweis:
Es seien $a\in A, b\in B, \tilde{a}=a+\triangle_a\in A+[-\triangle\,(a),\triangle(a)], |\,\triangle_a\,|\le\triangle(a)$ und $\tilde{b}=b+\triangle_b\in B+[-\triangle\,(b),\triangle(b)], |\,\triangle_b\,|\le\triangle(b)$ beliebig aber fest gewählt.

I) Abschätzung für $|\tilde{a}\cdot\tilde{b}-\tilde{a}\boxdot\tilde{b}|$:

a) $\tilde{a}\cdot\tilde{b}\in U \Longrightarrow \tilde{a}\boxdot\tilde{b}\in\overline{U} \overset{\text{Lemma 1}}{\Longrightarrow} |\tilde{a}\cdot\tilde{b}-\tilde{a}\boxdot\tilde{b}|\le\texttt{MinReal}$

b) $\tilde{a}\cdot\tilde{b}\notin U \Longrightarrow |\tilde{a}\boxdot\tilde{b}|\notin U$
$\Longrightarrow |\tilde{a}\cdot\tilde{b}-\tilde{a}\boxdot\tilde{b}|\le\varepsilon^*|\tilde{a}\tilde{b}|\le\varepsilon^*(|A|+\triangle(a))(|B|+\triangle(b))$
$= \varepsilon^*(|A||B|+|A|\,\triangle\,(b)+|B|\,\triangle\,(a)+\triangle(a)\,\triangle\,(b)).$

II) Abschätzung für $|a\cdot b-\tilde{a}\cdot\tilde{b}|$:
$|a\cdot b-\tilde{a}\cdot\tilde{b}| = |a\cdot b-(a+\triangle_a)(b+\triangle_b)| = |a\,\triangle_b+b\,\triangle_a+\triangle_a\,\triangle_b|$
$\le |A|\,\triangle\,(b)+|B|\,\triangle\,(a)+\triangle(a)\,\triangle\,(b).$

Gesamtfehlerabschätzung:

$$
\begin{aligned}
& |a\cdot b-\tilde{a}\boxdot\tilde{b}| \le |a\cdot b-\tilde{a}\cdot\tilde{b}| + |\tilde{a}\cdot\tilde{b}-\tilde{a}\boxdot\tilde{b}| \\
\overset{\text{I),II)}{\le}\ & \begin{cases} |A|\,\triangle\,(b)+|B|\,\triangle\,(a)+\triangle(a)\,\triangle\,(b)+\texttt{MinReal}, & \text{Fall I) a)} \\ (|A|\,\triangle\,(b)+|B|\,\triangle\,(a)+\triangle(a)\,\triangle\,(b))(1+\varepsilon^*)+|A||B|\varepsilon^*, & \text{Fall I) b)} \end{cases} \\
\le\ & |A||B|\varepsilon^*+(|A|\,\triangle\,(b)+|B|\,\triangle\,(a)+\triangle(a)\,\triangle\,(b))(1+\varepsilon^*)+\texttt{MinReal} \\
=\ & \triangle(\text{mul})\ \square
\end{aligned}
$$

Lemma 2 (Inverse)

$$\bigwedge_{|\eta| \le \eta^* < 0.5} \left| \frac{1}{1+\eta} \right| \le 1 + \varepsilon(\mathrm{inv})$$

mit $\varepsilon(\mathrm{inv}) := (1 + 2\eta^*) \cdot \eta^*$.

Beweis:
Mit $|\eta| \le \eta^* < 0.5$ gilt:

$$1 - \eta^* \le \frac{1}{1+\eta} \le 1 + (1 + 2\eta^*) \cdot \eta^*.$$

Daraus folgt $\varepsilon(\mathrm{inv}) = (1 + 2\eta^*) \cdot \eta^*$ □

Satz 4 (Division) Für die Division $\circ := /$ gilt

$$\bigwedge_{\substack{a \in A \\ b \in B}} \bigwedge_{\substack{|a - \tilde{a}| \le \triangle(a) \\ |b - \tilde{b}| \le \triangle(b)}} \left| a/b - \tilde{a} \boxplus \tilde{b} \right| \le \triangle(\mathrm{div}) \quad \text{mit}$$

$$\triangle(\mathrm{div}) := \frac{1}{\langle B \rangle - \triangle(b)} \Big(\triangle(a) + (|A| + \triangle(a)) \cdot (\varepsilon^* + \varepsilon(\mathrm{inv})) \Big) + \texttt{MinReal}$$

und

$$\varepsilon(\mathrm{inv}) := (1 + 2\frac{\triangle(b)}{\langle B \rangle})\frac{\triangle(b)}{\langle B \rangle}$$

Dabei wird vorausgesetzt, daß

$$\triangle(b) < \ 0.5 \ \cdot \langle B \rangle \tag{3.3}$$

gilt.

Beweis:
Es seien $a \in A, b \in B, \tilde{a} = a + \triangle_a \in A + [-\triangle(a), \triangle(a)], \triangle_a \le \triangle(a)$ und $\tilde{b} = b + \triangle_b \in B + [-\triangle(b), \triangle(b)], \triangle_b \le \triangle(b)$ beliebig aber fest gewählt.

I) Abschätzung für $|\tilde{a}/\tilde{b} - \tilde{a} \boxplus \tilde{b}|$:

a) $\tilde{a}/\tilde{b} \in U \Longrightarrow \tilde{a} \boxplus \tilde{b} \in \overline{U} \overset{\text{Lemma 1}}{\Longrightarrow} |\tilde{a}/\tilde{b} - \tilde{a} \boxplus \tilde{b}| \le \texttt{MinReal}$

b) $\tilde{a}/\tilde{b} \notin U \Longrightarrow |\tilde{a} \boxplus \tilde{b}| \notin U$
$\Longrightarrow |\tilde{a}/\tilde{b} - \tilde{a} \boxplus \tilde{b}| \le \varepsilon^* |\tilde{a}/\tilde{b}| \le \varepsilon^*(|A| + \triangle(a)) \cdot \frac{1}{\langle B \rangle - \triangle(b)}$

Dabei wird z.B. verwendet, daß $\tilde{b} \in B + [-\triangle(b), \triangle(b)]$ und damit $\frac{1}{|\tilde{b}|} \le \frac{1}{\langle B \rangle - \triangle(b)}$ gilt.

II) Abschätzung für $|a/b - \tilde{a}/\tilde{b}|$:

$$\begin{aligned}
|a/b - \tilde{a}/\tilde{b}| &= \left| \frac{a}{b} - \frac{(a+\triangle_a)}{b} \cdot \frac{1}{1+\frac{\triangle_b}{b}} \right| \\
&= \left| \frac{a}{b} - \frac{(a+\triangle_a)}{b} \cdot (1+\varepsilon_{\text{inv}}) \right| \\
&= \frac{1}{|b|} \left| \triangle_a + (a+\triangle_a) \cdot \varepsilon_{\text{inv}} \right| \\
&\overset{\text{mit (3.3)}}{\leq} \frac{1}{\langle B \rangle} \Big(\triangle(a) + (|A| + \triangle(a)) \cdot \varepsilon(\text{inv}) \Big)
\end{aligned}$$

mit $|\varepsilon_{\text{inv}}| \leq \varepsilon(\text{inv}) := (1 + 2 \cdot \frac{\triangle(b)}{\langle B \rangle}) \cdot \frac{\triangle(b)}{\langle B \rangle}$ nach Lemma 2.

Gesamtfehlerabschätzung:

$$\begin{aligned}
& |a/b - \tilde{a} \boxplus \tilde{b}| \leq |a/b - \tilde{a}/\tilde{b}| + |\tilde{a}/\tilde{b} - \tilde{a} \boxplus \tilde{b}| \\
\overset{\text{I),II)}{\leq} & \frac{1}{\langle B \rangle} \Big(\triangle(a) + (|A| + \triangle(a)) \cdot \varepsilon(\text{inv}) \Big) + \\
& \qquad \begin{cases} \texttt{MinReal}, & \text{Fall I) a)} \\ \varepsilon^* (|A| + \triangle(a)) \cdot \frac{1}{\langle B \rangle - \triangle(b)}, & \text{Fall I) b)} \end{cases} \\
\leq & \frac{1}{\langle B \rangle > - \triangle(b)} \Big(\triangle(a) + (|A| + \triangle(a)) \cdot (\varepsilon^* + \varepsilon(\text{inv})) \Big) + \texttt{MinReal} \\
= & \triangle(\text{div}) \quad \square
\end{aligned}$$

Die hergeleiteten Fehlerabschätzungen können nun mit Hilfe von Intervallrechnung in Intervallroutinen umgesetzt werden.

Das Modul `abs_ari`
In diesem Modul werden die Sätze 1 bis 4 in ein PASCAL-XSC Programm umgesetzt. Die Quelltexte sind im Anhang B auszugsweise angegeben. Mit den in diesem Modul realisierten Funktionen `DeltaAdd`, `DeltaSub`, `DeltaMul`, `DeltaDiv` kann die Fehlerfortpflanzung automatisch erfaßt werden. Z.B. liefert der Aufruf

```
DeltaRes := DeltaAdd( A, B, DeltaA, DeltaB );
```

eine Maschinenzahl `DeltaRes` als Ergebnis, für welche gilt

$$\bigwedge_{\substack{a \in A \\ b \in B}} \bigwedge_{\substack{|a - \tilde{a}| \leq \triangle(a) \\ |b - \tilde{b}| \leq \triangle(b)}} \left| a + b - \tilde{a} \boxplus \tilde{b} \right| \leq \triangle(\text{add}) \leq \texttt{DeltaRes}$$

(vergleiche Satz 1). Dabei sind $A, B \in IS$, DeltaA, DeltaB $\in S$. Entsprechendes gilt für die anderen Funktionen.

Ein Programm zur Fehlererfassung bei der Auswertung eines Polynoms $p(x) = \sum_{i=0}^{n} p_i x^i$ mittels des Hornerschemas kann z.B. für $x \in \texttt{X} \in IS, \tilde{x} = x + \triangle_x$ mit $| \triangle_x | \leq \triangle(x) \leq \texttt{DeltaX} \in S$ wie folgt programmiert werden:

```
H:= p[n]; DeltaH:= diam( H );
for i:= n-1 downto 0 do begin
   DeltaH:= DeltaMul( H, X, DeltaH, DeltaX );
   H:= H * X;
   DeltaH:= DeltaAdd( H, p[i], DeltaH, diam( p[i] ) );
   H:= H + p[i];
end;
PolX:= H;
AbsErr:= DeltaH;
```

Nach Ausführung dieses Programmstücks gilt dann

$$|p(x) - \tilde{p}(\tilde{x})| \leq \texttt{AbsErr}$$

für jedes $x \in \texttt{X}$ und jedes $\tilde{x} = x + \triangle_x$ mit $|\triangle_x| \leq \triangle(x)$. Weiter gilt $p(x) \in \texttt{PolX}$ für jedes $x \in \texttt{X}$ und jedes $\tilde{p}(\tilde{x}) = (\ldots((\tilde{p}_n \boxdot \tilde{x} \boxplus \tilde{p}_{n-1}) \boxdot \tilde{x} \boxplus \tilde{p}_{n-2}) \boxdot \tilde{x} \boxplus \ldots \boxplus \tilde{p}_1) \boxdot \tilde{x} \boxplus \tilde{p}_0$ mit $\tilde{p}_i \in p[i], i = 0(1)n$.

Numerisches Beispiel
Betrachtet wird das Polynom $p(x) = \sum_{i=0}^{15} p_i x^i := \sum_{i=0}^{15} \frac{1}{i!} x^i$. Als Polynomkoeffizienten p_i werden die engstmöglichen Maschinenintervalleinschließungen verwendet:

```
p[ 0] := [  1.000000000000000E+000,  1.000000000000000E+000 ];
p[ 1] := [  1.000000000000000E+000,  1.000000000000000E+000 ];
p[ 2] := [  5.000000000000000E-001,  5.000000000000000E-001 ];
...
p[13] := [  1.605904383682161E-010,  1.605904383682162E-010 ];
p[14] := [  1.147074559772972E-011,  1.147074559772973E-011 ];
p[15] := [  7.647163731819816E-013,  7.647163731819818E-013 ];
```

Mit dem oben vorgestellten Programmfragment erhält man folgenden Werte:

```
x = 1:
              PolX = [  2.718281828458994E+000,  2.718281828458995E+000 ]
            AbsErr =  6.402573517656651E-016    (Fehlerschranke absolut)
AbsErr/MinAbs(PolX) =  2.355375167734649E-016    (Fehlerschranke relativ)

x = -4:
              PolX = [     1.814980943022E-002,     1.814980943024E-002 ]
            AbsErr =  1.313450654637236E-014    (Fehlerschranke absolut)
AbsErr/MinAbs(PolX) =  7.236718708736003E-013    (Fehlerschranke relativ)
```

Im Fall $x = 1$ liegt die relative Fehlerschranke nahe der Maschinengenauigkeit (es werden nur positive Größen aufsummiert). Im Fall $x = -4$ ergibt sich aufgrund alternierender Vorzeichen und eines wesentlich kleineren Polynomwertes eine um drei Zehnerpotenzen größere Fehlerschranke.

3.2 Ausnutzung des Operatorkonzepts

Die Handhabung des in Abschnitt 3.1 und in [20] beschriebenen rechnergestützten Fehlerkalküls kann unter Verwendung eines Operatorkonzeptes ganz erheblich erleichtert werden. Die Einführung des neuen Datentyps `BoundType` gemäß

```
global type BoundType = global record  { Neuer Datentyp                    }
              Enclosure: interval;     { Einschliessung der korrekten Werte }
              AbsErr:    real;         { Zugehoeriger max. absoluter Fehler }
            end;
```

erlaubt das Überladen der Grundoperationen und Funktionen für diesen Datentyp. Z. B. kann dies in PASCAL-XSC [24] für die Addition durch die Anweisungsfolge

```
global operator + (x, y: BoundType) erg: BoundType;
begin
  erg.AbsErr    := DeltaAdd(x.Enclosure, y.Enclosure, x.AbsErr, y.AbsErr);
  erg.Enclosure := x.Enclosure + y.Enclosure;
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 `BoundType` und Einfügen einiger Schreibanweisungen durchgeführt werden.

3.3 Auslöschung führender Ziffern

Bei der Subtraktion zweier fehlerbehafteter Gleitkommazahlen $\tilde{a}$ und $\tilde{b}$ mit gleichem Vorzeichen und mit $\tilde{a} \approx \tilde{b}$ löschen sich u.U. führende Ziffern aus. Eine Oberschranke $\triangle(c)$ für den absoluten Fehler von $\tilde{c} := \tilde{a} \boxminus \tilde{b}$ wird vom verwendeten Fehlerkalkül automatisch richtig erfaßt.

Ein gerade bei Funktionsberechnungen wichtiger Sonderfall der Subtraktion ist, daß bei exakten, d.h. ungestörten Eingangsgrößen bei der Ergebnisberechnung Auslöschung stattfindet. Der folgende Satz gibt hierüber Auskunft.

Satz 5 $a, b, c \in S(2, 53)$, $a =: s_a \cdot m_a \cdot 2^{e_a}$ und $b =: s_b \cdot m_b \cdot 2^{e_b}$ seien normalisierte Gleitkommazahlen (d.h. $1 \leq m_a, m_b < 2$, $s_a, s_b \in \{-1, +1\}$, $e_a, e_b \in \mathbb{Z}$) mit gleichem Vorzeichen (d.h. $s_a = s_b$), weiter seien a und b exakt, d.h. nicht fehlerbehaftet gegeben. Mit $c := a \boxminus b =: s_c \cdot m_c \cdot 2^{e_c}$ gilt:

a) $e_a = e_b \Longrightarrow c = a - b$

b) $|e_a - e_b| = 1, e_c < \max\{e_a, e_b\} \Longrightarrow c = a - b$

c) $|e_a - e_b| \geq 2, e_c \leq \min\{e_a, e_b\} \Longrightarrow c = a - b$

In allen drei Fällen hat eine Auslöschung führender Ziffern stattgefunden, das Ergebnis kann in $S(2, 53)$ rundungsfehlerfrei berechnet werden.

Satz 6 Die Aussagen von Satz 5 gelten analog für die Addition von Gleitkommazahlen mit unterschiedlichem Vorzeichen.

Beweis:
Die Addition $c := a \boxplus b$ kann durch eine rundungsfehlerfreie Vorzeichenänderung in eine Subtraktion übergeführt werden $c = a \boxminus (-b)$. □

Anmerkung: Bei den in dieser Dokumentation beschriebenen Fehlerabschätzungen wurden die Aussagen der obigen Sätze z.T. noch manuel berücksichtigt. In den neueren Implementierungen des Moduls `abs_ari` werden die Voraussetzungen dieser Sätze automatisch abgeprüft und die entsprechenden Aussagen berücksichtigt. Siehe hierzu auch die Hinweise im Anhang.

3.4 Funktionen mit fehlerbehaftetem Argument

Im folgenden wird davon ausgegangen, daß für die Funktion f und ihre Maschinenrealisierung $\tilde{f}$ bei Anwendung auf exakt darstellbare Argumente eine relative Fehlerschranke $\varepsilon(f)$ bekannt ist, die relative Schranke $\varepsilon(f)$ gilt natürlich nur bei Anwendung auf die Argumente $x \in S$, für die $f(x) \in S_N$ bzw. $\tilde{f}(x) \in S_N$ gilt[1], d.h. der berechnete Funktionswert liegt im Bereich der normalisierten Gleitkommazahlen. Falls Funktionswerte im Bereich der denormalisierten Gleitkommazahlen liegen können, wird davon ausgegangen, daß für alle exakt darstellbaren Argumente $x \in S$ mit $f(x) \in S_D$ bzw. $\tilde{f}(x) \in S_D$ eine absolute Fehlerschranke $\triangle(f)$ bekannt ist.

Der Definitionsbereich der Funktion f sei $D_f \subseteq I\!R$, der Definitionsbereich der Maschinenrealisierung $\tilde{f}$ sei $D_{\tilde{f}}$ mit $D_{\tilde{f}} \subseteq D_f \cap S$.

Gesucht wird nun eine Oberschranke $\triangle(\mathrm{Res})$ für den maximalen Betrag des absoluten Fehlers, der bei Anwendung von $\tilde{f}$ auf ein bereits fehlerbehaftetes Argument $\tilde{a} := a + \triangle_a$ mit $| \triangle_a | \leq \triangle(a)$ entstehen kann, d.h.

$$|f(a) - \tilde{f}(a)| \leq \triangle_{\mathrm{Res}}, \quad | \triangle_{\mathrm{Res}} | \leq \triangle(\mathrm{Res}) = ?.$$

Satz 7 Es seien $a \in A \in IS, \varepsilon(f), \triangle(f), \triangle(a) \in I\!R^+, \tilde{a} = a + \triangle_a \in S$ mit $| \triangle_a | \leq \triangle(a)$. Weiter sei die Funktion

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

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

$$|f(a) - \tilde{f}(a)| \leq |f(a)|\varepsilon(f) \tag{3.4}$$

erfüllt sein. Weiter gelte für alle darstellbaren Argumente $a \in A + [-\triangle(a),\ \triangle(a)] \cap S$ mit $f(a) \in S_D$ die absolute Fehlerschranke $\triangle(f)$, d. h., es möge für alle zulässigen a die Abschätzung

$$|f(a) - \tilde{f}(a)| \leq \triangle(f) \tag{3.5}$$

[1]Mit S_N wird die Menge der normalisierten Gleitkommazahlen, mit S_D wird die Menge der denormalisierten Gleitkommazahlen bezeichnet, es gilt $S = S_N \cup S_D$.

erfüllt sein. Dann gilt die Darstellung
a)

$$f(\tilde{a}) \;=\; f(a) + \triangle_a f'(a + \Theta\triangle_a) \quad \text{mit} \quad \Theta = \Theta(a,\tilde{a}) \in (0,1) \tag{3.6}$$

Beweis:

Zu a): Für die Funktion f sind bezüglich des Intervalls $A + [-\triangle(a),\ \triangle(a)]$ die Voraussetzungen des Mittelwertsatzes der Differentialrechnung erfüllt. Aus diesem folgt unmittelbar die Darstellung (3.6).

□

Satz 8 Mit den Bezeichnungen von Satz 7 und den dort angegebenen Voraussetzungen gilt für die Funktionsauswertung die Abschätzung

$$\bigwedge_{a \in A} \bigwedge_{\substack{\tilde{a} \in S \\ |a-\tilde{a}| \le \triangle(a)}} |f(a) - \tilde{f}(\tilde{a})| \le \triangle(\mathrm{Res}) \quad \text{mit}$$

$$\triangle(\mathrm{Res}) := \triangle(f) + \varepsilon(f)\cdot|f(A)| + (1+\varepsilon(f))\cdot\triangle(a)\cdot|f'(A + [-\triangle(a), \triangle(a)])|.$$

Beweis:

I) Abschätzung für $a \in S$ mit $f(a) \in S_N$:
Mit Satz 7 gilt

$$|\varepsilon_f \cdot f(\tilde{a})| \le \varepsilon(f)\cdot|f(A)| + \varepsilon(f)\cdot\triangle(a)\cdot|f'(A + [-\triangle(a), \triangle(a)])|$$

und damit

$$\begin{aligned}
& |f(a) - \tilde{f}(\tilde{a})| \\
\le\ & |f(a) - f(\tilde{a})\cdot(1+\varepsilon_f)| \\
\le\ & |f(a) - f(\tilde{a})| + |\varepsilon_f \cdot f(\tilde{a})| \\
\le\ & \triangle(a)\cdot|f'(A + [-\triangle(a), \triangle(a)])| \\
& + \varepsilon(f)\cdot|f(A)| + \varepsilon(f)\cdot\triangle(a)\cdot|f'(A + [-\triangle(a), \triangle(a)])| \\
=\ & \varepsilon(f)\cdot|f(A)| + (1+\varepsilon(f))\cdot\triangle(a)\cdot|f'(A + [-\triangle(a), \triangle(a)])|
\end{aligned}$$

II) Abschätzung für $a \in S$ mit $f(a) \in S_D$:

$$\begin{aligned}
|f(a) - \tilde{f}(\tilde{a})| \;&\le\; |f(a) - (f(\tilde{a}) + \triangle_f)| \\
&\le\; |f(a) - f(\tilde{a})| + \triangle(f) \\
&\le\; \triangle(f) + \triangle(a)\cdot|f'(A + [-\triangle(a), \triangle(a)])|
\end{aligned}$$

Gesamtfehlerabschätzung für $x \in S$ und $f(x) \in S_N \cup S_D = S$:

$$\begin{array}{ll} & |f(a) - \tilde{f}(\tilde{a})| \\ \overset{\text{I),II)}{\leq} & \begin{cases} \varepsilon(f) \cdot |f(A)| + (1 + \varepsilon(f)) \cdot \triangle(a) \cdot |f'(A + [-\triangle(a), \triangle(a)])|, & \text{Fall I)} \\ \triangle(f) + \triangle(a) \cdot |f'(A + [-\triangle(a), \triangle(a)])|, & \text{Fall II)} \end{cases} \\ \leq & \triangle(f) + \varepsilon(f) \cdot |f(A)| + (1 + \varepsilon(f)) \cdot \triangle(a) \cdot |f'(A + [-\triangle(a), \triangle(a)])| \end{array}$$

3.4.1 Die Wurzelfunktion (absoluter Fehler)

Betrachtet wird die Funktion $f(x) := \sqrt{x}$ mit $D_f := [0, \infty)$, sowie die Maschinenrealisierung $\tilde{f}(x) := \texttt{sqrt}(x)$ mit $D_{\tilde{f}} := [0, \texttt{maxreal}]$.

Die Funktion f ist differenzierbar, es gilt

$$f'(x) = \frac{1}{2\sqrt{(x)}}.$$

Weiter ist

$$D_N := \{x \in S | f(x) \in S_N\} = [\texttt{dminreal}, \texttt{maxreal}] \cap S$$

und

$$D_D := \{x \in S | f(x) \in S_D\} = [0, 0].$$

Die Auswertung für $x := 0$ erfolgt fehlerfrei, d.h. es gilt $\triangle(\sqrt{\ }) = 0$.

Anwendung von Satz 8 ergibt:

$$|\sqrt{a} - \texttt{sqrt}(\tilde{a})| \leq \triangle(\text{Res}) \quad \text{mit}$$

$$\triangle(\text{Res}) := \varepsilon(\sqrt{\ }) \cdot |\sqrt{A}| + (1 + \varepsilon(\sqrt{\ })) \cdot \triangle(a) \cdot \frac{1}{2 \cdot \langle A + [-\triangle(a), \triangle(a)] \rangle}.$$

Diese Abschätzung ist allerdings nur für $a \geq \triangle(a)$ gültig, dies wird bei der Umsetzung in das Modul `abs_ari` abgeprüft.

3.4.2 Die Exponentialfunktion (absoluter Fehler)

Betrachtet wird die Funktion $f(x) := e^x$ mit $D_f := (-\infty, \infty)$, sowie die Maschinenrealisierung $\tilde{f}(x) := \texttt{exp}(x)$ mit $D_{\tilde{f}} := [-\texttt{maxreal}, \texttt{expmax}]$.

Die Funktion f ist differenzierbar, es gilt

$$f'(x) = e^x.$$

Weiter ist

$$D_N := \{x \in S | f(x) \in S_N\} = (\ln(\texttt{minreal}), \texttt{expreal}] \cap S$$

und

$$D_D := \{x \in S | f(x) \in S_D\} = [-\texttt{maxreal}, \ln(\texttt{minreal})] \cap S.$$

Anwendung von Satz 8 ergibt:

$$|e^a - \texttt{exp}(\tilde{a})| \leq \triangle(\text{Res}) \quad \text{mit}$$

$$\triangle(\text{Res}) := \triangle(\texttt{exp}) + \varepsilon(\texttt{exp}) \cdot |e^A| + (1 + \varepsilon(\texttt{exp})) \cdot \triangle(a) \cdot |e^{A + [-\triangle(a), \triangle(a)]}|.$$

3.4.3 Die Funktion $e^x - 1$ (absoluter Fehler)

Betrachtet wird die Funktion $f(x) := e^x - 1$ mit $D_f := (-\infty, \infty)$, sowie die Maschinenrealisierung $\tilde{f}(x) := \texttt{expm1}(x)$ mit $D_{\tilde{f}} := [-\texttt{maxreal}, \texttt{expm1max}]$.

Die Funktion f ist differenzierbar, es gilt

$$f'(x) = e^x.$$

Weiter ist

$$\begin{aligned} D_N &:= \{x \in S | f(x) \in S_N\} \\ &= ([-\texttt{maxreal}, \ln(1 - \texttt{minreal})) \cup (\ln(1 + \texttt{minreal}), \texttt{expreal}]) \cap S \end{aligned}$$

und

$$D_D := \{x \in S | f(x) \in S_D\} = [\ln(1 - \texttt{minreal}), \ln(1 + \texttt{minreal})] \cap S.$$

Anwendung von Satz 8 ergibt:

$$|(e^a - 1) - \texttt{expm1}(\tilde{a})| \leq \triangle(\text{Res}) \quad \text{mit}$$

$$\triangle(\text{Res}) := \triangle(\texttt{expm1}) + \varepsilon(\texttt{expm1}) \cdot |e^A - 1| + (1 + \varepsilon(\texttt{expm1})) \cdot \triangle(a) \cdot |e^{A + [-\triangle(a), \triangle(a)]}|.$$

3.4.4 Die Logarithmusfunktion (absoluter Fehler)

Betrachtet wird die Funktion $f(x) := \ln x$ mit $D_f := (0, \infty)$, sowie die Maschinenrealisierung $\tilde{f}(x) := \texttt{ln}(x)$ mit $D_{\tilde{f}} := (0, \texttt{maxreal}]$.

Die Funktion f ist differenzierbar, es gilt

$$f'(x) = \frac{1}{x}.$$

Weiter ist

$$\begin{aligned} D_N &:= \{x \in S | f(x) \in S_N\} \\ &= (0, e^{\texttt{-minreal}}) \cup (e^{\texttt{minreal}}, \texttt{maxreal}]) \cap S \end{aligned}$$

und

$$D_D := \{x \in S | f(x) \in S_D\} = [e^{\texttt{-minreal}}, e^{\texttt{minreal}}] \cap S.$$

Anwendung von Satz 8 ergibt:

$$|\ln a - \texttt{ln}(\tilde{a})| \leq \triangle(\text{Res}) \quad \text{mit}$$

$$\triangle(\text{Res}) := \triangle(\texttt{ln}) + \varepsilon(\texttt{ln}) \cdot |\ln A| + (1 + \varepsilon(\texttt{ln})) \cdot \triangle(a) \cdot \frac{1}{\langle A + [-\triangle(a), \triangle(a)] \rangle}.$$

Diese Abschätzung ist allerdings nur für $a \geq \triangle(a)$ gültig, dies wird bei der Umsetzung in das Modul `abs_ari` abgeprüft.

3.4.5 Die Funktion $\ln(x+1)$ (absoluter Fehler)

Betrachtet wird die Funktion $f(x) := \ln(x+1)$ mit $D_f := (-1, \infty)$, sowie die Maschinenrealisierung $\tilde{f}(x) := \texttt{ln1p}(x)$ mit $D_{\tilde{f}} := (-1, \texttt{maxreal}]$.

Die Funktion f ist differenzierbar, es gilt

$$f'(x) = \frac{1}{x+1}.$$

Weiter ist

$$\begin{aligned} D_N &:= \{x \in S | f(x) \in S_N\} \\ &= (-1, e^{\texttt{-minreal}} - 1) \cup (e^{\texttt{minreal}} - 1, \texttt{maxreal}]) \cap S \end{aligned}$$

und

$$D_D := \{x \in S | f(x) \in S_D\} = [e^{\texttt{-minreal}} - 1, e^{\texttt{minreal}} - 1] \cap S.$$

Anwendung von Satz 8 ergibt:

$$|\ln(a+1) - \texttt{ln1p}(\tilde{a})| \leq \triangle(\text{Res}) \quad \text{mit}$$

$$\begin{aligned} \triangle(\text{Res}) &:= \triangle(\texttt{ln1p}) + \varepsilon(\texttt{ln1p}) \cdot |\ln(A+1)| \\ &\quad + (1 + \varepsilon(\texttt{ln1p})) \cdot \triangle(a) \cdot \frac{1}{1 + \langle A + [-\triangle(a), \triangle(a)]\rangle}. \end{aligned}$$

3.4.6 Die Arkustangensfunktion (absoluter Fehler)

Betrachtet wird die Funktion $f(x) := \arctan x$ mit $D_f := (-\infty, \infty)$, sowie die Maschinenrealisierung $\tilde{f}(x) := \texttt{arctan}(x)$ mit $D_{\tilde{f}} := (-\texttt{maxreal}, \texttt{maxreal}]$.

Die Funktion f ist differenzierbar, es gilt

$$f'(x) = \frac{1}{1+x^2}.$$

Weiter ist

$$\begin{aligned} D_N &:= \{x \in S | f(x) \in S_N\} \\ &= (-\texttt{maxreal}, \tan(-\texttt{minreal})) \cup (\tan(\texttt{minreal}), \texttt{maxreal}]) \cap S \end{aligned}$$

und

$$D_D := \{x \in S | f(x) \in S_D\} = [\tan(-\texttt{minreal}), \tan(\texttt{minreal})] \cap S.$$

Anwendung von Satz 8 ergibt:

$$|\arctan a - \texttt{arctan}(\tilde{a})| \leq \triangle(\text{Res}) \quad \text{mit}$$

$$\begin{aligned} \triangle(\text{Res}) &:= \triangle(\texttt{arctan}) + \varepsilon(\texttt{arctan}) \cdot |\arctan A| \\ &\quad + (1 + \varepsilon(\texttt{arctan})) \cdot \triangle(a) \cdot \frac{1}{1 + \langle (A + [-\triangle(a), \triangle(a)])^2 \rangle}. \end{aligned}$$

3.4.7 Die hyperbolische Cotangensfunktion (absoluter Fehler)

Betrachtet wird die Funktion $f(x) := \coth x$ mit $D_f := (-\infty, 0) \cup (0, \infty)$, sowie die Maschinenrealisierung $\tilde{f}(x) := \texttt{coth}(x)$ mit $D_{\tilde{f}} := ([-\texttt{maxreal}, \texttt{-cotmin}] \cup (\texttt{cotmin}, \texttt{maxreal})) \cap S$. Die Funktion f ist differenzierbar, es gilt

$$f'(x) = \frac{1}{(\sinh x)^2}.$$

Weiter ist

$$\begin{aligned} D_N &:= \{x \in S | f(x) \in S_N\} \\ &= D_{\tilde{f}} \end{aligned}$$

und

$$D_D := \{\}$$

Anwendung von Satz 8 ergibt:

$$|\coth a - \texttt{coth}(\tilde{a})| \leq \triangle(\text{Res}) \quad \text{mit}$$

$$\begin{aligned} \triangle(\text{Res}) &:= \varepsilon(\texttt{coth}) \cdot |\coth A| \\ &\quad + (1 + \varepsilon(\texttt{coth})) \cdot \triangle(a) \cdot \frac{1}{\langle(\sinh(A + [-\triangle(a), \triangle(a)]))^2\rangle}. \end{aligned}$$

3.4.8 Die Funktion $\ln(x)$ (relativer Fehler)

Gesucht ist eine Oberschranke für den relativen Fehler

$$\left|\frac{\ln(x) - \tilde{\ln}(\tilde{x})}{\ln(x)}\right| \leq ?,$$

für das Argument x wird dabei $x \geq \gamma > 1$ vorausgesetzt und es sei $\tilde{x} := x(1 + \varepsilon_x)$ mit $|\varepsilon_x| \leq \varepsilon(x) < 1$.

Zunächst wird eine Fehlerschranke für die Auswertung des Logarithmus mit fehlerbehaftetem Argument hergeleitet:

$$\begin{aligned} \left|\frac{\ln(x) - \ln(\tilde{x})}{\ln(x)}\right| &= \left|\frac{\ln(x) - \ln(x + x\varepsilon_x)}{\ln(x)}\right| \\ &\overset{\text{MWS}}{=} \left|\frac{\ln(x) - (\ln(x) + x\varepsilon_x \cdot \frac{1}{x + x\varepsilon_x})}{\ln(x)}\right| \\ &= \left|\frac{x\varepsilon_x}{(x + x\varepsilon_x) \cdot \ln(x)}\right| \\ &= \left|\frac{\varepsilon_x}{(1 + \varepsilon_x) \cdot \ln(x)}\right| \\ &\leq \varepsilon(x) \cdot \frac{1}{1 - \varepsilon(x)} \cdot \frac{1}{\ln(\gamma)} \end{aligned}$$

Mit $\varepsilon_{\ln}$ wird der relative Fehler bei der Auswertung der Logarithmusfunktion für exakt darstellbare Argumente bezeichnet, es gilt $|\varepsilon_{\ln}| \leq \varepsilon(\ln)$. Damit läßt sich die folgende Abschätzung durchführen:

$$\begin{aligned}\left|\frac{\varepsilon_{\ln}\cdot\ln(\tilde{x})}{\ln(x)}\right| &\overset{\text{MWS}}{=} \left|\varepsilon_{\ln}\cdot\left(1+\frac{x\varepsilon_x}{(x+x\varepsilon_x)\cdot\ln(x)}\right)\right| \\ &\leq \varepsilon(\ln)\cdot\left(1+\varepsilon(x)\cdot\frac{1}{1-\varepsilon(x)}\cdot\frac{1}{\ln(\gamma)}\right)\end{aligned}$$

Mit diesen beiden Abschätzungen kann jetzt eine Oberschranke für den Gesamtfehler angegeben werden:

$$\begin{aligned}\left|\frac{\ln(x)-\tilde{\ln}(\tilde{x})}{\ln(x)}\right| &\leq \left|\frac{\ln(x)-\ln(\tilde{x})(1+\varepsilon_{\ln})}{\ln(x)}\right| \\ &\leq \left|\frac{\ln(x)-\ln(\tilde{x})}{\ln(x)}\right| + \left|\frac{\varepsilon_{\ln}\cdot\ln(\tilde{x})}{\ln(x)}\right| \\ &\leq \varepsilon(x)\cdot\frac{1}{1-\varepsilon(x)}\cdot\frac{1}{\ln(\gamma)} + \varepsilon(\ln)\cdot\left(1+\varepsilon(x)\cdot\frac{1}{1-\varepsilon(x)}\cdot\frac{1}{\ln(\gamma)}\right) \\ &\leq \varepsilon(\ln)+(1+\varepsilon(\ln))\cdot\varepsilon(x)\cdot\frac{1}{1-\varepsilon(x)}\cdot\frac{1}{\ln(\gamma)}\end{aligned}$$

3.4.9 Die Funktion $\mathbf{ln1p}(x) := \ln(x+1)$ (relativer Fehler)

Gesucht ist eine Oberschranke für den relativen Fehler

$$\left|\frac{\mathrm{ln1p}(x)-\tilde{\mathrm{ln1p}}(\tilde{x})}{\mathrm{ln1p(x)}}\right| \leq ?,$$

für das Argument x wird dabei $0 < x < 1$ angenommen und es sei $\tilde{x} := x(1+\varepsilon_x)$ mit $|\varepsilon_x| \leq \varepsilon(x) < 1$.

Zunächst wird eine Fehlerschranke für die Funktionsauswertung mit fehlerbehaftetem Argument hergeleitet:

$$\begin{aligned}\left|\frac{\mathrm{ln1p}(x)-\mathrm{ln1p}(\tilde{x})}{\mathrm{ln1p}(x)}\right| &= \left|\frac{\mathrm{ln1p}(x)-\mathrm{ln1p}(x+x\varepsilon_x)}{\mathrm{ln1p}(x)}\right| \\ &\overset{\text{MWS}}{=} \left|\frac{\mathrm{ln1p}(x)-(\mathrm{ln1p}(x)+x\varepsilon_x\cdot\frac{1}{1+x+x\varepsilon_x})}{\mathrm{ln1p}(x)}\right| \\ &= \left|\frac{x\varepsilon_x}{(1+x+x\varepsilon_x)\cdot(x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+-\ldots)}\right| \\ &= \left|\frac{\varepsilon_x}{(1+x+x\varepsilon_x)\cdot(1-\frac{x}{2}+\frac{x^2}{3}-\frac{x^3}{4}+-\ldots)}\right| \\ &\leq \varepsilon(x)\cdot\frac{1}{1-\varepsilon(x)}\cdot\frac{1}{1-\frac{x}{2}} \\ &\leq \varepsilon(x)\cdot\frac{2}{1-\varepsilon(x)}\end{aligned}$$

Mit $\varepsilon_{\ln 1p}$ wird der relative Fehler bei der Auswertung der Funktion $\ln 1p(x)$ für exakt darstellbare Argumente bezeichnet, es gilt $|\varepsilon_{\ln 1p}| \leq \varepsilon(\ln 1p)$. Damit läßt sich die folgende Abschätzung herleiten:

$$\begin{aligned}\left|\frac{\varepsilon_{\ln 1p} \cdot \ln 1p(\tilde{x})}{\ln 1p(x)}\right| &\overset{\text{MWS}}{=} \left|\varepsilon_{\ln 1p} \cdot (1 + \frac{x\varepsilon_x}{(1+x+x\varepsilon_x)\cdot \ln 1p(x)})\right| \\ &\leq \varepsilon(\ln 1p) \cdot (1 + \varepsilon(x) \cdot \frac{2}{1-\varepsilon(x)})\end{aligned}$$

Mit diesen beiden Abschätzungen kann jetzt eine Oberschranke für den Gesamtfehler angegeben werden:

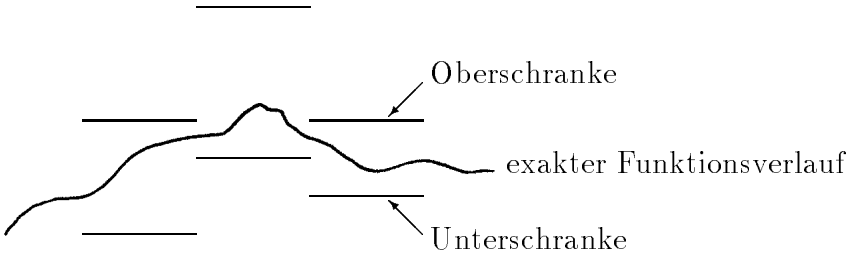
$$\begin{aligned}\left|\frac{\ln 1p(x) - \tilde{\ln 1p}(\tilde{x})}{\ln 1p(x)}\right| &\leq \left|\frac{\ln 1p(x) - \ln 1p(\tilde{x})(1+\varepsilon_{\ln 1p})}{\ln 1p(x)}\right| \\ &\leq \left|\frac{\ln 1p(x) - \ln 1p(\tilde{x})}{\ln 1p(x)}\right| + \left|\frac{\varepsilon_{\ln 1p}\cdot \ln 1p(\tilde{x})}{\ln 1p(x)}\right| \\ &\leq \varepsilon(x)\cdot\frac{2}{1-\varepsilon(x)} + \varepsilon(\ln 1p)\cdot(1+\varepsilon(x)\cdot\frac{2}{1-\varepsilon(x)}) \\ &\leq \varepsilon(\ln 1p) + (1+\varepsilon(\ln 1p))\cdot\varepsilon(x)\cdot\frac{2}{1-\varepsilon(x)}\end{aligned}$$

3.5 Verläßliche Approximationsgenauigkeit

In diesem Abschnitt wird eine in [34] ausführlich beschriebene Methode verwendet, die es erlaubt, bei gegebenen Approximationskoeffizienten (i.a. ist dies ein Satz von Maschinenzahlen, welcher aus einem Satz von Langzahlwerten gewonnen wird) eine sichere Oberschranke für den absoluten/relativen maximalen Approximationsfehler zu bestimmen.

Zunächst wird mit Hilfe von Langzahlintervallrechnung (siehe [32]) ein Intervallpolynom berechnet, dessen Graphenmenge die tatsächliche Fehlerkurve mit Sicherheit einschließt. In einem zweiten Schritt werden für dieses Intervallpolynom, dessen Intervallkoeffizienten i.a. Maschinenintervalle von wenigen ulp Durchmesser sind, mittels gewöhnlicher Intervallrechnung (z.B. mittels eines Branch-and-Bound-Verfahrens [36]) die betragsgrößten Funktionswerte sicher eingeschlossen. Das Maximum der Oberschranken dieser Einschließungen ist dann eine sichere obere Schranke des Approximationsfehlers. Diese Schranke gilt für **alle** Argumente im vorgegebenen Approximationsintervall.

In [34] ist ebenfalls beschrieben, wie man den Graph des Approximationsfehlers mittels Treppenfunktionen einschließen kann. Die folgenden Abbildungen sind mit diesem Verfahren erzeugt. In jeder Spalte, die einer Bildpunktbreite entspricht, sind zwei Begrenzungspixel gezeichnet. Der wahre Verlauf der Fehlerkurve befindet sich für alle Argumente, die einer solchen Spalte zugeordnet sind, mit Sicherheit zwischen diesen beiden Begrenzungspunkten. Die Treppenstufen sind sehr klein gewählt, was den Eindruck einer stetigen Funktion vermittelt. Das trifft jedoch **nicht** zu. Vielmehr steht jeder gezeichnete Bildpunkt für eine der Spaltenbreite entsprechenden stückweise konstanten Funktion. Die folgende Vergrößerung soll dies noch einmal verdeutlichen:



Im vorliegenden Fall der Implementierung der Funktion $e^x - 1$ sind die beiden Funktionen

$$f(x) := \frac{e^x - 1 - x}{x^2} = \sum_{k=0}^{\infty} \frac{x^k}{(k+2)!}, \quad x \in [-\frac{\ln(2)}{64}, \frac{\ln(2)}{64}] =: \text{Bereich I} \tag{3.7}$$

$$g(x) := \frac{e^x - 1 - x - \frac{x^2}{2}}{x^3} = \sum_{j=0}^{\infty} \frac{x^j}{(j+3)!}, \quad x \in [\ln(\frac{3}{4}), \ln(\frac{5}{4})] =: \text{Bereich II} \tag{3.8}$$

polynomial zu approximieren.

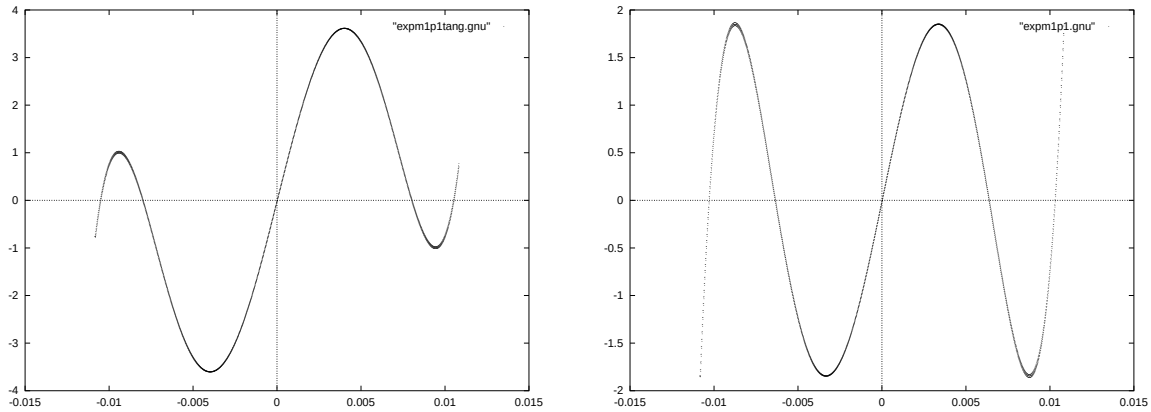


Abbildung 3.1: Fehlerkurven Bereich I

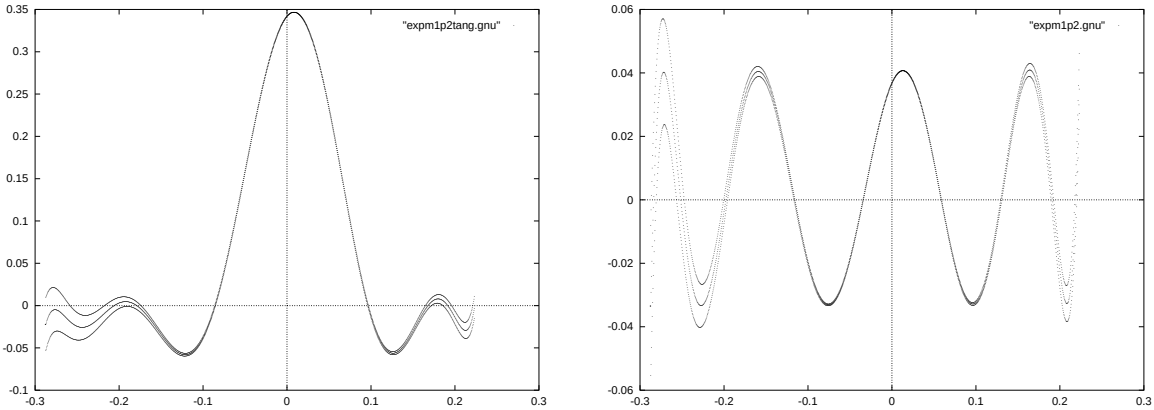


Abbildung 3.2: Fehlerkurven Bereich II

Die Abbildungen 3.1 und 3.2 zeigen jeweils links den Fehlerverlauf der Approximation, wie sie in [39] angegeben ist. Rechts ist der Fehlerverlauf einer mit Hilfe des Computeralgebrapakets Maple [19] berechneten Approximation abgebildet. Der maximale Fehler ist hier deutlich kleiner, so daß diese Approximation in der hier besprochenen Implementierung von $e^x - 1$ Verwendung findet. Man beachte, daß die Funktionswerte mit `PlotScale` $= 10^{15}$ skaliert sind (Programmlisting per FTP erhältlich). Als Schranken für die maximalen Approximationsfehler findet man

```
Maxi:  1.850454976079262E-015
```

im Fall (3.7) und

```
Maxi:  4.101904694867334E-017
```

im Fall (3.8). Dabei wird $f(x)$ approximiert durch $\sum_{k=0}^{4} a_k x^k$ mit

	hexadezimal	dezimal
a0	3FE0000000000000	5.000000000000000E-001
a1	3FC555555554DD45	1.666666666658136E-001
a2	3FA555555554B94D	4.166666666638950E-002
a3	3F811114F8A77AAA	8.333362425159880E-003
a4	3F56C1718E0F9DDC	1.388893979532449E-003

und $g(x)$ durch $\sum_{j=0}^{8} b_j x^j$ mit

	hexadezimal	dezimal
b0	3FC5555555555554	1.666666666666666E-001
b1	3FA5555555555503	4.166666666666610E-002
b2	3F81111111113FE1	8.333333333354122E-003
b3	3F56C16C16CA7FF7	1.388888889017890E-003
b4	3F2A01A0159D7CFF	1.984126964158297E-004
b5	3EFA019F817DAFAE	2.480157863209126E-005
b6	3EC71E05122BF5CB	2.755792722352050E-006
b7	3E928240725839F5	2.758025508816736E-007
b8	3E5A496317DE7DCF	2.448136759253856E-008

Im späteren Programm werden die hier hexadezimal angegebenen Koeffizienten verwendet.

Die mittleren Kurven in Abbildung 3.2 ergeben sich, wenn man zusätzlich zu den Begrenzungsbildpunkten noch einen Funktionswert der Fehlerkurve an einer zufällig gewählten Stelle in jedem einer Spalte entsprechenden Abszissenintervall einzeichnet. Die hier berechneten Oberschranken für die absoluten Approximationsfehler werden in Abschnitt 5.1.4 benötigt.

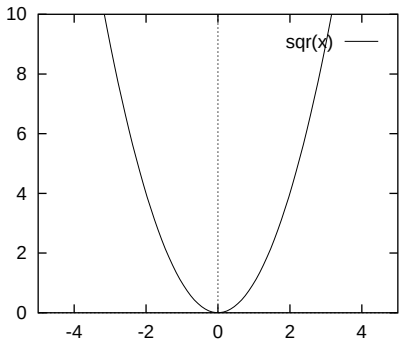
Kapitel 4

Quadrat- und Wurzelfunktion

Sehr einfache Algorithmen ergeben sich für Quadrat- und Wurzelfunktion. Diese dienen auch als Hilfsfunktion für viele andere Routinen.

4.1 Quadratfunktion

Quadratfunktion	
Math. Schreibweise	x^2
Definitionsbereich	$\mathbb{R}$
1. Ableitung	$2x$
Nullstellen	$x_0 = 0$
Minima	$x_T = 0$
Symmetrie	Symmetrie zur y-Achse
Monotonie	monoton fallend für $x < 0$ monoton steigend für $x > 0$



4.1.1 Idee

Die Quadratfunktion wird durch Ausführen einer Multiplikation $x^2 = x \cdot x$ berechnet. Bei der Intervallfunktion sind einige zusätzliche Fallunterscheidungen nötig.

4.1.2 Algorithmus `q_sqr` (Punktfunktion)

Der Gesamtalgorithmus `q_sqr` zur Berechnung von x^2 stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x < -1.34 \ldots e154$ oder $x > 1.34 \ldots e154$ (Test des Definitionsbereiches) $\Longrightarrow$ Fehlermeldung: Overflow

II. Berechnung mit Hilfe der Multiplikation
res $:= x \boxdot x$

Nun gilt `q_sqr` := res $\approx x^2$.

4.1.3 Fehlerabschätzung

Es wird genau eine Grundoperation mit entsprechendem Rundungsfehler ausgeführt.

Gesamtfehlerschranke:
relative Fehlerschranke für `q_sqr`$(x) \in S_N$: $\varepsilon(\texttt{q\_sqr}) \leq 2.2205E-016 = 2.00 \cdot \varepsilon^*$
absolute Fehlerschranke für `q_sqr`$(x) \in S_D$: $\triangle(\texttt{q\_sqr}) \leq 4.9497E-324$

4.1.4 Algorithmus `j_sqr` (Intervallfunktion)

Der Gesamtalgorithmus `j_sqr` zur Berechnung von X^2 für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x.INF oder x.SUP ist NaN (not a number)
  $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x.\text{INF} < -1.34\ldots e154$ oder $x.\text{SUP} > 1.34\ldots e154$ (Test des Definitionsbereiches) $\Longrightarrow$ Fehlermeldung: Overflow

II. Berechnung mit Hilfe der Multiplikation

a) Falls x.INF $= 0$:
res.INF:= 0
res.SUP:= succ(x.SUP $\boxdot$ x.SUP)

b) Falls x.INF > 0:
res.INF:= pred(x.INF $\boxdot$ x.INF)
res.SUP:= succ(x.SUP $\boxdot$ x.SUP)

c) Falls x.SUP $= 0$:
res.INF:= 0
res.SUP:= succ(x.INF $\boxdot$ x.INF)

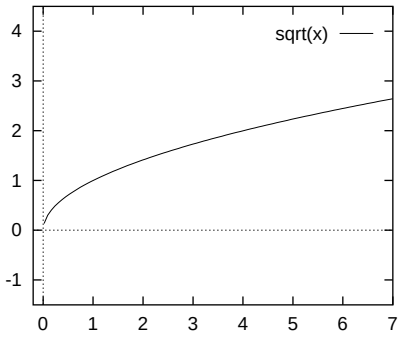
d) Falls x.SUP < 0:
res.INF:= pred(x.SUP $\boxdot$ x.SUP)
res.SUP:= succ(x.INF $\boxdot$ x.INF)

e) Sonst:
res.INF:= 0
Falls $(-x.\text{INF} > x.\text{SUP})$ dann: res.SUP:= succ(x.INF $\boxdot$ x.INF)
Sonst: res.SUP:= succ(x.SUP $\boxdot$ x.SUP)

Nun gilt `j_sqr` := res $\supseteq X^2$ mit res := [res.INF, res.SUP].

4.2 Wurzelfunktion

Wurzelfunktion	
Math. Schreibweise	$\sqrt{x}$
Definitionsbereich	$[0, \infty)$
1. Ableitung	$\frac{1}{2\sqrt{x}}$
Nullstellen	$x_0 = 0$
Minima	$x_T = 0$
Monotonie	monoton steigend



4.2.1 Idee

Im IEEE-Standard [4] wird gefordert, daß die Wurzelfunktion als Grundoperation mit den entsprechenden Rundungen zur Verfügung gestellt wird. Aus diesem Grund wird hier die Wurzelfunktion `sqrt` verwendet, die über die mathematische Bibliothek (`libm`, zugehöriges Headerfile `math.h`) des C-Compilers zur Verfügung gestellt wird.

Anmerkung: Der Benutzer muß somit sicherstellen, daß die auf seinem System vorhandene Wurzelfunktion dem IEEE-Standard entspricht (üblicherweise ist dies heute der Fall).

Bei der Intervallfunktion sind wiederum einige zusätzliche Fallunterscheidungen nötig.

4.2.2 Algorithmus `q_sqrt` (Punktfunktion)

Der Gesamtalgorithmus `q_sqrt` zur Berechnung von $\sqrt{x}$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x < 0$ (Test des Definitionsbereiches) $\Longrightarrow$ Fehlermeldung: Invalid Argument

II. Berechnung
res := `sqrt`(x)

Nun gilt `q_sqrt` := res $\approx \sqrt{x}$.

4.2.3 Fehlerabschätzung

Es wird genau eine Grundoperation mit entsprechendem Rundungsfehler ausgeführt.

Gesamtfehlerschranke:
relative Fehlerschranke für `q_sqrt`$(x) \in S_N$: $\varepsilon(\texttt{q\_sqrt}) \leq 2.2205E-016 = 2.00 \cdot \varepsilon^*$
absolute Fehlerschranke für `q_sqrt`$(x) \in S_D$: $\triangle(\texttt{q\_sqrt}) \leq 4.9497E-324$

4.2.4 Algorithmus `j_sqrt` (Intervallfunktion)

Der Gesamtalgorithmus `j_sqrt` zur Berechnung von $\sqrt{X}$ für Intervallargumente $X := [x.\mathrm{INF}, x.\mathrm{SUP}]$ stellt sich wie folgt dar:

I. X ist Punktintervall, d.h. $x.\mathrm{INF} = x.\mathrm{SUP}$

a) Falls $x.\mathrm{INF} = 0$:
res.INF:= 0
res.SUP:= 0

b) Falls $x.\mathrm{INF} \neq 0$:
$y :=$ `q_sqrt`$(x.\mathrm{INF})$
res.INF:= pred(y)
res.SUP:= succ(y)

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:
Falls $x.\mathrm{INF} = 0$ dann: res.INF:= 0
Sonst: res.INF:= pred(`q_sqrt`$(x.\mathrm{INF})$)

b) Berechnung der Oberschranke res.SUP:
res.SUP:= succ(`q_sqrt`$(x.\mathrm{SUP})$)

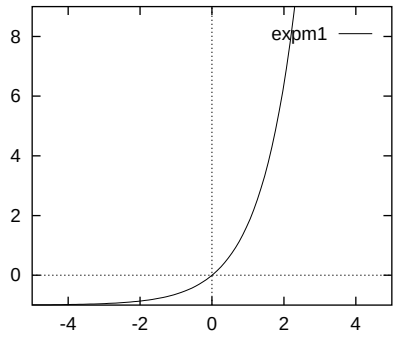
Nun gilt `j_sqrt` := res $\supseteq \sqrt{X}$ mit res := [res.INF, res.SUP].

Kapitel 5

Exponential- und Logarithmusfunktionen

5.1 Exponentialfunktion-minus-Eins: $e^x - 1$

Exponentialfunktion-minus-Eins	
Math. Schreibweise	$e^x - 1$
Definitionsbereich	$\mathbb{R}$
Potenzreihe	$\sum_{k=1}^{\infty} \frac{x^k}{k!} = \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$
1. Ableitung	e^x
Nullstellen	$x_0 = 0$
Symmetrie	-
Monotonie	monoton steigend



Es wird im folgenden ein sehr schnelles und gleichzeitig hochgenaues Tabellenverfahren zur Berechnung der Funktion $e^x - 1$ für $x \in S(2, 53, -1022, 1023)$ vorgestellt (siehe auch [39]).

5.1.1 Idee

Die gesamte Funktionsberechnung erfolgt im Zieldatenformat (IEEE double, 64 Bit), d.h. für Zwischenrechnungen wird kein höhergenaues Datenformat benutzt. Die Berechnung erfolgt wie üblich in drei Schritten:

I) Argumentreduktion

II) Polynomapproximation im reduzierten Bereich

III) Ergebnisanpassung, dabei Verwendung von tabellierten Konstanten

Im Intervall $[\ln(1 - \frac{1}{4}), \ln(1 + \frac{1}{4})]$ erfolgt die Berechnung mit Hilfe einer zweiten Polynomapproximation. In diesem Fall ist eine abschließende Ergebnisanpassung nicht notwendig. Dies sichert auch im Bereich um die Null (hier ist $e^x - 1 \approx 0$) die gewünschte hohe relative Ergebnisgenauigkeit.

5.1.2 Verfahren

Ausnahmefälle und Bereichsaufspaltung

Zunächst werden anhand des Eingabearguments x mehrere Sonderfälle abgeprüft. Für x = NAN (not a number) wird eine Fehlerbehandlungsroutine aufgerufen. In dieser Routine kann festgelegt werden, ob das Programm grundsätzlich abgebrochen und eine Fehlermeldung ausgegeben werden soll, oder ob eine Unterscheidung nach „signaling NAN“ bzw. „quiet NAN“ mit unterschiedlicher Fehlerbehandlung erfolgen soll. Für $x >$ `qExpT2c` $= 709.0895657128240$ erfolgt ein Programmabbruch, eine Fehlermeldung[1] wird ausgegeben. Der Fall $x = +\infty$ wird hier miterfaßt und muß nicht getrennt abgeprüft werden. Für $x < -37.42994775023704$ braucht nicht mehr gerechnet zu werden, als Ergebnis wird der Wert -1 zurückgegeben. Der Fall $x = -\infty$ wird hier ebenfalls miterfaßt. Für $x = 0$ sowie für kleine Eingabeargumente $0 \neq |x| < 2^{-54}$ wird x selbst zurückgegeben.

Liegt keiner der oben genannten Fälle vor werden ausgehend vom Argument x die folgenden beiden Fälle unterschieden:

$$\begin{aligned} &\text{Bereich I:} \quad x \leq \ln(1-\tfrac{1}{4}) \quad \text{oder} \quad \ln(1+\tfrac{1}{4}) \leq x \\ &\text{Bereich II:} \quad \ln(1-\tfrac{1}{4}) < x < \ln(1+\tfrac{1}{4}) \end{aligned}$$

Nur im Bereich I wird ein Tabellenverfahren verwendet.

Bereich I $\quad (x \leq \ln(1-\frac{1}{4}) \text{ oder } \ln(1+\frac{1}{4}) \leq x)$

Ausgehend vom Argument x wird ein reduziertes Argument r mit $r \in [-\ln(2)/64, \ln(2)/64]$ berechnet. Dazu wird $m \in \mathbb{Z}$ und $j \in \{0, 1, \ldots, 31\}$ so gewählt, daß in der Darstellung

$$x = (32m + j)\ln(2)/32 + r \tag{5.1}$$

der Rest $|r| \leq \ln(2)/64$ ist.

Um r zu berechnen wird also vom Argument x ein geeignetes ganzzahliges Vielfaches des Wertes $\ln(2)/32$ subtrahiert. Da es hierbei zu Auslöschung kommen kann, wird die Näherung des Wertes $\ln(2)/32$ im IEEE-Datenformat als Summe (die Addition wird nicht explizit ausgeführt) von 2 Maschinenzahlen $l_{\text{lead}}, l_{\text{trail}} \in S(2, 53)$ mit $l_{\text{lead}} + l_{\text{trail}} \approx \ln(2)/32$ höhergenau dargestellt. l_{lead} wird so gewählt, daß die letzten 20 binären Mantissenziffern den Wert 0 haben. Dadurch kann l_{lead} mit einer ganzen Zahl $n, |n| < 2^{20}$ rundungsfehlerfrei multipliziert werden. Das reduzierte Argument r selbst wird mit einigen zusätzlichen Mantissenziffern berechnet, die Darstellung erfolgt ebenfalls als Summe zweier Maschinenzahlen $r_{\text{lead}}, r_{\text{trail}} \in S(2, 53)$.

Mit Hilfe einer Polynomapproximation der Form

$$p(r) = r + a_0 r^2 + a_1 r^3 + \ldots + a_4 r^6 = r + r^2(a_0 + \ldots + a_4 r^4)$$

wird eine Näherung für $(e^r - 1)$ berechnet.

[1] Für $x > 709.78271289338399$ liegt das Ergebnis $e^x - 1$ im Überlaufbereich. Für $709.0895657128240 < x \leq 709.78271289338399$ ist für eine Zwischenrechnung eine zusätzliche Fallunterscheidung nötig, aus Gründen der Laufzeit wird darauf verzichtet.

Die Koeffizienten $a_i, i = 0, 1, \ldots, 4$ wurden mit Hilfe eines Remez-Algorithmus (Computeralgebrapaket, z.B. Maple) gefunden (siehe Abschnitt 3.5). Die Berechnung des Polynoms $r^2(a_0 + a_1 r + \ldots + a_4 r^4)$ erfolgt in der Genauigkeit des IEEE-Datenformats, zur abschließenden Addition von r bei der Berechnung von $p(r)$ wird jedoch die höhergenaue Darstellung $r_{\text{lead}} + r_{\text{trail}}$ benutzt:

$$\begin{aligned} r &:= r_{\text{lead}} \boxplus r_{\text{trail}} \\ q &:= r \boxdot r \boxdot (a_0 \boxplus r \boxdot (a_2 \boxplus \ldots)\ldots) \\ p &:= r_{\text{lead}} \boxplus (r_{\text{trail}} \boxplus q) \qquad \text{(Klammerung beachten!)} \end{aligned}$$

Die Ergebnisanpassung erfolgt durch Anwendung der Gleichung

$$e^x - 1 = 2^m(2^{j/32} + 2^{j/32}(e^r - 1)) - 1, \tag{5.2}$$

wobei m und j die gemäß (5.1) bestimmten und bekannten Größen sind. Die benötigte Potenz $2^{j/32}, j \in \{0, 1, 2, \ldots, 31\}$ wird im voraus berechnet. Näherungen für $2^{j/32}$, $j = 0(1)31$ werden vorab in jeweils 2 Konstanten lead(j), trail(j) auf ungefähr 100 Mantissenbits in einer Tabelle abgespeichert. Die Summe lead(j)+trail(j) dieser beiden Konstanten ergibt also eine Näherung für $2^{j/32}$ mit zusätzlichen Ziffern. Um den Fehler bei der Berechnung der rechten Seite von (5.2) auf dem Rechner möglichst klein zu halten, werden in Abhängigkeit von m die folgenden drei Ausdrücke verwendet:

$$\begin{aligned} m \geq 53: &\quad 2^m[\text{lead}(j) \boxplus (s \boxdot p \boxplus (\text{trail}(j) \boxminus 2^{-m}))] \\ m \leq -8: &\quad 2^m[\text{lead}(j) \boxplus (s \boxdot p \boxplus \text{trail}(j))] \boxminus 1 \\ \text{Sonst:} &\quad 2^m[(\text{lead}(j) \boxminus 2^{-m}) \boxplus (\text{lead}(j) \boxdot p \boxplus \text{trail}(j) \boxdot (1 \boxplus p))] \\ \text{jeweils mit} &\quad s := \text{lead}(j) \boxplus \text{trail}(j) \approx 2^{j/32} \end{aligned}$$

Bereich II $\quad (\ln(1 - \frac{1}{4}) < x < \ln(1 + \frac{1}{4}))$

Ausgehend von der Reihenentwicklung

$$e^x - 1 = x + \frac{x^2}{2} + \sum_{j=3}^{\infty} \frac{x^j}{j} =: x + \frac{x^2}{2} + x \cdot x^2 \cdot g(x)$$

wird eine polynomiale Bestapproximation $p(x)$ für $g(x)$ verwendet. Mit dieser in Abschnitt 3.5 angegebenen Bestapproximation wird zunächst $q := x \boxdot (x \boxdot x) \boxdot p(x)$ berechnet. Dies kann in normaler IEEE-Genauigkeit geschehen. Zur Addition des Terms $x + \frac{x^2}{2}$ muß eine höhere Genauigkeit simuliert werden. Dazu wird die Mantisse von x in zwei Teile u, v aufgespalten, $u := x|_{24}$ enthält die führenden 24 Bits der Mantisse, v ergibt sich durch $v := x - u$. Sowohl u als auch v sind auf dem Rechner exakt darstellbar.

Damit wird $y := u \cdot u \cdot \frac{1}{2}$ und $z := v \boxdot (x \boxplus u) \cdot \frac{1}{2}$ berechnet. Da von u nur die ersten 24 Bit ungleich Null sind, ist $u \cdot u$ mit maximal 48 Bit ebenfalls exakt darstellbar, woraus sich schließlich y durch eine einfache und ebenfalls rundungsfehlerfreie Exponentenanpassung ergibt. Die Summe $y + z$ ist dann eine höhergenaue Näherung für $\frac{x^2}{2}$. Dies wird durch die folgende Handrechnung (vgl. [28]) gezeigt:

Es sei o.B.d.A.: $x > 0$. Mit $x = 0.x_1x_2 \ldots x_{53} \cdot b^{\mathrm{ex}}$, $u = x|_{24} = 0.x_1x_2 \ldots x_{24} \cdot b^{\mathrm{ex}}$ und

$$x - x|_{24} = 0.\underbrace{00\ldots0}_{24\text{ Stück}}x_{25}x_{26}\ldots x_{53} \cdot b^{\mathrm{ex}} = v < 2^{-24}x$$

erhält man:

$$\begin{aligned}
y + z &= \frac{1}{2}u^2 + \frac{1}{2}v \boxdot (x \boxplus u) \\
&= \frac{1}{2}((x|_{24})^2 + (x - x|_{24}) \boxdot (x \boxplus x|_{24})) \\
&= \frac{1}{2}((x|_{24})^2 + (x - x|_{24})(x \boxplus x|_{24})(1+\varepsilon)) \\
&= \frac{1}{2}((x|_{24})^2 + (x - x|_{24})(x + x|_{24})(1+\varepsilon)^2) \\
&= \frac{1}{2}((x|_{24})^2 + (x^2 - (x|_{24})^2)(1+2\varepsilon)) \\
&= \frac{1}{2}(x^2 + 2\varepsilon((x|_{24} - v)^2 - (x|_{24})^2)) \\
&= \frac{1}{2}(x^2 + 2\varepsilon(2 \cdot v \cdot x|_{24} + v^2)) \\
&= \frac{1}{2}(x^2 + 2\varepsilon 2^{-22}x) \\
&= \frac{x^2}{2} + \varepsilon(2^{-22}x) \\
\Longrightarrow \left|\frac{x^2}{2} - (y+z)\right| &\leq \varepsilon \cdot 2^{-22}x
\end{aligned}$$

Wegen $x \in [-0.29, 0.244]$ und somit $|x| \in [0, 0.29]$ ergibt sich die folgende Abschätzung:

$$|\frac{x^2}{2} - (y+z)| \leq \varepsilon \cdot 2^{-22}|x| < \varepsilon \cdot 2^{-23}$$

Die Summe $y + z$ ist damit eine Näherung für $\frac{x^2}{2}$ mit 22 zusätzlichen Mantissenstellen. Im Fehlerkalkül wird diese Handrechnung automatisch mit erfaßt (siehe vollständiges Programmlisting `ffexpm.p`).

Zum Schluß müssen die Werte x, y, z und q in Abhängigkeit ihrer Größe geeignet aufaddiert werden. Dazu werden die folgenden beiden Fälle unterschieden:

$$e^x - 1 \approx \begin{cases} (u \boxplus y) \boxplus (q \boxplus (v \boxplus z)), & y \geq 2^{-7}, \\ x \boxplus (y \boxplus (q \boxplus z)), & y < 2^{-7}\ . \end{cases}$$

5.1.3 Algorithmus `q_expm` (Punktfunktion)

Der Gesamtalgorithmus `expm1` zur Berechnung von $e^x - 1$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument

- $x >$`qExpT2a`$= 709.7827\ldots \Longrightarrow$ Fehlermeldung: Overflow
  Der Fall $x = +\infty$ wird mit abgedeckt!
- $x = 0 \Longrightarrow \text{res} := x = \pm 0$, d.h. der exakte Funktionswert $e^0 - 1 = 0$ wird zurückgegeben[2].
- $|x| <$`qExpT1`$= 2^{-54} = 5.5511\ldots \cdot 10^{-17} \Longrightarrow \text{res}(x) := x$
- $x <$`qExpT3`$= -37.4299\ldots \Longrightarrow \text{res}(x) := -1$
  $x = -\infty$ muß damit nicht gesondert behandelt werden.

II. Fallunterscheidung

1.) Bereich I: $x \leq \ln(1 - \frac{1}{4})$ oder $\ln(1 + \frac{1}{4}) \leq x$

(a) Reduktion (red. Argument r wird berechnet als $r_{\text{lead}} + r_{\text{trail}}$)

$$\begin{aligned} n &:= \text{round}(x \boxdot \texttt{qExpInvL}), \quad \text{mit } \texttt{qExpInvL} \approx \frac{32}{\ln 2} \\ n_2 &:= n \bmod 32 \\ n_1 &:= n - n_2 \end{aligned}$$

$$\begin{aligned} r_{\text{lead}} &:= (x \boxminus n \boxdot l_{\text{lead}}) \qquad \text{mit } l_{\text{lead}} + l_{\text{trail}} \approx \frac{\ln(2)}{32} \\ r_{\text{trail}} &:= -n \boxdot l_{\text{trail}} \quad \{\Longrightarrow r_{\text{lead}} + r_{\text{trail}} = x - n \cdot (l_{\text{lead}} + l_{\text{trail}})\} \end{aligned}$$

$$\begin{aligned} m &:= n_1/32 \\ j &:= n_2 \end{aligned}$$

(b) Approximation $p(r) = r + a_0 r^2 + a_1 r^3 + \ldots + a_4 r^6$,
$r \in [-0.01083\ldots, 0.01083\ldots]$

$$\begin{aligned} r &:= r_{\text{lead}} \boxplus r_{\text{trail}} \\ q &:= r \boxdot r \boxdot (a_0 \boxplus r \boxdot (a_1 \boxplus r \boxdot (a_2 \boxplus r \boxdot (a_3 \boxplus r \boxdot a_4)))) \\ p &:= r_{\text{lead}} \boxplus (r_{\text{trail}} \boxplus q) \end{aligned}$$

(c) Ergebnisanpassung

Falls $m \geq 53$:

$$\text{res} := 2^m[\text{lead}(j) \boxplus (s \boxdot p \boxplus (\text{trail}(j) \boxminus 2^{-m}))]$$

Falls $m \leq -8$:

$$\text{res} := 2^m[\text{lead}(j) \boxplus (s \boxdot p \boxplus \text{trail}(j))] \boxminus 1$$

Falls $-7 \leq m \leq 52$:

$$\text{res} := 2^m[(\text{lead}(j) \boxminus 2^{-m}) \boxplus (\text{lead}(j) \boxdot p \boxplus \text{trail}(j) \boxdot (1 \boxplus p))]$$

mit $s := \text{lead}(j) \boxplus \text{trail}(j) \approx 2^{j/32}$ { auf 100 Bit }

2.) Bereich II: $\ln(1 - \frac{1}{4}) < x < \ln(1 + \frac{1}{4})$

(a) Genaue Berechnung von $x^2/2$

$$u := \text{CUT24}(x), \qquad \{ \text{ die ersten 24 Bits von } x \ \}$$

[2] Im IEEE-Standard wird ± 0 unterschieden.

$$
\begin{aligned}
v &:= x - u \\
y &:= u \cdot u \cdot 0.5 \\
z &:= v \boxdot (x \boxplus u) \cdot 0.5 \\
&\quad \{\Longrightarrow y + z \approx \frac{x^2}{2} \text{ mit zusätzlichen Bits } \}
\end{aligned}
$$

(b) Approximation (Horner-Schema)
$q := x^3 p(x) = x^3(b_0 + b_1 x^1 + \ldots + b_8 x^8), x \in [-0.28768\ldots, 0.22314\ldots]$

$$
q := x \boxdot x \boxdot x \boxdot (b_0 \boxplus x \boxdot (b_1 \boxplus x \boxdot (\ldots \boxplus x \boxdot b_8)\ldots))
$$

(c) Genaue Addition von $x + y + z + q$ durch Fallunterscheidung:
Falls $y \geq 2^{-7}$:
$\text{res} := (u \boxplus y) \boxplus (q \boxplus (v \boxplus z))$
Falls $y < 2^{-7}$:
$\text{res} := x \boxplus (y \boxplus (q \boxplus z))$

Nun gilt `q_expm` := res $\approx (e^x - 1)$.

Beispielhaft sei für diese Funktion die C–Quelle angegeben (der Quellcode der weiteren Funktionen findet sich in der frei erhältlichen Bibliothek `fi_lib`):

```
/*****************************************************************/
/*                                                               */
/*      Entries         : a_real q_expm(a_real x)                */
/*                        a_real x  ;                            */
/*                                                               */
/*      Arguments       : x = real argument of exp(x)-1          */
/*                                                               */
/*      Description     : Fast real exponential function - 1     */
/*                        for double IEEE Floating-Point         */
/*                        Arithmetic                             */
/*                                                               */
/*      Date            : 1997-07-07                             */
/*                                                               */
/*      Author          : W. Hofschuster                         */
/*                                                               */
/*****************************************************************/

#ifndef ALL_IN_ONE
#ifdef AIX
#include "/u/p88c/runtime/o_defs.h"
#else
#include "o_defs.h"
#endif
#define local
#endif


#include "q_defs.h"
#include "q_fcth.h"
```

```
/* ------------------------------------------------------------------- */
/* - Computation of exp(x)-1, table lookup method                     - */
/* - Idea goes back to P.T.P. Tang                                    - */
/* ------------------------------------------------------------------- */

/* ------   Prozedure for Range 1   ---------------------------------- */

#ifdef LINT_ARGS
local double q_p1ex(double x)
#else
local double q_p1ex(x)

double x;
#endif
  {
    int j;
    long int n,m;
    double r,r1,r2,q,s;
    double res;

    /* Step 1 */
    if (x>0) n=CUTINT((x*q_exil)+0.5);
    else     n=CUTINT((x*q_exil)-0.5);      /* round (x)       */
    j=n % 32;                               /* n2=n mod 32     */
    if (j<0) j+=32;                         /* We force n2>=0  */
    m=(n-j)/32;
    r1=x-n*q_exl1;
    r2=-(n*q_exl2);

    /* Step 2 */
    r=r1+r2;
    q=(((q_exa[4]*r+q_exa[3])*r+q_exa[2])*r+q_exa[1])*r+q_exa[0];
    q=r*r*q;
    q=r1+(r2+q);

    /* Step 3 */
    s=q_exld[j]+q_extl[j];
      if (m>=53)
{
          if (m<1023) { res=1.0; POWER2(res,-m); } else res=0.0;
  res=(q_exld[j]+(s*q+(q_extl[j]-res)));
  POWER2(res,m);
}
      else
{ if (m<=-8)
    {
     res=(q_exld[j]+(s*q+q_extl[j]));
     POWER2(res,m);
     res-=1;
    }
  else
    {
      res=1.0;
```

```
      POWER2(res,-m);
      res=((q_exld[j]-res)+(q_exld[j]*q+q_extl[j]*(1+q)));
      POWER2(res,m);
    }
      }
    return(res);
  }

/* ------   Prozedure for Range 2   ---------------------------------- */
#ifdef LINT_ARGS
local double q_p2ex(double x)
#else
local double q_p2ex(x)

double x;
#endif
  {
    double  u,v,y,z,q;

    /* Step 1 */
    u=(double) (CUT24(x));
    v=x-u;
    y=u*u*0.5;
    z=v*(x+u)*0.5;

    /* Step 2 */
    q=(((((((q_exb[8]*x+q_exb[7])*x+q_exb[6])*x+q_exb[5])
     *x+q_exb[4])*x+q_exb[3])*x+q_exb[2])*x+q_exb[1])*x+q_exb[0];
    q=x*x*x*q;

    /* Step 3 */
    if (y>=7.8125e-3)                /* = 2^-7 */
      return ((u+y)+(q+(v+z)) );
    else
      return (x+(y+(q+z)) );
  }

/* ------   Main program with different cases   ----------------------- */

#ifdef LINT_ARGS
local double q_expm(double x)
#else
local double q_expm(x)

double x;
#endif
{
  double fabsx,res;

#ifdef PXSCTRAP
  E_SPUSH("q_expm");
#endif
  if NANTEST(x)                                    /* Test: x=NaN */
      res=q_abortnan(INV_ARG,&x,3);
```

```
  else {
  if (x<0) fabsx=-x; else fabsx=x;
  if (fabsx<q_ext1)
      {
res = (q_p2h * x + fabsx) * q_p2mh;
      }
    else
      {
        if (q_ex2a<x)
  res=q_abortr1(OVER_FLOW,&x,3);
        else
  { if (x<q_ext3)
      {
res=-1.0+q_p2mh;
      }
    else
      { if (x==0)
  res=x;
else
  { if ((q_ext4<x) && (x<q_ext5))
      res=q_p2ex(x);
    else
      res=q_p1ex(x);
  }
      }
  }
      }
  }
#ifdef PXSCTRAP
  E_SPOPP("q_expm");
#endif
  return(res);
}
```

5.1.4 Fehlerabschätzung

Im folgenden wird eine relative Fehlerschranke für die Berechnung von $e^x - 1$ nach dem im Abschnitt 5.1 vorgestellten Verfahren hergeleitet. Diese Fehlerschranke soll für alle zulässigen Eingabeargumente aus dem Bereich der normalisierten IEEE-Zahlen gelten.

Die Fehlerabschätzung wird größtenteils auf dem Rechner mit PASCAL-XSC unter Verwendung des Moduls `abs_ari` (siehe Abschnitt 3.1) durchgeführt. Das zugehörige Programm `ffexpm1.p` ist in der Programmsammlung enthalten. Die Fehlerabschätzung wird unter der Annahme durchgeführt, daß das exakte reelle Ergebnis einer Grundoperation $(+, -, \bullet, /)$ auf dem Rechner zu einer benachbarten Gleitkommazahl gerundet wird („faithful arithmetic“). Der maximale relative Fehler einer Gleitkommaoperation ist demnach `eps52`$= 2^{-52}$. Der IEEE-Standard schreibt sogar einen Rundungsmodus zur nächstgelegenen Gleitkommazahl vor. Um nicht durch Umschalten des Rundungsmodus Laufzeiteinbußen zu verursachen, werden keine speziellen Annahmen über den eingestellten Rundungsmodus bei Programmaufruf vorausgesetzt. Dies bedeutet, daß die für alle Rundungsmodi gültige Fehlerschranke `eps52` verwendet werden muß.

Erzwingt man bei Programmausführung die Rundung zur nächstgelegenen Gleitkommazahl, so darf man die in dieser Arbeit angegebene Gesamtfehlerschranke auf nahezu die Hälfte ($1.302 \cdot 10^{-16}$) reduzieren. Diese Aussage kann sehr einfach bewiesen werden, indem man in den Fehlerabschätzungsprogrammen überall die Größe `eps52` durch `eps53`$= 2^{-53}$ ersetzt!

Die im Algorithmus verwendeten Konstanten- und Tabellenwerte sind jeweils die zur nächstgelegenen Gleitkommazahl gerundeten exakten Werte, d.h. ihr maximaler relativer Fehler beträgt `eps53`.

Bei der automatische Fehlerabschätzung wird analog zum Algorithmus zwischen Bereich I und Bereich II unterschieden. Für beide Bereiche wird jeweils ein eigenes Unterprogramm erstellt:

Unterprogrammname	Abschätzung für Argument $x \in I$ mit
`fehler1`	$I = [-37.44077..., -0.27076] \cup [0.223143..., 709.79...]$
`fehler2`	$I = [-0.287682..., -5.55...e-17] \cup [5.55...e-17, 0.223144...]$

Tabelle 5.1: Teilbereiche mit unterschiedlichen Fehlerabschätzung

Die Funktion `fehler1` berechnet eine Oberschranke des maximalen relativen Gesamtfehlers bei der Auswertung von $e^x - 1$ für ein Argument x aus einem vorgegebenen Intervall $I^* = [a, b]$ mit $I^* \subset I$. In diesem Gesamtfehler sind alle Rundungsfehler, der Approximationsfehler und die Konstantenfehler berücksichtigt. Das Intervall I^* wird durch die Parameter `kk_lb`, `kk_ub`, `k_lb` und `k_ub` festgelegt. Diese Parameter werden beim Aufruf der Funktion `fehler1` übergeben. Es gilt:

$$\begin{aligned} I^* \quad := \quad & [((\ \texttt{kk\_lb} \cdot 32 - 0.5) + \texttt{k\_lb}\ ) \bigtriangledown \tfrac{\bigtriangledown(\ln 2)}{32}, \\ & ((\ \texttt{kk\_ub} \cdot 32 - 0.5) + \texttt{k\_ub} + 1) \bigtriangleup \tfrac{\bigtriangleup(\ln 2)}{32}] \end{aligned}$$

Da bei der Ergebnisanpassung 32 verschiedene Konstanten verwendet werden, müssen bei der Fehlerabschätzung 32 Teilfälle betrachtet werden. Dies bedeutet, daß im folgenden die Teilintervalle

$$\begin{aligned} I_{kk,k} \quad := \quad & [((kk \cdot 32 - 0.5) + k) \bigtriangledown \tfrac{\bigtriangledown(\ln 2)}{32}, \\ & ((kk \cdot 32 - 0.5) + k + 1) \bigtriangleup \tfrac{\bigtriangleup(\ln 2)}{32}] \end{aligned}$$

mit $k =$`k_lb`(1)`k_ub`, $kk =$`kk_lb`(1)`kk_ub` untersucht werden. Diese Intervalle $I_{kk,k}$ sind so gewählt, daß für alle $x \in I_{kk,k}$ im Algorithmus die gleiche Fallunterscheidung ausgewählt wird. Bei der Realisierung im Programm muß beachtet werden, das die folgende Bedingung gilt:

$$I^* \subseteq \bigcup_{kk} \bigcup_{k} I_{kk,k}$$

für $k =$`k_lb`(1)`k_ub` und $kk =$`kk_lb`(1)`kk_ub`. Aufgrund der Überschätzung durch die Intervallarithmetik müssen die Teilintervalle $I_{kk,k}$ eventuell noch weiter unterteilt werden, um zu brauchbaren, scharfen Fehleraussagen zu kommen. Die Funktion `fehler1` wird vom Hauptprogramm aus mit 5 verschiedenen Parametersätzen aufgerufen:

kk_lb	kk_ub	k_lb	k_ub	Intervall $I^* = [a^*, b^*]$	Anzahl Unterteilung von $I_{kk,k}$
-54	-2	0	31	$[-37.44077..., -0.703977...]$	1
-1	-1	0	19	$[-0.703977..., -0.27076...]$ [3]	200
0	0	10	31	$[0.223143..., 0.6823...]$ [4]	500
1	1023	0	31	$[0.6823..., 709.77...]$	1
1024	1024	0	0	$[709.77..., 709.79...]$	1

In der Funktion `fehler1` werden für jedes Teilintervall alle Berechnungen des Algorithmus mit Hilfe der Intervallarithmetik von PASCAL-XSC nachvollzogen. Der bei normaler Gleitkommarechnung entstehende Rundungsfehler wird als absoluter Fehler mit Hilfe der Funktionen des Moduls `abs_ari` erfaßt. Die Konstantenfehler werden ebenfalls als absolute Fehler erfaßt, der absolute Approximationsfehler wird an geeigneter Stelle aufaddiert. Zum Schluß wird, sofern möglich, der maximale relative Gesamtfehler von allen betrachteten Teilintervallen bestimmt und an das Hauptprogramm übergeben.

Mit Hilfe der Funktion `fehler2` wird der Fehler für Argumente x mit $x \in [\ln(1 - \frac{1}{4}), -2^{-54}] \cup [\ln(1 + \frac{1}{4}), 2^{-54}]$ bestimmt. Um sowohl die Vorzeichen als auch die Fallunterscheidung bei der Summation des Ergebnisses von $e^x - 1$ richtig zu erfassen, müssen hier 4 Teilbereiche getrennt untersucht werden. Diese Teilbereiche können Tabelle 5.2 entnommen werden.

Unterprogrammname	Teilbereich	Abschätzung für Argument $x \in I$ mit
`fehler2`	a	$I = [-0.287682..., -0.125]$
	b	$I = [-0.125, -5.55...e-17]$
	c	$I = [5.55...e-17, 0.125]$
	d	$I = [0.125, 0.223144...]$

Tabelle 5.2: Teilbereiche des Bereichs II

Zur Bestimmung von scharfen Fehlerschranken ist es auch hier wieder nötig, die zu untersuchenden Teilbereiche in kleine Teilintervalle zu unterteilen. Eine Unterteilung in 50 bzw. 20 Teilintervalle hat sich als ausreichend erwiesen.

Analog zur Funktion `fehler1` wird auch hier wieder der Algorithmus nachvollzogen, sämtliche Fehler werden mit Hilfe der Funktionen des Moduls `abs_ari` erfaßt. Die Fehlerabschätzung wird mit folgendem Programm durchgeführt (die anderen Fehlerabschätzungprogramme haben einen ähnlichen Aufbau; sie werden deshalb in diesem Dokument nicht aufgelistet):

```
{------------------------------------------------------------------------}
{- Fehlerabschaetzung fuer expm1-Funktion, Tabellenverfahren            -}
{- Version: getrennte Rechnung fuer alle 32 Tabellenwerte,              -}
{-          Rechnung mit absoluten Fehlern !!!                          -}
{------------------------------------------------------------------------}

program ffexpm1(input,output);
use eps_ari,                          { Fehlerschrankenarithmetik }
```

[3] b^* wird explizit gesetzt.
[4] a^* wird explizit gesetzt.

```
    i_ari;                          { Intervallarithmetik      }

{-globale Variablen-}
var qExpInvL,qExpL1,qExpL2:real;    { Tabellenwerte der exp-Funktion }
    qExpA:array [0..4] of real;     { Tabellenwerte der exp-Funktion }
    qExpB:array [0..8] of real;     { Tabellenwerte der exp-Funktion }
    qExpLead,                       { Tabellenwerte der exp-Funktion }
    qExpTrail:array[0..31] of real; { Tabellenwerte der exp-Funktion }

{-Variablen Hauptprogramm -}
var relfehler,maxrelfehler:real;

{-------------------------------------------------------------------------}
{-                      Initialisierungsteil                              -}
{-------------------------------------------------------------------------}
procedure init;
begin
{----- Tabellenwerte der exp-Funktion ---------------------------------}
{ qExpInvL:= 4.616624130844683E+001 }
  qExpInvL:=       6497320848556798.0 /            140737488355328.0;
{ qExpL1:= 2.166084939017310E-002 }
  qExpL1:=       6243314767495168.0 /         288230376151711744.0;
{ qExpL2:= 2.325192846878874E-012 }
  qExpL2:=       5756898648422637.0 / 2475880078570760549798248448.0;

{ qExpA[0]:= 5.000000000000000E-001 }
  qExpA[0]:=  4503599627370496.0 /           9007199254740992.0;
{ qExpA[1]:= 1.666666666658136E-001 }
  qExpA[1]:=  6004799503129925.0 /          36028797018963968.0;
{ qExpA[2]:= 4.166666666638950E-002 }
  qExpA[2]:=  6004799503120717.0 /         144115188075855872.0;
{ qExpA[3]:= 8.333362425159880E-003 }
  qExpA[3]:=  4803856372824746.0 /         576460752303423488.0;
{ qExpA[4]:= 1.388893979532449E-003 }
  qExpA[4]:=  6405142946487772.0 /        4611686018427387904.0;

{ qExpB[0]:= 1.666666666666666E-001 }
  qExpB[0]:=            6004799503160660.0 /          36028797018963968.0;
{ qExpB[1]:= 4.166666666666610E-002 }
  qExpB[1]:=            6004799503160579.0 /         144115188075855872.0;
{ qExpB[2]:= 8.333333333354122E-003 }
  qExpB[2]:=            4803839602540513.0 /         576460752303423488.0;
{ qExpB[3]:= 1.388888889017890E-003 }
  qExpB[3]:=            6405119470632951.0 /        4611686018427387904.0;
{ qExpB[4]:= 1.984126964158297E-004 }
  qExpB[4]:=            7320136463514879.0 /       36893488147419103232.0;
{ qExpB[5]:= 2.480157863209126E-005 }
  qExpB[5]:=            7320133978402734.0 /      295147905179352825856.0;
{ qExpB[6]:= 2.755792722352050E-006 }
  qExpB[6]:=            6506931592885707.0 /     2361183241434822606848.0;
{ qExpB[7]:= 2.758025508816736E-007 }
  qExpB[7]:=            5209762888694261.0 /    18889465931478580854784.0;
{ qExpB[8]:= 2.448136759253856E-008 }
  qExpB[8]:=            7399039345524175.0 /   302231454903657293676544.0;
```

```
{  qExpLead [ 0]:= 1.000000000000000E+000 }
   qExpLead [ 0]:=  4503599627370496.0 /                4503599627370496.0;
{  qExpLead [ 1]:= 1.021897148654105E+000 }
   qExpLead [ 1]:=  4602215617889600.0 /                4503599627370496.0;
{  qExpLead [ 2]:= 1.044273782427410E+000 }
   qExpLead [ 2]:=  4702991017412864.0 /                4503599627370496.0;
{  qExpLead [ 3]:= 1.067140400676820E+000 }
   qExpLead [ 3]:=  4805973110840128.0 /                4503599627370496.0;
{  qExpLead [ 4]:= 1.090507732665245E+000 }
   qExpLead [ 4]:=  4911210218475840.0 /                4503599627370496.0;
{  qExpLead [ 5]:= 1.114386742595883E+000 }
   qExpLead [ 5]:=  5018751718701440.0 /                4503599627370496.0;
{  qExpLead [ 6]:= 1.138788634756679E+000 }
   qExpLead [ 6]:=  5128648071143936.0 /                4503599627370496.0;
{  qExpLead [ 7]:= 1.163724858777570E+000 }
   qExpLead [ 7]:=  5240950840352448.0 /                4503599627370496.0;
{  qExpLead [ 8]:= 1.189207115002716E+000 }
   qExpLead [ 8]:=  5355712719992576.0 /                4503599627370496.0;
{  qExpLead [ 9]:= 1.215247359980467E+000 }
   qExpLead [ 9]:=  5472987557571008.0 /                4503599627370496.0;
{  qExpLead [10]:= 1.241857812073476E+000 }
   qExpLead [10]:=  5592830379701248.0 /                4503599627370496.0;
{  qExpLead [11]:= 1.269050957191723E+000 }
   qExpLead [11]:=  5715297417922816.0 /                4503599627370496.0;
{  qExpLead [12]:= 1.296839554651001E+000 }
   qExpLead [12]:=  5840446135085568.0 /                4503599627370496.0;
{  qExpLead [13]:= 1.325236643159741E+000 }
   qExpLead [13]:=  5968335252311936.0 /                4503599627370496.0;
{  qExpLead [14]:= 1.354255546936884E+000 }
   qExpLead [14]:=  6099024776549376.0 /                4503599627370496.0;
{  qExpLead [15]:= 1.383909881963831E+000 }
   qExpLead [15]:=  6232576028726656.0 /                4503599627370496.0;
{  qExpLead [16]:= 1.414213562373092E+000 }
   qExpLead [16]:=  6369051672525760.0 /                4503599627370496.0;
{  qExpLead [17]:= 1.445180806977035E+000 }
   qExpLead [17]:=  6508515743784768.0 /                4503599627370496.0;
{  qExpLead [18]:= 1.476826145939498E+000 }
   qExpLead [18]:=  6651033680544128.0 /                4503599627370496.0;
{  qExpLead [19]:= 1.509164427593419E+000 }
   qExpLead [19]:=  6796672353750528.0 /                4503599627370496.0;
{  qExpLead [20]:= 1.542210825407935E+000 }
   qExpLead [20]:=  6945500098633920.0 /                4503599627370496.0;
{  qExpLead [21]:= 1.575980845107878E+000 }
   qExpLead [21]:=  7097586746770880.0 /                4503599627370496.0;
{  qExpLead [22]:= 1.610490331949251E+000 }
   qExpLead [22]:=  7253003658850432.0 /                4503599627370496.0;
{  qExpLead [23]:= 1.645755478153959E+000 }
   qExpLead [23]:=  7411823758157120.0 /                4503599627370496.0;
{  qExpLead [24]:= 1.681792830507419E+000 }
   qExpLead [24]:=  7574121564787584.0 /                4503599627370496.0;
{  qExpLead [25]:= 1.718619298122476E+000 }
   qExpLead [25]:=  7739973230616128.0 /                4503599627370496.0;
```

```
{  qExpLead [26]:= 1.756252160373293E+000 }
   qExpLead [26]:=  7909456575025792.0 /                4503599627370496.0;
{  qExpLead [27]:= 1.794709075003098E+000 }
   qExpLead [27]:=  8082651121422400.0 /                4503599627370496.0;
{  qExpLead [28]:= 1.834008086409341E+000 }
   qExpLead [28]:=  8259638134547584.0 /                4503599627370496.0;
{  qExpLead [29]:= 1.874167634110293E+000 }
   qExpLead [29]:=  8440500658608960.0 /                4503599627370496.0;
{  qExpLead [30]:= 1.915206561397142E+000 }
   qExpLead [30]:=  8625323556245696.0 /                4503599627370496.0;
{  qExpLead [31]:= 1.957144124175400E+000 }
   qExpLead [31]:=  8814193548346688.0 /                4503599627370496.0;

{  qExpTrail[ 0]:= 0.000000000000000E+000 }
   qExpTrail[ 0]:=  0.0;
{  qExpTrail[ 1]:= 1.159741170639136E-014 }
   qExpTrail[ 1]:=  7350732955350452.0 /   633825300114114700748351602688.0;
{  qExpTrail[ 2]:= 3.416187970930849E-015 }
   qExpTrail[ 2]:=  8661065463685896.0 /  2535301200456458802993406410752.0;
{  qExpTrail[ 3]:= 3.695759744057116E-015 }
   qExpTrail[ 3]:=  4684932057853331.0 /  1267650600228229401496703205376.0;
{  qExpTrail[ 4]:= 1.307016386977872E-014 }
   qExpTrail[ 4]:=  8284200537303158.0 /   633825300114114700748351602688.0;
{  qExpTrail[ 5]:= 9.429976191419770E-015 }
   qExpTrail[ 5]:=  5976957489595592.0 /   633825300114114700748351602688.0;
{  qExpTrail[ 6]:= 1.252362600256201E-014 }
   qExpTrail[ 6]:=  7937791009590795.0 /   633825300114114700748351602688.0;
{  qExpTrail[ 7]:= 7.365764010895274E-015 }
   qExpTrail[ 7]:=  4668607584775442.0 /   633825300114114700748351602688.0;
{  qExpTrail[ 8]:= 4.702756855740314E-015 }
   qExpTrail[ 8]:=  5961452550906630.0 /  1267650600228229401496703205376.0;
{  qExpTrail[ 9]:= 2.365364347248530E-015 }
   qExpTrail[ 9]:=  5996911069096105.0 /  2535301200456458802993406410752.0;
{  qExpTrail[10]:= 7.596096843369434E-015 }
   qExpTrail[10]:=  4814598361444511.0 /   633825300114114700748351602688.0;
{  qExpTrail[11]:= 9.994675153757751E-015 }
   qExpTrail[11]:=  6334877978873592.0 /   633825300114114700748351602688.0;
{  qExpTrail[12]:= 8.685122094871109E-015 }
   qExpTrail[12]:=  5504850118309409.0 /   633825300114114700748351602688.0;
{  qExpTrail[13]:= 4.155018977496740E-016 }
   qExpTrail[13]:=  8427379681253482.0 / 202824096036516704239472512860 16.0;
{  qExpTrail[14]:= 9.180838285724313E-015 }
   qExpTrail[14]:=  5819047581748367.0 /   633825300114114700748351602688.0;
{  qExpTrail[15]:= 1.042517908037209E-015 }
   qExpTrail[15]:=  5286193807488183.0 /  5070602400912917605986812821504.0;
{  qExpTrail[16]:= 2.789906930890878E-015 }
   qExpTrail[16]:=  7073254391049437.0 /  2535301200456458802993406410752.0;
{  qExpTrail[17]:= 1.151608187475169E-014 }
   qExpTrail[17]:=  7299184050403205.0 /   633825300114114700748351602688.0;
{  qExpTrail[18]:= 1.519472288906291E-015 }
   qExpTrail[18]:=  7704639836248887.0 /  5070602400912917605986812821504.0;
{  qExpTrail[19]:= 4.117201960800166E-015 }
   qExpTrail[19]:=  5219173536869173.0 /  1267650600228229401496703205376.0;
{  qExpTrail[20]:= 6.074702681072822E-015 }
```

```
   qExpTrail[20]:=  7700600499869997.0 /  12676506002282294014967032O5376.0;
{  qExpTrail[21]:= 8.205513465754880E-015 }
   qExpTrail[21]:=  5200862035022496.0 /   633825300114114700748351602688.0;
{  qExpTrail[22]:= 3.577420871370299E-015 }
   qExpTrail[22]:=  4534919714861555.0 /  12676506002282294014967032O5376.0;
{  qExpTrail[23]:= 6.338036743689160E-015 }
   qExpTrail[23]:=  8034416082406136.0 /  12676506002282294014967032O5376.0;
{  qExpTrail[24]:= 1.007399732183222E-014 }
   qExpTrail[24]:=  6385154375859097.0 /   633825300114114700748351602688.0;
{  qExpTrail[25]:= 1.535798430292588E-015 }
   qExpTrail[25]:=  7787423207959887.0 /  50706024009129176059868128215O4.0;
{  qExpTrail[26]:= 6.246850344855366E-015 }
   qExpTrail[26]:=  7918823589191826.0 /  12676506002282294014967032O5376.0;
{  qExpTrail[27]:= 9.122056260354196E-015 }
   qExpTrail[27]:=  5781790046876837.0 /   633825300114114700748351602688.0;
{  qExpTrail[28]:= 1.587143306717675E-015 }
   qExpTrail[28]:=  8047772661635512.0 /  50706024009129176059868128215O4.0;
{  qExpTrail[29]:= 6.822155118545929E-015 }
   qExpTrail[29]:=  8648109030874835.0 /  12676506002282294014967032O5376.0;
{  qExpTrail[30]:= 5.666960267488855E-015 }
   qExpTrail[30]:=  7183725584551774.0 /  12676506002282294014967032O5376.0;
{  qExpTrail[31]:= 8.960767791036668E-017 }
   qExpTrail[31]:=  7269838508040587.0 / 81129638414606681695789005144064.0;
end;


{------------------------------------------------------------------------}
{-  Fehlerabschaetzung fuer Argument aus dem Bereich I                  -}
{------------------------------------------------------------------------}
function fehler1(kk_lb,kk_ub,k_lb,k_ub,jmax:integer):real;
var d1,d2,d3,d4,d5,d6,d6a:real;{ abs. Fehler der Zwischenergebnisse    }
    dmax:real;                { max. abs. Fehler der exp-Funktion     }
    e1,                       { rel. Fehler einer Funktionsauswertung }
    emax:real;                { max. rel. Fehler der exp-Funktion     }

    n,n2,n1,m,j:integer;         { Var. zur Arg.red. der exp-Funktion }
    r1,r2:interval;              { Var. zur Arg.red. der exp-Funktion }

    arg:interval;             { Var. fuer die 32 Fallunterscheidungen }
    kkonst:interval;          { Var. fuer die 32 Fallunterscheidungen }
    harg:real;                { Var. fuer die 32 Fallunterscheidungen }

    maxjj,
    maxk,maxkk,maxn,maxm:integer;{ Fall mit max. rel. Fehler speichern }
    maxd,maxe:real;             { Fall mit max. rel. Fehler speichern  }
    maxarg:interval;            { Fall mit max. rel. Fehler speichern  }

    i,                          { Laufvariable Hornerschema            }
    k,                          { Laufvariable 32 Fallunterscheidungen }
    kk:integer;                 { Laufvariable gesamter Wertebereich   }
    jj:integer;                 { fuer Teilintervalle                  }
    jmin,jdiff:real;            { fuer Teilintervalle                  }

    h,                          { Hilfsgroesse fuer Zwischenrechnungen }
```

```
    x,
    r,                          { Var. der exp-Fkt. fuer red. Argument }
    q,q1:interval;              { Var. der exp-Fkt. fuer Hornerschema  }


begin
  {--- Initialisierung der Schleifenvariablen ---}
  dmax:=0; emax:=0;
  kkonst:=ln(intval(2.0))/32.0;
  maxk:=0; maxkk:=0; maxn:=0; maxm:=0; maxjj:=0;

  {--- Durchlaufen des gesamten Wertebereichs der Funktion ---}
   for kk:=kk_lb to kk_ub do begin {-54..-2, -1..-1, 0..0, 1..1023, 1024..1024}

        harg:=kk*32-0.5;  { Hilfsargument fuer die 32 Fallunterscheidungen }

    {--- Durchlaufen der 32 Fallunterscheidungen ---}
    for k:=k_lb to k_ub do     {0..31}
    for jj:=0 to jmax-1 do begin

      { intval( (harg+k)  *<kkonst.inf , (harg+k+1)*>kkonst.sup ) }
      { wird unterteilt in jmax Intervalle                        }

      jmin :=(harg+k)  *<kkonst.inf;
      jdiff:=( (harg+k+1)*>kkonst.sup - (harg+k)*<kkonst.inf ) /jmax;

      arg:=intval( jmin + jj*jdiff , jmin + (jj+1)*jdiff );

      if (jj=jmax-1) then arg.sup:= (harg+k+1)*>kkonst.sup;
      if ((kk=0) and (k=10)) then begin
        if (arg.inf<0.223143) then arg.inf:=0.223143;
        if arg.sup<arg.inf then arg.sup:=arg.inf;
      end;
      if ((kk=-1) and (k>=19)) then begin
        if (arg.sup>-0.287682072451781) then arg.sup:=-0.287682072451781;
        if arg.inf>arg.sup then arg.inf:=arg.sup;
      end;

      { Integer-Groessen n,n1,n2,m,j werden bestimmen               }
      n:=round((mid(arg)*qExpInvL));          {  round ()           }
      n2:=n mod 32;                           {  n2=n mod 32        }
      if (n2<0) then n2:=n2+32;               {  Es muss gelten n2>=0 }
      n1:=n-n2;
      m:=n1 div 32;                           {  Ganzzahlige Div.   }
      j:=n2;


      {--- Argumentreduktion: r2:=-n*L2  ---}
      h:=qExpL2*intval(1-<Eps53,1+>Eps53);
      d1:=DeltaMul(intval(n),h,0.0,rel2abs(h,Eps53));
      r2:=-n*h;

      {--- alle anderen Berechnungen der Argumentreduktion sind exakt ---}
      r1:=arg-n*qExpL1;
```

```
{--- Red. Argument als eine IEEE-Zahl:  r:=r1+r2 ---}
d2:=DeltaAdd(r1,r2,0.0,d1);
r:=r1+r2;

{--- Hornerschema: Polynomauswertung ---}
q:=qExpA[4]*intval(1-<Eps53,1+>Eps53);
d3:=rel2abs(q,Eps53);     { abs. Fehler von q }
for i:=3 downto 0 do begin
  d3:=DeltaMul(r,q,d2,d3);
  q:=r*q;
  h:=qExpA[i]*intval(1-<Eps53,1+>Eps53);  { h Hilfsvariable }
  d3:=DeltaAdd(h,q,rel2abs(h,Eps53),d3);
  q:=h+q;
end;

{--- absoluten Approximationsfehler aufaddieren ---}
d3:=d3+1.86e-15;

{--- Multiplikation mit r*r ---}
d4:=DeltaMul(r,r,d2,d2);
d4:=DeltaMul(r*r,q,d4,d3);
q:=r*r*q;

{--- Red. Argument hoehergenau aufaddieren:  p=r1+(r2+q) ---}
d5:=DeltaAdd(r2,q,d1,d4);
q:=r2+q;
d5:=DeltaAdd(r1,q,0.0,d5);
q:=r1+q;

{--- Ergebnisanpassung mit Fallunterscheidungen ---}
if m>=53 then begin

  h:=qExpTrail[j]*intval(1-<Eps53,1+>Eps53);
  d6:=DeltaAdd(intval(qExpLead[j]),h,0.0,rel2abs(h,Eps53));
  h:=qExpLead[j]+h;
  d6:=DeltaMul(h,q,d6,d5);
  q:=h*q;

  h:=qExpTrail[j]*intval(1-<Eps53,1+>Eps53);
  d6a:=DeltaAdd(h,intval(-power(2.0,-m)),rel2abs(h,Eps53),0);
  h:=h-power(2.0,-m);
  d6:=DeltaAdd(q,h,d6,d6a);
  q:=q+h;

  h:=qExpLead[j];
  d6:=DeltaAdd(h,q,0.0,d6);
  q:=q+qExpLead[j];

  { Multiplikation mit 2^m, kein zusaetlicher Fehler }

end else if m<=-8 then begin

  h:=qExpTrail[j]*intval(1-<Eps53,1+>Eps53);
```

```
    d6:=DeltaAdd(intval(qExpLead[j]),h,0.0,rel2abs(h,Eps53));
    h:=qExpLead[j]+h;
    d6:=DeltaMul(h,q,d6,d5);
    q:=h*q;


    h:=qExpTrail[j]*intval(1-<Eps53,1+>Eps53);
    d6:=DeltaAdd(h,q,rel2abs(h,Eps53),d6);
    q:=h+q;

    h:=qExpLead[j];
    d6:=DeltaAdd(h,q,0.0,d6);
    q:=q+qExpLead[j];

    { Multiplikation mit 2^m, kein zusaetlicher Fehler }
    q:=q*power(2,m);
    d6:=d6*power(2,m);   { abs. Fehler wird mit 2^m multipliziert }

    d6:=DeltaAdd(q,intval(-1.0),d6,0.0);
    q:=q-1.0;

  end else begin

    d6:=DeltaAdd(intval(1.0),q,0.0,d5);
    q1:=1+q;

    h:=qExpTrail[j]*intval(1-<Eps53,1+>Eps53);
    d6:=DeltaMul(h,q1,rel2abs(h,Eps53),d6);
    q1:=h*q1;

    h:=qExpLead[j];
    d6a:=DeltaMul(h,q,0.0,d5);
    q:=h*q;

    d6:=DeltaAdd(q,q1,d6a,d6);
    q:=q+q1;

    d6a:=0;         { h+2^-m ist exakt }
    q1:=h-power(2,-m);

    d6:=DeltaAdd(q1,q,d6a,d6);
    q:=q1+q;

   { Multiplikation mit 2^m, kein zusaetzlicher Fehler }
  end;

{--- Berechnung des relativen Fehlers ---}
if ((q.inf<=0) and (q.sup>=0)) then
   writeln ('Relativer Fehler kann nicht berechnet werden !!!')
else begin
   if abs(q.inf)<abs(q.sup) then
     e1:=d6/>abs(q.inf)
   else
     e1:=d6/>abs(q.sup);
```

```
        if emax<e1 then begin  { Fall mit max. rel. Fehler speichern }
         emax:=e1;
         maxk:=k;
         maxkk:=kk;
         maxjj:=jj;
         maxn:=n;
         maxm:=m;
         maxe:=e1;
         maxd:=d6;
         maxarg:=arg;
       end;
     end;

     if dmax<d6 then dmax:=d6;
   end;
 end;

fehler1:=maxe;
end;


{------------------------------------------------------------------------}
{-  Fehlerabschaetzung fuer Argument aus dem Bereich II                 -}
{------------------------------------------------------------------------}
function fehler2(jmax,ubereich:integer):real;
var d1,d2,d3,d4,d5,d6,d7,d8:real;{ abs. Fehler der Zwischenergebnisse  }
    dmax:real;                 { max. abs. Fehler der exp-Funktion     }
    e1,                        { rel. Fehler einer Funktionsauswertung }
    emax:real;                 { max. rel. Fehler der exp-Funktion     }

    i,                          { Laufvariable Hornerschema            }
    j,jj,                       { Laufvariablen fuer Teilintervalle    }
                                { Anzahl der Unterteilungen            }
    jjmin,jjmax,
    maxj,maxjj:integer;         { Fall mit max. rel. Fehler speichern  }

    h,                          { Hilfsgroesse fuer Zwischenrechnungen }
    x,maxarg,maxq,
    u,v,y,z,                    { Var. der exp-Fkt. fuer red. Argument }
    q,q1:interval;              { Var. der exp-Fkt. fuer Hornerschema  }

begin

  {--- Initialisierung der Schleifenvariablen ---}
  dmax:=0; emax:=0;    {Max. Fehler speichern}
  maxj:=0; maxjj:=0;   {Fallunterscheidung mit max. Fehler festhalten}

  if ubereich=1 then begin
    { Definitionsbereich D = [ -0.287682,-2^-3 ]  -  Teilbereich IIa }
    jjmin:=0; jjmax:=0;  {Schleife soll einmal durchlaufen werden!}
  end else
  if ubereich=2 then begin
    { Definitionsbereichs D = [ -2^-3,-2^-54 ]    -  Teilbereich IIb }
    jjmin:=-53; jjmax:=-3; { Betrachte Teilintervall Djj=[-2^jj, -2^(jj-1)] }
```

```
end else
if ubereich=3 then begin
  { Definitionsbereichs D = [ 2^-54 , 2^-3 ]    -  Teilbereich IIc }
  jjmin:=-54; jjmax:=-4; { Betrachte Teilintervall Djj=[ 2^jj, 2^(jj+1)] }
end else begin  { ubereich=4 }
  { Definitionsbereichs D = [ 2^-3 , 0.223144 ] -  Teilbereich IId }
  jjmin:=0; jjmax:=0;  {Schleife soll einmal durchlaufen werden!}
end;

{--- Durchlaufen des vorgegebenen Definitionsbereichs D ---}
for jj:=jjmin to jjmax do
  for j:=0 to jmax-1 do begin  { Betrachte D unterteilt in jmax Intervalle }

    if ubereich=1 then
       x:=intval(-0.287682 +   j  *(-0.125 + 0.287682)/jmax ,
                 -0.287682 + (j+1)*(-0.125 + 0.287682)/jmax)
    else if ubereich=2 then
       x:=intval(-power(2,jj)+  j  *(-power(2,jj-1) + power(2,jj))/jmax ,
                 -power(2,jj)+(j+1)*(-power(2,jj-1) + power(2,jj))/jmax)
    else if ubereich=3 then
       x:=intval(power(2,jj)+  j  *(power(2,jj+1) - power(2,jj))/jmax ,
                 power(2,jj)+(j+1)*(power(2,jj+1) - power(2,jj))/jmax)
    else  { ubereich=4 }
       x:=intval(0.125 +   j  *(0.223144 - 0.125)/jmax ,
                 0.125 + (j+1)*(0.223144 - 0.125)/jmax);

    {--- Benoetigte Hilfsvariablen berechnen  ---}
    d1:=0;                          { d1 = abs. Fehler von u, u ist exakt }
    if (ubereich=1) or (ubereich=2) then
       u:=x*intval(1.0,1+<power(2,-23))
    else  { ubereich=3 or ubereich=4 }
       u:=x*intval(1-<power(2,-23),1.0);
                                    { u enthaelt die ersten 24 Bits von x }

    d2:=0;                          { d2 = abs. Fehler von v, v ist exakt }
    if (ubereich=1) or (ubereich=2) then
      v:=intval(-power(2,-23+expo(x.inf)),0.0)     { v = x - u }
    else  { ubereich=3 or ubereich=4 }
      v:=intval(0.0,power(2,-23+expo(x.sup)));  { v = x - u }

    d3:=0;                          { d3 = abs. Fehler von y, y ist exakt }
    y:=u*u*0.5;

    d4:=DeltaAdd(x,u,0.0,d1);
    z:=x+u;
    d4:=DeltaMul(v,z,d2,d4);          { d4 = abs. Fehler von z }
    z:=v*z*0.5;                       { z = v * (x+u) * 0.5 }

    {--- Hornerschema: Polynomauswertung ---}
    q:=qExpB[8]*intval(1-<Eps53,1+>Eps53);
    d5:=rel2abs(q,Eps53);     { abs. Fehler von q }
    for i:=7 downto 0 do begin
      d5:=DeltaMul(x,q,0.0,d5);
      q:=x*q;
```

```
        h:=qExpB[i]*intval(1-<Eps53,1+>Eps53);  { h Hilfsvariable }
        d5:=DeltaAdd(h,q,rel2abs(h,Eps53),d5);
        q:=h+q;
      end;

      {--- absoluten Approximationsfehler aufaddieren ---}
      d5:=d5+4.11e-17;

      {--- Multiplikation mit x*x*x ---}
      d6:=DeltaMul(x,x,0.0,0.0);
      d6:=DeltaMul(x*x,x,d6,0.0);
      d6:=DeltaMul(x*x*x,q,d6,d5);
      q:=x*x*x*q;

      if (ubereich=1) or (ubereich=4) then begin
        {--- Ergebnisanpassung, Fall: y >= 2^-7 ---}

        d7:=DeltaAdd(v,z,d2,d4);
        d7:=DeltaAdd(q,v+z,d6,d7);
        q:=q+(v+z);

        d8:=DeltaAdd(u+y,q,0.0,d7);  { u+y ist exakt, da u max. 48 Bits}
        q:=(u+y)+q;
      end else begin    { ubereich=2 or ubereich=3 }
        {--- Ergebnisanpassung, Fall: y < 2^-7 ---}

         d7:=DeltaAdd(q,z,d6,d4);
         q:=q+z;
         d7:=DeltaAdd(y,q,d3,d7);
         q:=y+q;
         d8:=DeltaAdd(x,q,0.0,d7);
         q:=x+q;
       end;

      {--- Berechnung des relativen Fehlers ---}
      if ((q.inf<=0) and (q.sup>=0)) then
        writeln ('Relativer Fehler kann nicht berechnet werden !!!')
      else begin
        if abs(q.inf)<abs(q.sup) then
          e1:=d8/>abs(q.inf)
        else
          e1:=d8/>abs(q.sup);
        if emax<e1 then begin  { Fall mit max. rel. Fehler speichern }
          emax:=e1;
          dmax:=d8;
          maxj:=j;
          maxarg:=x;
          maxq:=q;
        end;
      end;
    end;

  fehler2:=emax;
end;
```

```
{- Beginn des Hauptprogramms -}

begin
  init;
  maxrelfehler:=0;

  relfehler:=fehler1(-54,-2,0,31,1);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich Ia:  Max. rel. Fehler = ',relfehler:23:0:1);

  relfehler:=fehler1(-1,-1,0,19,200);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich Ib:  Max. rel. Fehler = ',relfehler:23:0:1);

  relfehler:=fehler2(50,1);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich IIa: Max. rel. Fehler = ',relfehler:23:0:1);

  relfehler:=fehler2(20,2);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich IIb: Max. rel. Fehler = ',relfehler:23:0:1);

  relfehler:=fehler2(20,3);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich IIc: Max. rel. Fehler = ',relfehler:23:0:1);

  relfehler:=fehler2(50,4);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich IId: Max. rel. Fehler = ',relfehler:23:0:1);

  relfehler:=fehler1(0,0,10,31,500);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich Ic:  Max. rel. Fehler = ',relfehler:23:0:1);

  relfehler:=fehler1(1,1023,0,31,1);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich Id:  Max. rel. Fehler = ',relfehler:23:0:1);

  relfehler:=fehler1(1024,1024,0,0,1);
  if relfehler>maxrelfehler then maxrelfehler:=relfehler;
  writeln('Bereich Ie:  Max. rel. Fehler = ',relfehler:23:0:1);

  writeln;
  writeln('Max. rel. Fehlerschranke der ffexpm1-Funktion:',maxrelfehler:23:0:1);
end.
```

Die Ausführung des Programms liefert das folgende Ergebnis:

```
Test mit 1000 Zufallszahlen (Bereich I):
Minimale Groessenordnung der Fehlerschranke :  2.507476018027458E-016
```

```
Test mit 1000 Zufallszahlen (Bereich II):
Minimale Groessenordnung der Fehlerschranke :  2.395891220728322E-016

Bereich Ia:  Max. rel. Fehler =  2.340427167524933E-016
Bereich Ib:  Max. rel. Fehler =  2.443828079287488E-016
Bereich IIa: Max. rel. Fehler =  2.489959377170005E-016
Bereich IIb: Max. rel. Fehler =  2.476426669537622E-016
Bereich IIc: Max. rel. Fehler =  2.458824792336060E-016
Bereich IId: Max. rel. Fehler =  2.359601148103171E-016
Bereich Ic:  Max. rel. Fehler =  2.592561649228401E-016
Bereich Id:  Max. rel. Fehler =  2.468390169928835E-016
Bereich Ie:  Max. rel. Fehler =  2.370185165851906E-016
```

Als Gesamtfehlerschranke aller bisher untersuchten Teilbereiche ergibt sich somit

```
Max. rel. Fehlerschranke der ffexpm1-Funktion: 2.592561649228401E-016
```

Jetzt muß noch gezeigt werden, daß der relative Fehler in den bisher noch nicht untersuchten Teilintervallen (siehe Tabelle 5.1) kleiner als diese mit dem Programm berechnete Fehlerschranke ist. Die Schranke gilt dann für alle Eingabeargumente aus dem Bereich der normalisierten Gleitkommazahlen.

Einfache Handrechnung ergibt[5]:

- $x < -37.4298...$
  Es ist $e^{-37.4298...} = 5.5519\ldots \cdot 10^{-17}$ und aufgrund der Monotonie der Exponentialfunktion gilt $e^x \leq 5.5519\ldots \cdot 10^{-17}$ für alle $x < -37.4298\ldots$
  $\Longrightarrow (e^x - 1) \in [-1, -1 + \varepsilon^*]$ für $x < -37.4298\ldots$, $\varepsilon^* := 2^{-53}$
  Als Ergebnis für die Punktfunktion wird der Wert -1 zurückgeliefert, für den relativen Fehler gilt damit:

$$|\frac{(e^x - 1) - (-1)}{e^x - 1}| = |\frac{e^x}{e^x - 1}| \leq 5.6 \cdot 10^{-17}$$

- $x \in [-2^{-54}, 0)$
  Es gilt $x < 0$:

$$|\frac{(e^x - 1) - x}{e^x - 1}| \leq \frac{\frac{x^2}{2}(\frac{1}{1-x})}{-x} \leq -\frac{x}{2} \leq 2^{-54} = 5.5511... \cdot 10^{-17}$$

- $x = 0$
  Es gilt $(e^x - 1) = 0$, das Ergebnis wird direkt gesetzt und ist damit exakt.

- $x \in (0, 2^{-54}]$
  Es gilt $x > 0$:

$$|\frac{(e^x - 1) - x}{e^x - 1}| \leq \frac{\frac{x^2}{2}(\frac{1}{1-x})}{e^x - 1} \leq x^2 \frac{1}{x} = x \leq 2^{-54} = 5.5511... \cdot 10^{-17}$$

[5] Um zu verdeutlichen, daß hier die Funktion $e^x - 1$ untersucht wird, wird dieser Ausdruck geklammert.

Damit ist gezeigt, daß

$$|\frac{(e^x-1)-\texttt{expm1}(x)}{e^x-1}| \leq 2.592561649228397 \cdot 10^{16} =: \varepsilon(\texttt{expm1})$$

für alle zulässigen normalisierten Eingabeargumente zutrifft.

Die gefundene relative Fehlerschranke gilt nicht für Argumente aus dem Bereich der denormalisierten Zahlen. Es kann aber zumindest eine Aussage über den absoluten Fehler gemacht werden. Ausgehend von der Reihenentwicklung

$$(e^x-1) = \sum_{j+1}^{\infty} \frac{x^j}{j!} = x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots$$

sind die folgenden beiden Aussagen für $x \in [-\texttt{q\_minr},\texttt{q\_minr}]$ (mit $\texttt{q\_minr} := 2.225073... \cdot 10^{-308}$, kleinste positive normalisierte Zahl) sofort einsichtig:

$$(e^x-1) \geq x \tag{5.3}$$

$$|(e^x-1)-x| \leq \frac{x^2}{2}(\frac{1}{1-x}) \leq 2.475... \cdot 10^{-616} \tag{5.4}$$

Der Abstand d zweier benachbarter denormalisierter Zahlen ist immer gleich und beträgt $d := 2^{-1022-52} = 4.9406... \cdot 10^{-324}$. Zusammen mit (5.3), (5.4) folgt, daß für die Einschließung des Funktionswertes $x \in [-\texttt{q\_minr},\texttt{q\_minr}]$ gilt[6]:

$$(e^x-1) \in [x, \mathrm{succ}(x)] \; .$$

Der absolute Fehler ist kleiner als der Abstand zweier benachbarter denormalisierter Zahlen.

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_expm}(x) \in S_N$: $\varepsilon(\texttt{q\_expm}) \leq 2.5926E-016 = 2.34 \cdot \varepsilon^*$
Absolute Fehlerschranke für $\texttt{q\_expm}(x) \in S_D$: $\triangle(\texttt{q\_expm}) \leq 4.9497E-324$

5.1.5 Algorithmus j_expm (Intervallfunktion)

Der Gesamtalgorithmus `j_expm` zur Berechnung von $e^X - 1$ für Intervallargumente $X := [x.\mathrm{INF}, x.\mathrm{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\begin{aligned} \texttt{q\_exmm} &:= (1 \bigtriangledown \varepsilon(\texttt{q\_expm})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer}) \\ \texttt{q\_exmp} &:= (1 \bigtriangleup \varepsilon(\texttt{q\_expm})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer}) \end{aligned}$$

I. X ist Punktintervall, d.h. $x.\mathrm{INF} = x.\mathrm{SUP}$

a) $x.\mathrm{INF} \leq -\texttt{q\_minr}$:
$y := \texttt{q\_expm}(x.\mathrm{INF})$
res.INF:= $y \boxdot \texttt{q\_exmp}$
res.SUP:= $y \boxdot \texttt{q\_exmm}$
Falls (res.INF< x.INF) dann: res.INF:= x.INF

[6] Mit succ(x) wird der Gleitkommanachfolger der Gleitkommazahl x bezeichnet.

b) $-\texttt{q\_minr} < x.\mathrm{INF} < 0$:
res.INF:= x.INF
res.SUP:= succ(x.INF)

c) $0 \leq x.\mathrm{INF} < \texttt{q\_minr}$:
res.INF:= x.INF
Falls (x.INF = 0) dann: res.SUP:= 0
sonst: res.SUP:= succ(x.INF)

d) $\texttt{q\_minr} \leq x.\mathrm{INF}$:
y := $\texttt{q\_expm}(x.\mathrm{INF})$
res.INF:= $y \boxdot \texttt{q\_exmm}$
res.SUP:= $y \boxdot \texttt{q\_exmp}$

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\mathrm{INF} \leq -\texttt{q\_minr}$:
res.INF:= $\texttt{q\_expm}(x.\mathrm{INF}) \boxdot \texttt{q\_exmp}$

ii. $-\texttt{q\_minr} < x.\mathrm{INF} \leq 0$:
res.INF:= x.INF

iii. $0 < x.\mathrm{INF} < \texttt{q\_minr}$:
res.INF:= x.INF

iv. $\texttt{q\_minr} \leq x.\mathrm{INF}$:
res.INF:= $\texttt{q\_expm}(x.\mathrm{INF}) \boxdot \texttt{q\_exmm}$

b) Berechnung der Oberschranke res.SUP:

i. $x.\mathrm{SUP} \leq -\texttt{q\_minr}$:
res.SUP:= $\texttt{q\_expm}(x.\mathrm{SUP}) \boxdot \texttt{q\_exmm}$

ii. $-\texttt{q\_minr} < x.\mathrm{SUP} < 0$:
res.SUP:= succ(x.SUP)

iii. $0 \leq x.\mathrm{SUP} < \texttt{q\_minr}$:
res.SUP:= succ(x.SUP)

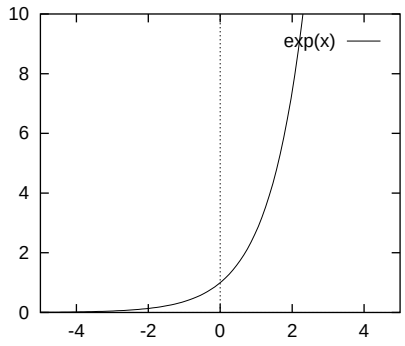
iv. $\texttt{q\_minr} \leq x.\mathrm{SUP}$:
res.SUP:= $\texttt{q\_expm}(x.\mathrm{SUP}) \boxdot \texttt{q\_exmp}$

Falls (res.INF< -1.0) dann: res.INF:= -1.0

Nun gilt $\texttt{j\_expm} := \mathrm{res} \supseteq e^X - 1$ mit res := [res.INF, res.SUP].

5.2 Exponentialfunktion zur Basis e: e^x

Exponentialfunktion	
Math. Schreibweise	e^x
Definitionsbereich	$\mathbb{R}$
Potenzreihe	$\sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$
1. Ableitung	e^x
Nullstellen	keine
Symmetrie	-
Monotonie	monoton steigend



Es wird im folgenden wiederrum ein Tabellenverfahren verwendet (siehe auch [39]).

5.2.1 Idee

Es wird die gleiche Idee verwendet, die auch der Berechnung der Funktion $e^x - 1$ zugrunde liegt. Im Bereich um die Null kann hier jedoch die Berechnung ebenfalls mit Hilfe des Tabellenverfahrens ausgeführt werden, eine eigene Fallunterscheidung ist nicht notwendig.

5.2.2 Verfahren

Ausnahmefälle und Bereichsaufspaltung

Zunächst werden anhand des Eingabearguments x mehrere Sonderfälle abgeprüft. Für x = NAN (not a number) wird eine Fehlerbehandlungsroutine aufgerufen. In dieser Routine kann festgelegt werden, ob das Programm grundsätzlich abgebrochen und eine Fehlermeldung ausgegeben werden soll, oder ob eine Unterscheidung nach „signaling NAN“ bzw. „quiet NAN“ mit unterschiedlicher Fehlerbehandlung erfolgen soll. Für $x >$ `qExp2a` $= 709.7827128933840$ erfolgt ein Programmabbruch, eine Fehlermeldung wird ausgegeben. Der Fall $x = +\infty$ wird hier miterfaßt und muß nicht getrennt abgeprüft werden. Für $x <$ `qmine` $= -708.39641853226410626$ braucht nicht mehr gerechnet zu werden, als Ergebnis wird bei der Punktfunktion der Wert 0 zurückgegeben. Der Fall $x = -\infty$ wird hier ebenfalls miterfaßt. Für $|x| < 2^{-54}$ wird als Ergebnis $x + 1$ berechnet.

In allen anderen Fällen wird wie im folgenden beschrieben vorgegangen:

Ausgehend vom Argument x wird ein reduziertes Argument r mit $r \in [-\ln(2)/64, \ln(2)/64]$ berechnet. Dazu wird $m \in \mathbb{Z}$ und $j \in \{0, 1, \ldots, 31\}$ so gewählt, daß in der Darstellung

$$x = (32m + j)\ln(2)/32 + r \tag{5.5}$$

der Rest $|r| \leq \ln(2)/64$ ist.

Um r zu berechnen wird also vom Argument x ein geeignetes ganzzahliges Vielfaches des Wertes $\ln(2)/32$ subtrahiert. Da es hierbei zu Auslöschung kommen kann, wird die Näherung des Wertes $\ln(2)/32$ im IEEE-Datenformat als Summe (die Addition wird nicht explizit ausgeführt) von 2 Maschinenzahlen $l_{\text{lead}}, l_{\text{trail}} \in S(2, 53)$ mit

$l_{\text{lead}} + l_{\text{trail}} \approx \ln(2)/32$ höhergenau dargestellt. l_{lead} wird so gewählt, daß die letzten 20 binären Mantissenziffern den Wert 0 haben. Dadurch kann l_{lead} mit einer ganzen Zahl $n, |n| < 2^{20}$ rundungsfehlerfrei multipliziert werden. Das reduzierte Argument r selbst wird mit einigen zusätzlichen Mantissenziffern berechnet, die Darstellung erfolgt ebenfalls als Summe zweier Maschinenzahlen $r_{\text{lead}}, r_{\text{trail}} \in S(2, 53)$.

Mit Hilfe einer Polynomapproximation der Form

$$p(r) = r + a_0 r^2 + a_1 r^3 + \ldots + a_4 r^6 = r + r^2(a_0 + \ldots + a_4 r^4)$$

wird eine Näherung für $(e^r - 1)$ berechnet.

Die Koeffizienten $a_i, i = 0, 1, \ldots, 4$ wurden mit Hilfe eines Remez-Algorithmus gefunden (siehe Abschnitt 3.5). Die Berechnung des Polynoms $r^2(a_0 + a_1 r + \ldots + a_4 r^4)$ erfolgt in der Genauigkeit des IEEE-Datenformats, zur abschließenden Addition von r bei der Berechnung von $p(r)$ wird jedoch die höhergenaue Darstellung $r_{\text{lead}} + r_{\text{trail}}$ benutzt:

$$\begin{aligned} r &:= r_{\text{lead}} \boxplus r_{\text{trail}} \\ q &:= r \boxdot r \boxdot (a_0 \boxplus r \boxdot (a_2 \boxplus \ldots)\ldots) \\ p &:= r_{\text{lead}} \boxplus (r_{\text{trail}} \boxplus q) \qquad \text{(Klammerung beachten!)} \end{aligned}$$

Die Ergebnisanpassung erfolgt durch Anwendung der Gleichung

$$e^x = 2^m(2^{j/32} + 2^{j/32}(e^r - 1)), \tag{5.6}$$

wobei m und j die gemäß (5.5) bestimmten und bekannten Größen sind. Die benötigte Potenz $2^{j/32}, j \in \{0, 1, 2, \ldots, 31\}$ wird im voraus berechnet. Näherungen für $2^{j/32}$, $j = 0(1)31$ werden vorab in jeweils 2 Konstanten lead(j), trail(j) auf ungefähr 100 Mantissenbits in einer Tabelle abgespeichert. Die Summe lead(j)+trail(j) dieser beiden Konstanten ergibt also eine Näherung für $2^{j/32}$ mit zusätzlichen Ziffern. Um den Fehler bei der Berechnung auf dem Rechner möglichst klein zu halten, wird der folgende Ausdruck verwendet:

$$2^m[\text{lead}(j) \boxplus (s \boxdot p \boxplus \text{trail}(j))] \text{ mit } s := \text{lead}(j) \boxplus \text{trail}(j) \approx 2^{j/32}$$

5.2.3 Algorithmus `q_exp` (Punktfunktion)

Der Gesamtalgorithmus `q_exp` zur Berechnung von e^x stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x >$`qExp2a`$= 709.7827\ldots \Longrightarrow$ Fehlermeldung: Overflow
  Der Fall $x = +\infty$ wird mit abgedeckt!
- $|x| <$`qExpT1`$= 2^{-54} = 5.5511\ldots \cdot 10^{-17} \Longrightarrow \text{res}(x) := x + 1$
- $x <$`q_mine`$= -708.3964\ldots \Longrightarrow \text{res}(x) := 0$
  $x = -\infty$ muß damit nicht gesondert behandelt werden.

II. Berechnung der Exponentialfunktion

(a) Reduktion (red. Argument r wird berechnet als $r_{\text{lead}} + r_{\text{trail}}$)

$$\begin{aligned} n &:= \text{round}(x \boxdot \texttt{qExpInvL}), \quad \text{mit } \texttt{qExpInvL} \approx \frac{32}{\ln 2} \\ n_2 &:= n \bmod 32 \\ n_1 &:= n - n_2 \end{aligned}$$

$$\begin{aligned} r_{\text{lead}} &:= (x - n \cdot l_{\text{lead}}) \qquad \text{mit } l_{\text{lead}} + l_{\text{trail}} \approx \frac{\ln(2)}{32} \quad \{ \text{ auf 86 Bit } \} \\ r_{\text{trail}} &:= -n \boxdot l_{\text{trail}} \quad \{\Longrightarrow r_{\text{lead}} + r_{\text{trail}} = x - n \cdot (l_{\text{lead}} + l_{\text{trail}})\} \end{aligned}$$

$$\begin{aligned} m &:= n_1/32 \\ j &:= n_2 \end{aligned}$$

(b) Approximation $p(r) = r + a_0 r^2 + a_1 r^3 + \ldots + a_4 r^6$,
$r \in [-0.01083\ldots, 0.01083\ldots]$

$$\begin{aligned} r &:= r_{\text{lead}} \boxplus r_{\text{trail}} \\ q &:= r \boxdot r \boxdot (a_0 \boxplus r \boxdot (a_1 \boxplus r \boxdot (a_2 \boxplus r \boxdot (a_3 \boxplus r \boxdot a_4)))) \\ p &:= r_{\text{lead}} \boxplus (r_{\text{trail}} \boxplus q) \end{aligned}$$

(c) Ergebnisanpassung

$$\text{res} := 2^m[\text{lead}(j) \boxplus (s \boxdot p \boxplus \text{trail}(j))]$$

$$\text{mit } s := \text{lead}(j) \boxplus \text{trail}(j) \approx 2^{j/32} \quad \{ \text{ auf 100 Bit } \}$$

Nun gilt `q_exp` := res $\approx e^x$.

5.2.4 Fehlerabschätzung

Im folgenden wird eine relative Fehlerschranke für die Berechnung von e^x nach dem eben vorgestellten Verfahren hergeleitet. Diese Fehlerschranke soll für alle zulässigen Eingabeargumente aus dem Bereich der normalisierten IEEE-Zahlen gelten.

Die Fehlerabschätzung wird größtenteils auf dem Rechner mit PASCAL-XSC unter Verwendung des Moduls `eps_ari` (siehe Abschnitt 3.1) durchgeführt. Es wird das Programm `ffexp.p` verwendet.

Ein wesentlicher Bestandteil des Verfahrens ist die fehlerfreie Berechnung von

$$r_{\text{lead}} := (x - n \cdot l_{\text{lead}})$$

im Gleitkommaraster des IEEE double-Formates.

Hierzu wird benötigt, daß bei der Subtraktion zweier exakter Größen das Ergebnis wieder exakt ist, wenn Auslöschung führender Ziffern auftritt (siehe Satz 5).

Im folgenden wird nun der Teil $X := [-708.39\ldots, 709.78\ldots]$ des zulässigen Definitionsbereiches der Exponentialfunktion betrachtet, für den das Tabellenverfahren angewandt wird. Mit l_{lead} :=`3F962E42 FEF00000`$= 1.011000101110\ldots \cdot 2^{-6} \approx \frac{\ln 2}{32}$ gilt

$$\begin{aligned} X &\subset [(2\cdot(-2^{15})-1)\cdot\frac{l_{\text{lead}}}{2}, (2\cdot 2^{15}+1)\cdot\frac{l_{\text{lead}}}{2}] \\ &= \bigcup_{n=-2^{15}(1)2^{15}} [(2n-1)\cdot\frac{l_{\text{lead}}}{2}, (2n+1)\cdot\frac{l_{\text{lead}}}{2}] \\ &= \bigcup_{n=-2^{15}(1)2^{15}} \underbrace{[(2n-1)\cdot\frac{l_{\text{lead}}}{2}, (n\cdot l_{\text{lead}}]}_{=: X_{n,1}} \cup \bigcup_{n=-2^{15}(1)2^{15}} \underbrace{[n\cdot l_{\text{lead}}, ((2n+1)\cdot\frac{l_{\text{lead}}}{2}]}_{=: X_{n,2}} \end{aligned}$$

Es müssen die folgenden Fälle untersucht werden:

I) $n = 0$: Es gilt

$$r_{\text{lead}} := (x - 0\cdot l_{\text{lead}}) = x,$$

d.h. es entsteht kein Rundungsfehler.

II) $n > 0$: Im folgenden sei n beliebig, aber fest gewählt. Das Argument x muß damit in einem der beiden zugehörigen Intervalle $X_{n,1}$, $X_{n,2}$ liegen, d.h. $x \in X_{n,1} \cup X_{n,2}$.

a) Fall $x \in X_{n,1}$:
Bestimme $m \in \mathbb{N}_0$ mit $2^m \le n < 2^{m+1}$ (wegen $n > 0$ gilt $m \ge 0$).
Dann gilt:

$$1.011\ldots\cdot 2^{m-6} \le n\cdot l_{\text{lead}} < 1.011\ldots\cdot 2^{m-5}$$

Daraus folgt: $\text{expo}(n\cdot l_{\text{lead}}) = m-6$.
Wegen $x \le n\cdot l_{\text{lead}}$ gilt $\text{expo}(x) \le m-6$.
Mit der Ungleichungskette

$$0 \le -r_{\text{lead}} \le n\cdot l_{\text{lead}} - x \le n\cdot l_{\text{lead}} - (2n-1)\cdot\frac{l_{\text{lead}}}{2} \le \frac{l_{\text{lead}}}{2} \le 1.011\ldots\cdot 2^{-7}$$

folgt $\text{expo}(-r_{\text{lead}}) = \text{expo}(r_{\text{lead}}) \le -7$
und daraus:

$$\text{expo}(r_{\text{lead}}) \le -7 \le -6 \le m-6 \le \max\{\text{expo}(x), \text{expo}(n\cdot l_{\text{lead}})\}$$

Nun kann Satz (5) angewandt werden und liefert das gewünschte Ergebnis.

Damit wurde gezeigt, daß r_{lead} auf der Maschine fehlerfrei berechnet werden kann.

Da bei der Ergebnisanpassung 32 verschiedene Konstanten verwendet werden, müssen bei der Fehlerabschätzung 32 Teilfälle betrachtet werden. Dies bedeutet, daß im folgenden die Teilintervalle

$$\begin{aligned} I_{kk,k} := [&((kk\cdot 32 - 0.5) + k) \bigtriangledown \tfrac{\bigtriangledown(\ln 2)}{32}, \\ &((kk\cdot 32 - 0.5) + k + 1) \bigtriangleup \tfrac{\bigtriangleup(\ln 2)}{32}] \end{aligned}$$

mit k =`k_lb`(1)`k_ub`, kk =`kk_lb`(1)`kk_ub` untersucht werden. Diese Intervalle $I_{kk,k}$ sind so

gewählt, daß für alle $x \in I_{kk,k}$ im Algorithmus die gleiche Fallunterscheidung ausgewählt wird. Bei der Realisierung im Programm muß beachtet werden, das die folgende Bedingung gilt:

$$I^* \subseteq \bigcup_{kk} \bigcup_{k} I_{kk,k}$$

für k =`k_lb`(1)`k_ub` und kk =`kk_lb`(1)`kk_ub`. Aufgrund der Überschätzung durch die Intervallarithmetik müssen die Teilintervalle $I_{kk,k}$ eventuell noch weiter unterteilt werden, um zu brauchbaren, scharfen Fehleraussagen zu kommen.

Als Gesamtfehlerschranke aller bisher untersuchten Teilbereiche ergibt sich somit

```
Max. rel. Fehlerschranke der expm1-Funktion: 2.357962555295842E-016
```

Jetzt muß noch gezeigt werden, daß der relative Fehler in den bisher noch nicht untersuchten Teilintervallen kleiner als diese mit dem Programm berechnete Fehlerschranke ist. Die Schranke gilt dann für alle Eingabeargumente aus dem Bereich der normalisierten Gleitkommazahlen.

Einfache Handrechnung ergibt:

- $x < -708.3964...$
  Es ist $e^{-708.3964...} = 2.225... \cdot 10^{-308}$ und aufgrund der Monotonie der Exponentialfunktion gilt $e^x \leq 2.225... \cdot 10^{-308}$ für alle $x < -708.3964...$
  $\Longrightarrow e^x \in [0, \texttt{nminreal}]$ für $x < -708.3964...$
  Da das Ergebnis im Unterlaufbereich liegt, wird kein relativer angegeben.

- $x \in [-2^{-54}, 0)$
  Es gilt $x < 0$:

- $x = 0$
  Es gilt $e^x = 1$, das Ergebnis ist exakt.

- $x \in (0, 2^{-54}]$
  Es gilt $x > 0$:

Damit ist gezeigt, daß

$$\left|\frac{e^x - \texttt{exp}(x)}{e^x}\right| \leq 2.357962555295842E - 016 \cdot 10^{16} =: \varepsilon(\texttt{exp})$$

für alle zulässigen Eingabeargumente zutrifft.

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_exp}(x) \in S_N$: $\varepsilon(\texttt{q\_exp}) \leq 2.3580E - 016 = 2.13 \cdot \varepsilon^*$

5.2.5 Algorithmus `j_exp` (Intervallfunktion)

Der Gesamtalgorithmus `j_exp` zur Berechnung von e^X für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\begin{aligned}
\texttt{q_mine} &:= -708.396418... \\
\texttt{q_exem} &:= (1 \bigtriangledown \varepsilon(\texttt{q_exp})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer}) \\
\texttt{q_exep} &:= (1 \bigtriangleup \varepsilon(\texttt{q_exp})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})
\end{aligned}$$

I. X ist Punktintervall, d.h. x.INF = x.SUP

a) x.INF = 0:
res.INF:= 1.0
res.SUP:= 1.0

b) x.INF $\leq -$`q_mine`:
res.INF:= 0.0
res.SUP:= `q_minr`
Falls (res.INF$<$ x.INF) dann: res.INF:= x.INF

c) Sonst:
y := `q_exp`(x.INF)
res.INF:= $y \boxdot$ `q_exem`
res.SUP:= $y \boxdot$ `q_exep`

II. X ist echtes Intervall, d.h. x.INF $\neq$ x.SUP

a) Berechnung der Unterschranke res.INF:

i. x.INF $\leq$ `q_mine`:
res.INF:= 0.0

ii. Sonst:
res.INF:= `q_exp`(x.INF) $\boxdot$ `q_exem`

b) Berechnung der Oberschranke res.SUP:

i. x.SUP $\leq$ `q_mine`:
res.SUP:= `q_minr`

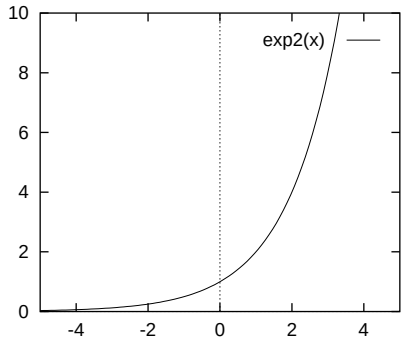
ii. Sonst:
res.SUP:= `q_exp`(x.SUP) $\boxdot$ `q_exep`

Falls (res.INF$<$ 0.0) dann: res.INF:= 0.0
Falls (x.SUP$\leq$ 0.0) und (res.SUP$>$ 1.0) dann: res.SUP:= 1.0
Falls (x.INF$\geq$ 0.0) und (res.INF$<$ 1.0) dann: res.INF:= 1.0

Nun gilt `j_exp` := res $\supseteq e^X$ mit res := [res.INF, res.SUP].

5.3 Exponentialfunktion zur Basis 2: 2^x

Exponentialfunktion zur Basis 2	
Math. Schreibweise	2^x
Definitionsbereich	$\mathbb{R}$
Potenzreihe	$\sum_{k=0}^{\infty} \frac{(x \ln 2)^k}{k!}$ $= 1 + \frac{x \ln 2}{1} + \frac{(x \ln 2)^2}{2!} + \ldots$
1. Ableitung	$2^x \ln 2$
Nullstellen	keine
Monotonie	monoton steigend



5.3.1 Idee

Das bei der Exponentialfunktion zur Basis e beschriebene Tabellenverfahren wird so modifiziert, daß es für die Exponentialfunktion zur Basis 2 verwendet werden kann. Bei der Berechnung können für beide Exponentialfunktionen die gleichen Tabellenwerte verwendet werden.

5.3.2 Algorithmus `q_exp2` (Punktfunktion)

Der Gesamtalgorithmus `q_exp2` zur Berechnung von 2^x stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x > 1023 \Longrightarrow$ Fehlermeldung: Overflow
  Der Fall $x = +\infty$ wird mit abgedeckt!
- $|x| <$`qExpT1`$= 2^{-54} = 5.5511\ldots \cdot 10^{-17} \Longrightarrow \mathrm{res}(x) := x + 1$
- $x < -1022 \Longrightarrow \mathrm{res}(x) := 0$
  $x = -\infty$ muß damit nicht gesondert behandelt werden.
- x ist ganzzahlig $\Longrightarrow$ res $= 2^x$ (Realisierung über rundungsfehlerfreie Änderung des Exponenten)

II. Berechnung der Exponentialfunktion zur Basis 2

(a) Reduktion

$$\begin{aligned} n &:= \mathrm{round}(x \boxdot 32) \\ j &:= n \bmod 32 \\ n_1 &:= n - j \end{aligned}$$

$$r := x \boxminus n \boxdot 0.03125$$

$$m := n_1/32$$

(b) Approximation $p(r) = r \cdot (c_0 + c_1 r + \ldots + c_6 r^6)$,
$r \in [\ldots, \ldots]$

$$\begin{aligned} q &:= c_0 \boxplus r \boxdot (c_1 \boxplus r \boxdot (c_2 \boxplus r \boxdot (c_3 \boxplus r \boxdot (c_4 \boxplus r \boxdot (c_5 \boxplus r \boxdot c_6))))) \\ q &:= r \boxdot q \end{aligned}$$

(c) Ergebnisanpassung

$$\mathrm{res} := 2^m[\mathrm{lead}(j) \boxplus (s \boxdot q \boxplus \mathrm{trail}(j))]$$

mit $s := \mathrm{lead}(j) \boxplus \mathrm{trail}(j) \approx 2^{j/32}$ { auf 100 Bit }

Nun gilt q_exp2 := res $\approx 2^x$.

5.3.3 Fehlerabschätzung

Gesamtfehlerschranke:
Relative Fehlerschranke für q_exp(x) $\in S_N$: $\varepsilon(\texttt{q\_exp}) \leq 2.3305E-016 = 2.10 \cdot \varepsilon^*$

5.3.4 Algorithmus j_exp2 (Intervallfunktion)

Der Gesamtalgorithmus j_exp2 zur Berechnung von 2^X für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_e2em} := (1 \bigtriangledown \varepsilon(\texttt{q\_exp2})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_e2ep} := (1 \bigtriangleup \varepsilon(\texttt{q\_exp2})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\text{INF} \leq -1022$:
res.INF:= 0.0
res.SUP:= q_minr

b) $x.$INF ganzzahlig:
y := q_exp2($x.$INF)
res.INF:= y
res.SUP:= y

c) Sonst:
y := q_exp2($x.$INF)
res.INF:= $y \boxdot$ q_e2em
res.SUP:= $y \boxdot$ q_e2ep

II. X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\text{INF} \leq -1022$:
res.INF:= 0.0

ii. $x.$INF ganzzahlig:
res.INF:= q_exp2($x.$INF)

iii. Sonst:
res.INF:= q_exp2($x.$INF) $\boxdot$ q_e2em

b) Berechnung der Oberschranke res.SUP:

i. $x.\text{SUP} \leq -1022$:
res.SUP:= q_minr

ii. x.SUP ganzzahlig:
res.SUP:= `q_exp2`(x.SUP)

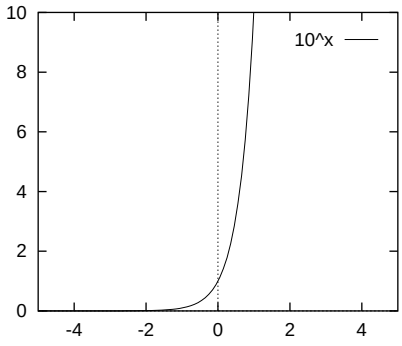
iii. Sonst:
res.SUP:= `q_exp2`(x.SUP) $\boxdot$ `q_e2ep`

Falls (res.INF<0.0) dann: res.INF:$=0.0$
Falls (x.SUP≤ 0.0) und (res.SUP>1.0) dann: res.SUP:$=1.0$
Falls (x.INF≥ 0.0) und (res.INF<1.0) dann: res.INF:$=1.0$

Nun gilt `j_exp2` := res $\supseteq 2^X$ mit res := [res.INF, res.SUP].

5.4 Exponentialfunktion zur Basis 10: 10^x

Exponentialfunktion zur Basis 10	
Math. Schreibweise	10^x
Definitionsbereich	$\mathbb{R}$
Potenzreihe	$\sum_{k=0}^{\infty} \frac{(x \ln 10)^k}{k!}$ $= 1 + \frac{x \ln 10}{1} + \frac{(x \ln 10)^2}{2!} + \ldots$
1. Ableitung	$2^x \ln 10$
Nullstellen	keine
Monotonie	monoton steigend



5.4.1 Idee

Hier wird ebenfalls das Tabellenverfahren zur Berechnung der Exponentialfunktion zur Basis e modifiziert und eingesetzt.

5.4.2 Algorithmus `q_ex10` (Punktfunktion)

Der Gesamtalgorithmus `q_ex10` zur Berechnung von 10^x stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x > 307.9536\ldots \Longrightarrow$ Fehlermeldung: Overflow
  Der Fall $x = +\infty$ wird mit abgedeckt!
- $|x| <$`qExpT1`$= 2^{-54} = 5.5511\ldots \cdot 10^{-17} \Longrightarrow \mathrm{res}(x) := x + 1$
- $x <$`q_mine`$= -708.3964\ldots \Longrightarrow \mathrm{res}(x) := 0$
  $x = -\infty$ muß damit nicht gesondert behandelt werden.

II. Berechnung der Exponentialfunktion zur Basis 10

(a) Reduktion (red. Argument r wird berechnet als $r_{\text{lead}} + r_{\text{trail}}$)

$$\begin{aligned} n &:= \text{round}(x \boxdot \texttt{q\_e10i}), \quad \text{mit } \texttt{q\_e10i+} \approx \frac{32}{\log_{10} 2} \\ j &:= n \bmod 32 \\ n_1 &:= n - j \end{aligned}$$

$$\begin{aligned} r_{\text{lead}} &:= (x \boxminus n \boxdot e1_{\text{lead}}) \qquad \text{mit } e1_{\text{lead}} + e1_{\text{trail}} \approx \frac{\log_{10}(2)}{32} \\ r_{\text{trail}} &:= -n \boxdot e1_{\text{trail}} \quad \{\Longrightarrow r_{\text{lead}} + r_{\text{trail}} = x - n \cdot (e1_{\text{lead}} + e2_{\text{trail}})\} \end{aligned}$$

$$m := n_1/32$$

(b) Approximation $p(r) = r * (d_0 + d_1 r^1 + \ldots + d_6 r^6)$,
$r \in [\ldots, \ldots]$

$$\begin{aligned} r &:= r_{\text{lead}} \boxplus r_{\text{trail}} \\ q &:= d_0 \boxplus r \boxdot (d_1 \boxplus r \boxdot (d_2 \boxplus r \boxdot (d_3 \boxplus r \boxdot (d_4 \boxplus r \boxdot (d_5 \boxplus r \boxdot d_6))))) \\ q &:= r \boxdot q \\ p &:= r_{\text{lead}} \boxplus (r_{\text{trail}} \boxplus q) \end{aligned}$$

(c) Ergebnisanpassung

$$\text{res} := 2^m[\text{lead}(j) \boxplus (s \boxdot p \boxplus \text{trail}(j))]$$

$$\text{mit } s := \text{lead}(j) \boxplus \text{trail}(j) \approx 2^{j/32} \quad \{ \text{ auf 100 Bit } \}$$

Nun gilt $\texttt{q\_ex10} := \text{res} \approx 10^x$.

5.4.3 Fehlerabschätzung

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_ex10}(x) \in S_N$: $\varepsilon(\texttt{q\_ex10}) \leq 2.4181E-016 = 2.18 \cdot \varepsilon^*$

5.4.4 Algorithmus `j_ex10` (Intervallfunktion)

Der Gesamtalgorithmus `j_ex10` zur Berechnung von 10^X für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\begin{aligned} \texttt{q\_e10m} &:= (1 \bigtriangledown \varepsilon(\texttt{q\_ex10})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer}) \\ \texttt{q\_e10p} &:= (1 \bigtriangleup \varepsilon(\texttt{q\_ex10})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer}) \end{aligned}$$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $0 \leq x.\mathrm{INF} \leq 22$ und $x.\mathrm{INF}$ ganzzahlig:
$y :=$ `q_ex10`$(x.\mathrm{INF})$
res.INF:= y
res.SUP:= y

b) $x.\mathrm{INF} \leq -307.6526\ldots$:
res.INF:= 0.0
res.SUP:= `q_minr`

c) Sonst:
$y :=$ `q_ex10`$(x.\mathrm{INF})$
res.INF:= $y \boxdot$ `q_e10m`
res.SUP:= $y \boxdot$ `q_e10p`

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\mathrm{INF} \leq -307.6526\ldots$:
res.INF:= 0.0

ii. $0 \leq x.\mathrm{INF} \leq 22$ und $x.\mathrm{INF}$ ganzzahlig:
res.INF:= `q_ex10`$(x.\mathrm{INF})$

iii. Sonst:
res.INF:= `q_ex10`$(x.\mathrm{INF}) \boxdot$ `q_e10m`

b) Berechnung der Oberschranke res.SUP:

i. $x.\mathrm{SUP} \leq -307.6526\ldots$:
res.SUP:= `q_minr`

ii. $0 \leq x.\mathrm{SUP} \leq 22$ und $x.\mathrm{SUP}$ ganzzahlig:
res.SUP:= `q_ex10`$(x.\mathrm{SUP})$

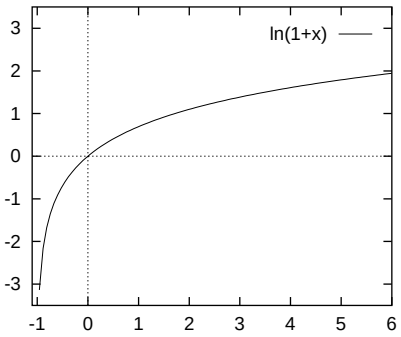
iii. Sonst:
res.SUP:= `q_ex10`$(x.\mathrm{SUP}) \boxdot$ `q_e10p`

Falls (res.INF<0.0) dann: res.INF:= 0.0
Falls (x.SUP≤ 0.0) und (res.SUP>1.0) dann: res.SUP:= 1.0
Falls (x.INF≥ 0.0) und (res.INF<1.0) dann: res.INF:= 1.0

Nun gilt `j_ex10` := res $\supseteq 10^X$ mit res := [res.INF, res.SUP].

5.5 Logarithmusfunktion: $\ln(1+x)$

Logarithmusfunktion	
Math. Schreibweise	$\ln(1+x)$
Definitionsbereich	$(-1,\infty)$
Potenzreihe für $-1 < x \leq 1$	$\sum_{k=1}^{\infty}(-1)^{k+1}\frac{x^k}{k}$ $= x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + - \ldots$
1. Ableitung	$\frac{1}{1+x}$
Nullstellen	$x_0 = 0$
Monotonie	monoton steigend



5.5.1 Idee

Es wird ein Tabellenverfahren nach der Idee von Tang [38] realisiert. Im wesentlichen werden dabei zwei Fälle unterschieden.

1. Der Fall $e^{-1/16} - 1 < x < e^{1/16} - 1$:
   Mit der Transformation $\alpha := \dfrac{2x}{2+x}$ gilt

   $$\ln(1+x) = \ln\left(\frac{1+\alpha/2}{1-\alpha/2}\right) = \alpha + \alpha \cdot \left(\frac{1}{3}\left(\frac{\alpha}{2}\right)^2 + \frac{1}{5}\left(\frac{\alpha}{2}\right)^4 + \cdots\right).$$

   Nun wird eine Bestapproximation p mit

   $$\alpha^2 p(\alpha^2) \approx \frac{1}{\alpha}\left(\ln\left(\frac{1+\alpha/2}{1-\alpha/2}\right) - \alpha\right)$$

   bestimmt. Die eigentliche Funktionsberechnung besteht dann aus einer sorgfältigen Auswertung von $\alpha + \alpha^3 p(\alpha^2)$.

2. Der Fall $-1 < x \leq e^{-1/16} - 1$ oder $e^{-1/16} - 1 \leq x$:
   Ausgehend vom Argument x wird zunächst eine ganze Zahl m mit

   $$1 \leq 2^{-m}(1+x) < 2$$

   bestimmt. Weiter werden zwei Gleitkommazahlen F und f so bestimmt, daß gilt:

   $$\begin{aligned} 1+x &\approx 2^m(F+f) \\ F &= 1 + j \cdot 2^{-7}, \quad j = 0, 1, \ldots, 2^7, \quad \text{und} \\ |f| &\leq 2^{-8} \end{aligned}$$

   Anschließend wird die Umformung

   $$\ln(2^m(F+f)) = m \cdot \ln(2) + \ln(F) + \ln(1 + \frac{f}{F})$$

verwendet. Die Werte $\ln(2)$ sowie $\ln(F)$ für alle zulässigen F (F kann 128 verschiedene Werte annehmen) werden im voraus berechnet und in einer Tabelle abgespeichert. Zur Berechnung von $\ln(1+\frac{f}{F})$ wird die Approximation

$$\ln(1+\frac{f}{F}) \approx \beta + \beta \cdot \left(\frac{1}{3}\left(\frac{\beta}{2}\right)^2 + \frac{1}{5}\left(\frac{\beta}{2}\right)^4 + \cdots \right)$$

mit $\beta := \dfrac{2f}{2F+f}$ verwendet.

Für betragsmäßig sehr kleine Argumente wird die Approximation

$$\ln(1+x) \approx x$$

angewandt.

5.5.2 Algorithmus `q_ln1p` (Punktfunktion)

Der Gesamtalgorithmus `q_ln1p` zur Berechnung von $\ln(1+x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x \leq -1 \Longrightarrow$ Fehlermeldung: Invalid Argument
- `-q_lgt5`$= -2^{-53} < x <$`q_lgt5`$= 2^{-53} \Longrightarrow \mathrm{res}(x) := x$

II. Berechnung von $\ln(x+1)$

1. Fall: `q_lgt3`$= e^{-1/16} - 1 < x < e^{1/16} - 1 =$`q_lgt4`

   i. Transformation
   $g := 1 \boxslash (2+x)$
   $u := 2 \cdot x \boxdot g$

   ii. Approximation
   $v := u \boxdot u$
   $q := u \boxdot v \boxdot (c_0 \boxplus v \boxdot (c_1 \boxplus v \boxdot (c_2 \boxplus v \boxdot c_3)))$

   iii. Ergebnisanpassung
   $u_1 := u|_{24}$
   $f_1 := x|_{24}$
   $f_2 := x - f_1$
   $u_2 := ((2 \cdot (x \boxminus u_1) \boxminus u_1 \cdot f_1) \boxminus u_1 \cdot f_2) \boxdot g$
   $\mathrm{res} := u_1 \boxplus (u_2 \boxplus q)$

2. Fall: $-1 < x \leq e^{-1/16} - 1$ oder $e^{-1/16} - 1 \leq x$

i. Berechnung des reduzierten Arguments, d.h. Bestimmung von m, F, f
Falls $x < 2^{55}$ dann $y := 1 \boxplus x$, sonst $y := x$
$m := \text{expo}(y) - 1023$
$y := 2^{-m} \cdot y$
$F := 2^{-7} \cdot \text{round}(2^7 \cdot y)$
Berechnung von f mit folgenden Fallunterscheidungen:

A. Falls $m \leq -2$ dann
$f := y - F$

B. Falls $-1 \leq m \leq 52$ dann
$f := 1.0 \cdot 2^{-m}$
$h := x \cdot 2^{-m}$
$f := (f \boxminus F) \boxplus h$

C. Falls $m > 52$ dann
$f := 1.0 \cdot 2^{-m}$
$h := x \cdot 2^{-m}$
$f := (h \boxminus F) \boxplus h$

ii. Initialisierungen
$j := \text{floor}((F-1) \cdot 128)$
$l_{\text{lead}} := m \boxdot \text{lead}(128) \boxplus \text{lead}(j)$
$l_{\text{trail}} := m \boxdot \text{trail}(128) \boxplus \text{trail}(j)$

iii. Approximation
$u := (2 \cdot f) \boxslash (y + F)$
$v := u \boxdot u$
$q := u \boxdot v \boxdot (b_0 \boxplus v \boxdot b_1)$

iv. Ergebnisanpassung
$\text{res} := l_{\text{lead}} \boxplus (u \boxplus (q \boxplus l_{\text{trail}}))$

Nun gilt `q_ln1p` $:=$ res $\approx \ln(x+1)$.

5.5.3 Fehlerabschätzung

Gesamtfehlerschranke:
Relative Fehlerschranke für `q_ln1p`$(x) \in S_N$: $\varepsilon(\texttt{q\_ln1p}) \leq 2.5082E-016 = 2.26 \cdot \varepsilon^*$

5.5.4 Algorithmus `j_lg1p` (Intervallfunktion)

Der Gesamtalgorithmus `j_lg1p` zur Berechnung von $\log(X+1)$ für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\begin{aligned} \texttt{q\_lgpm} &:= (1 \bigtriangledown \varepsilon(\texttt{q\_lg1p})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer}) \\ \texttt{q\_lgpp} &:= (1 \bigtriangleup \varepsilon(\texttt{q\_lg1p})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer}) \end{aligned}$$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\mathrm{INF} < 0$:
$y :=$ `q_lg1p`$(x.\mathrm{INF})$
res.INF:= $y \boxdot$ `q_lgpp`
res.SUP:= $y \boxdot$ `q_lgpm`

b) $x.\mathrm{INF} \geq 0$:
$y :=$ `q_lg1p`$(x.\mathrm{INF})$
res.INF:= $y \boxdot$ `q_lgpm`
res.SUP:= $y \boxdot$ `q_lgpp`

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\mathrm{INF} < 0$:
res.INF:= `q_lg1p`$(x.\mathrm{INF}) \boxdot$ `q_lgpp`

ii. $x.\mathrm{INF} \geq 0$:
res.INF:= `q_lg1p`$(x.\mathrm{INF}) \boxdot$ `q_lgpm`

b) Berechnung der Oberschranke res.SUP:

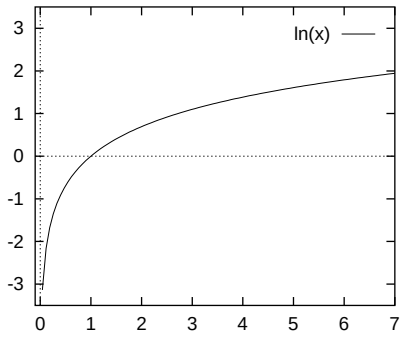
i. $x.\mathrm{SUP} < 0$:
res.SUP:= `q_lg1p`$(x.\mathrm{SUP}) \boxdot$ `q_lgpm`

ii. $x.\mathrm{SUP} \geq 0$:
res.SUP:= `q_lg1p`$(x.\mathrm{SUP}) \boxdot$ `q_lgpp`

Nun gilt `j_lg1p` := res $\supseteq \ln(X+1)$ mit res := [res.INF, res.SUP].

5.6 Logarithmusfunktion: $\ln x$

Logarithmusfunktion zur Basis e	
Math. Schreibweise	$\ln x$
Definitionsbereich	$(0, \infty)$
Potenzreihe für $0 < x \leq 2$	$\sum_{k=1}^{\infty} (-1)^{k+1} \frac{(x-1)^k}{k}$ $= \frac{(x-1)}{1} - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} \mp \ldots$
1. Ableitung	$\frac{1}{x}$
Nullstellen	$x_0 = 1$
Monotonie	monoton steigend



5.6.1 Idee

Es wird ein Tabellenverfahren nach der Idee von Tang [38] realisiert. Im wesentlichen werden wiederrum zwei Fälle unterschieden.

1. Der Fall $e^{-1/16} < x < e^{1/16}$:
   Mit $y := x - 1$ wird die Transformation $\alpha := \dfrac{2y}{2+y}$ angewandt:

$$\ln(x) = \ln(1+y) = \ln\left(\frac{1+\alpha/2}{1-\alpha/2}\right) = \alpha + \alpha \cdot \left(\frac{1}{3}\left(\frac{\alpha}{2}\right)^2 + \frac{1}{5}\left(\frac{\alpha}{2}\right)^4 + \cdots\right).$$

   Nun wird eine Bestapproximation p mit

$$\alpha^2 p(\alpha^2) \approx \frac{1}{\alpha}\left(\ln\left(\frac{1+\alpha/2}{1-\alpha/2}\right) - \alpha\right)$$

   bestimmt. Die eigentliche Funktionsberechnung besteht dann aus einer sorgfältigen Auswertung von $\alpha + \alpha^3 p(\alpha^2)$.

2. Der Fall $-1 < x \leq e^{-1/16}$ oder $e^{-1/16} \leq x$:
   Ausgehend vom Argument x wird zunächst eine ganze Zahl m mit

$$1 \leq 2^{-m} x < 2$$

   bestimmt. Weiter werden zwei Gleitkommazahlen F und f so bestimmt, daß gilt:

$$\begin{aligned} x &= 2^m(F+f) \\ F &= 1 + j \cdot 2^{-7}, \quad j = 0, 1, \ldots, 2^7, \quad \text{und} \\ |f| &\leq 2^{-8} \end{aligned}$$

   Anschließend wird die Umformung

$$\ln(2^m(F+f)) = m \cdot \ln(2) + \ln(F) + \ln(1 + \frac{f}{F})$$

   verwendet. Die Werte $\ln(2)$ sowie $\ln(F)$ für alle zulässigen F (F kann 128 verschiedene Werte annehmen) werden im voraus berechnet und in einer Tabelle abgespeichert. Zur Berechnung von $\ln(1 + \frac{f}{F})$ wird die Approximation

$$\ln(1 + \frac{f}{F}) \approx \beta + \beta \cdot \left(\frac{1}{3}\left(\frac{\beta}{2}\right)^2 + \frac{1}{5}\left(\frac{\beta}{2}\right)^4 + \cdots\right)$$

   mit $\beta := \dfrac{2f}{2F+f}$ verwendet.

5.6.2 Algorithmus q_log (Punktfunktion)

Der Gesamtalgorithmus q_log zur Berechnung von $\ln(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x \leq$ q_minr $\Longrightarrow$ Fehlermeldung: Invalid Argument

II. Berechnung von $\ln(x)$

1. Fall: q_lgt1$< x <$q_lgt2

   i. Transformation
   $y := x \boxminus 1$
   $g := 1 \not\boxplus (2 + y)$
   $u := 2 \cdot y \boxdot g$

   ii. Approximation
   $v := u \boxdot u$
   $q := u \boxdot v \boxdot (c_0 \boxplus v \boxdot (c_1 \boxplus v \boxdot (c_2 \boxplus v \boxdot c_3)))$

   iii. Ergebnisanpassung
   $u_1 := u|_{24}$
   $f_1 := x|_{24}$
   $f_2 := x - f_1$
   $u_2 := ((2 \cdot (x \boxminus u_1) \boxminus u_1 \cdot f_1) \boxminus u_1 \cdot f_2) \boxdot g$
   res $:= u_1 \boxplus (u_2 \boxplus q)$

2. Fall: $0 < x \leq$q_lgt1 oder q_lgt2$\leq x$

   i. Berechnung des reduzierten Arguments, d.h. Bestimmung von m, F, f
   $m := \text{expo}(x) - 1023$
   $y := 2^{-m} \cdot x$
   $F := 2^{-7} \cdot \text{round}(2^7 \cdot y)$
   $f := y - F$

   ii. Initialisierungen
   $j := \text{floor}((F - 1) \cdot 128)$
   $l_{\text{lead}} := m \boxdot \text{lead}(128) \boxplus \text{lead}(j)$
   $l_{\text{trail}} := m \boxdot \text{trail}(128) \boxplus \text{trail}(j)$

   iii. Approximation
   $u := (2 \cdot f) \not\boxplus (y + F)$
   $v := u \boxdot u$
   $q := u \boxdot v \boxdot (b_0 \boxplus v \boxdot b_1)$

   iv. Ergebnisanpassung
   res $:= l_{\text{lead}} \boxplus (u \boxplus (q \boxplus l_{\text{trail}}))$

Nun gilt q_log := res $\approx \ln(x)$.

5.6.3 Fehlerabschätzung

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_log}(x) \in S_N$: $\varepsilon(\texttt{q\_log}) \leq 2.9398E-016 = 2.65 \cdot \varepsilon^*$

5.6.4 Algorithmus `j_log` (Intervallfunktion)

Der Gesamtalgorithmus `j_log` zur Berechnung von $\log(X)$ für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\begin{aligned}\texttt{q\_logm} &:= (1 \bigtriangledown \varepsilon(\texttt{q\_log})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer}) \\ \texttt{q\_logp} &:= (1 \bigtriangleup \varepsilon(\texttt{q\_log})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})\end{aligned}$$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\text{INF} < 0$:
$y := \texttt{q\_log}(x.\text{INF})$
res.INF:= $y \boxdot \texttt{q\_logp}$
res.SUP:= $y \boxdot \texttt{q\_logm}$

b) $x.\text{INF} \geq 0$:
$y := \texttt{q\_log}(x.\text{INF})$
res.INF:= $y \boxdot \texttt{q\_logm}$
res.SUP:= $y \boxdot \texttt{q\_logp}$

II. X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\text{INF} < 0$:
res.INF:= $\texttt{q\_log}(x.\text{INF}) \boxdot \texttt{q\_logp}$

ii. $x.\text{INF} \geq 0$:
res.INF:= $\texttt{q\_log}(x.\text{INF}) \boxdot \texttt{q\_logm}$

b) Berechnung der Oberschranke res.SUP:

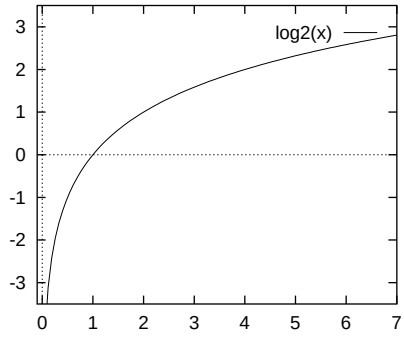
i. $x.\text{SUP} < 0$:
res.SUP:= $\texttt{q\_log}(x.\text{SUP}) \boxdot \texttt{q\_logm}$

ii. $x.\text{SUP} \geq 0$:
res.SUP:= $\texttt{q\_log}(x.\text{SUP}) \boxdot \texttt{q\_logp}$

Nun gilt `j_log` := res $\supseteq \ln(X)$ mit res := [res.INF, res.SUP].

5.7 Logarithmusfunktion zur Basis 2: $\log_2 x$

Logarithmusfunktion zur Basis 2	
Math. Schreibweise	$\log_2 x$
Definitionsbereich	$(0, \infty)$
1. Ableitung	$\frac{1}{x \ln 2}$
Nullstellen	$x_0 = 1$
Monotonie	monoton steigend



5.7.1 Idee

Die Funktion $\log_2 x$ wird mit Hilfe der Formel

$$\log_2 x = \ln x \cdot \frac{1}{ln2}$$

auf die Berechnung des natürlichen Logarithmus zurückgeführt. Der Wert $\frac{1}{ln2}$ wird im voraus berechnet und gerundet zur nächstgelegenen Gleitkommazahl als Konstante unter dem Namen `q_l2i` abgespeichert.

5.7.2 Algorithmus `q_log2` (Punktfunktion)

Der Gesamtalgorithmus `q_log2` zur Berechnung von $\log_2 x$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument

II. Berechnung mit Hilfe der Logarithmusfunktion
res:= `q_log`(x) $\boxdot$ `q_l2i`

Nun gilt `q_log2` := res $\approx \log_2 x$.

5.7.3 Fehlerabschätzung

Die Fehlerabschätzung wird mit dem Programm `fflog2.p` durchgeführt, man erhält als Ergebnis:

```
rel. Fehlerschranke der fflog2-Funktion: 2.775305637122755E-015
```

Gesamtfehlerschranke:
Relative Fehlerschranke für `q_log2`$(x) \in S_N$: $\varepsilon(\texttt{q\_log2}) \leq 2.7754E-015 = 25.0 \cdot \varepsilon^*$

5.7.4 Algorithmus `j_log2` (Intervallfunktion)

Der Gesamtalgorithmus `j_log2` zur Berechnung von $\log_2(X)$ für Intervallargumente $X := [x.\mathrm{INF}, x.\mathrm{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_lg2m} := (1 \bigtriangledown \varepsilon(\texttt{q\_log2})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_lg2p} := (1 \bigtriangleup \varepsilon(\texttt{q\_log2})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. X ist Punktintervall, d.h. $x.\mathrm{INF} = x.\mathrm{SUP}$

a) $x.\mathrm{INF} < 0$:
$y :=$ `q_log2`$(x.\mathrm{INF})$
res.INF:= $y \boxdot$ `q_lg2p`
res.SUP:= $y \boxdot$ `q_lg2m`

b) $x.\mathrm{INF} \geq 0$:
$y :=$ `q_log2`$(x.\mathrm{INF})$
res.INF:= $y \boxdot$ `q_lg2m`
res.SUP:= $y \boxdot$ `q_lg2p`

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\mathrm{INF} < 0$:
res.INF:= `q_log2`$(x.\mathrm{INF}) \boxdot$ `q_lg2p`

ii. $x.\mathrm{INF} \geq 0$:
res.INF:= `q_log2`$(x.\mathrm{INF}) \boxdot$ `q_lg2m`

b) Berechnung der Oberschranke res.SUP:

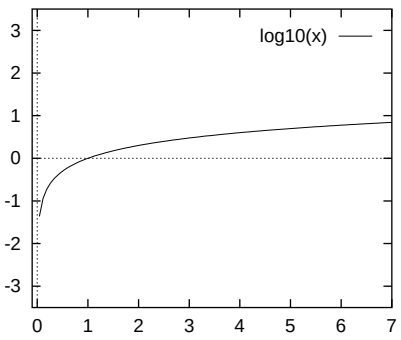
i. $x.\mathrm{SUP} < 0$:
res.SUP:= `q_log2`$(x.\mathrm{SUP}) \boxdot$ `q_lg2m`

ii. $x.\mathrm{SUP} \geq 0$:
res.SUP:= `q_log2`$(x.\mathrm{SUP}) \boxdot$ `q_lg2p`

Nun gilt `j_log2` := res $\supseteq \log_2(X)$ mit res := [res.INF, res.SUP].

5.8 Logarithmusfunktion zur Basis 10: $\log_{10} x$

Logarithmusfunktion zur Basis 10	
Math. Schreibweise	$\log_1 0x$
Definitionsbereich	$(0, \infty)$
1. Ableitung	$\frac{1}{x \ln 10}$
Nullstellen	$x_0 = 1$
Monotonie	monoton steigend



5.8.1 Idee

Die Funktion $\log_{10} x$ wird mit Hilfe der Formel

$$\log_{10} x = \ln x \cdot \frac{1}{ln10}$$

auf die Berechnung des natürlichen Logarithmus zurückgeführt. Der Wert $\frac{1}{ln10}$ wird im voraus berechnet und gerundet zur nächstgelegenen Gleitkommazahl als Konstante unter dem Namen `q_l10i` abgespeichert.

5.8.2 Algorithmus `q_lg10` (Punktfunktion)

Der Gesamtalgorithmus `q_lg10` zur Berechnung von $\log_{10} x$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument

II. Berechnung mit Hilfe der Logarithmusfunktion
res:= `q_log`(x) $\boxdot$ `q_l10i`

Nun gilt `q_lg10` := res $\approx \log_{10} x$.

5.8.3 Fehlerabschätzung

Die Fehlerabschätzung wird mit dem Programm `fflog10.p` durchgeführt, man erhält als Ergebnis:

```
rel. Fehlerschranke der fflog10-Funktion: 2.775305637122755E-015
```

Gesamtfehlerschranke:
Relative Fehlerschranke für `q_lg10`$(x) \in S_N$: $\varepsilon(\texttt{q\_lg10}) \leq 2.7754E-015 = 25.0 \cdot \varepsilon^*$

5.8.4 Algorithmus `j_lg10` (Intervallfunktion)

Der Gesamtalgorithmus `j_lg10` zur Berechnung von $\log_{10}(X)$ für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_l10m} := (1 \bigtriangledown \varepsilon(\texttt{q\_lg10})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_l10p} := (1 \bigtriangleup \varepsilon(\texttt{q\_lg10})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\text{INF} < 0$:
y := `q_lg10`$(x.\text{INF})$
res.INF:= y $\boxdot$ `q_l10p`
res.SUP:= y $\boxdot$ `q_l10m`

b) x.INF ≥ 0:
$y :=$ `q_lg10`$(x.\mathrm{INF})$
res.INF:= $y \boxdot$ `q_l10m`
res.SUP:= $y \boxdot$ `q_l10p`

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. x.INF < 0:
res.INF:= `q_lg10`$(x.\mathrm{INF}) \boxdot$ `q_l10p`

ii. x.INF ≥ 0:
res.INF:= `q_lg10`$(x.\mathrm{INF}) \boxdot$ `q_l10m`

b) Berechnung der Oberschranke res.SUP:

i. x.SUP < 0:
res.SUP:= `q_lg10`$(x.\mathrm{SUP}) \boxdot$ `q_l10m`

ii. x.SUP ≥ 0:
res.SUP:= `q_lg10`$(x.\mathrm{SUP}) \boxdot$ `q_l10p`

Nun gilt `j_lg10`$| := \mathrm{res} \supseteq \log_{10}(X)$ mit res := [res.INF, res.SUP].

Kapitel 6

Trigonometrische Funktionen

Für die trigonometrischen Funktionen Sinus und Cosinus ist es ausreichend, eine Approximation für Eingabeargumente x mit $x \in [-\pi/2, \pi/2]$ zur Verfügung zu stellen. Zu allen anderen Eingabeargumenten x kann ein reduziertes Argument r mit $r \in [-\pi/2, \pi/2]$ berechnet werden. Es werden dabei die Periodizität der Sinus- bzw. Cosinusfunktion

$$\sin(x) = \sin(2k\pi + x), \quad x \in I\!R, k \in \mathbb{Z},$$

$$\cos(x) = \cos(2k\pi + x), \quad x \in I\!R, k \in \mathbb{Z},$$

und die Reduktionsformeln

$$\sin(\pi/2 + x) = \cos(x), \quad \sin(\pi + x) = -\sin(x)$$

$$\cos(\pi/2 + x) = -\sin(x), \quad \cos(\pi + x) = -\cos(x)$$

ausgenutzt.

Um den Aufwand bei der Funktionsberechnung auf dem Rechner möglichst klein zu halten wird der maximal mögliche Definitionsbereich $[-\texttt{maxreal}, \texttt{maxreal}]$ eingeschränkt auf den Bereich $[-\texttt{trimax}, \texttt{trimax}]$ mit $\texttt{trimax} := 2^{31} \cdot \frac{\pi}{2} + \frac{\pi}{4}$. Bei den Punktfunktionen wird für Argumente $x >$ `trimax` eine Fehlermeldung („invalid argument“) ausgegeben. Die Sinus- bzw. die Cosinusintervallfunktion liefert für Argumente $X = [\texttt{X.inf}, \texttt{X.sup}]$ mit `X.inf` $< -$`trimax` oder `X.sup` $>$`trimax` das Ergebnisintervall $[-1, 1]$.

Bei der Ausführung der Argumentreduktion auf dem Rechner muß aufgrund möglicher Auslöschung sehr sorgfältig vorgegangen werden. Dies wird im folgenden Abschnitt beschrieben.

6.1 Argumentreduktion

6.1.1 Idee

Ausgehend von einem gegebenen Argument $x \in S$ mit $|x| <$`trimax` wird eine möglichst gute Näherung $r \in S$ für $x - k \cdot \frac{\pi}{2}$ gesucht. Die ganze Zahl k wird dabei durch $k := \text{round}(x \cdot$ `q_pi2i` $)$ bestimmt, die Konstante stellt die dem Wert $\frac{2}{\pi}$ nächstgelegene Maschinenzahl dar.

Da sich bei der Subtraktion $x - k \cdot \frac{\pi}{2}$ führende Ziffern auslöschen können, muß die Näherung für $\frac{\pi}{2}$ auf der Maschine höhergenau, d.h. mit ausreichend vielen Mantissenziffern dargestellt werden. Hierzu die folgenden Überlegungen:

Bekannt ist, daß für den Abstand d einer Maschinenzahl x ($x \neq 0$) zu einem ganzzahligen Vielfachen von $\frac{\pi}{2}$ gilt: $d \geq 4.6871 \cdot 10^{-19}$ (siehe z.B. [33]). Daraus folgt für das reduzierte Argument:

$$|r| \geq d \geq 4.6871 \cdot 10^{-19} \geq 2^{-61}.$$

Bei der Berechnung von r können sich damit maximal $32+61 = 93$ Bits auslöschen. Damit nach der Ausführung der Subtraktion eine volle Mantissenlänge (53 Bits) des Ergebnisses zur Verfügung steht, muß die Näherung für $\frac{\pi}{2}$ mit mindestens $53 + 93 = 146$ Binärziffern Genauigkeit zur Verfügung stehen.

6.1.2 Berechnung der Konstanten $\pi/2$

Die benötigte Näherung für die Konstante $\frac{\pi}{2}$ wird mit Hilfe der Langzahlintervallarithmetik `mpi_ari` von Pascal–XSC berechnet. Zur Speicherung wird die Binärdarstellung der Näherung in 7 Portionen mit jeweils 22 Bits aufgeteilt, d.h. es werden $7 \cdot 22 = 154$ Binärstellen gespeichert. Die 7 Portionen werden in einem Feld `q_pih[.]` mit 7 Speicherplätzen vom Typ `double` abgelegt.

Diese Art der Speicherung ermöglicht es später, eine Portion der Konstanten mit einer 31–stelligen Binärzahl zu multiplizieren und das Ergebnis rundungsfehlerfrei im Datenformat `double` abzuspeichern.

6.1.3 Algorithmus `q_rtrg(x,k)` (Argumentreduktion)

Der Algorithmus `q_rtrg` zur Argumentreduktion sieht wie folgt aus:

I) Falls $|k| < 512$ dann:

$$\begin{aligned} r2 &:= x - k \cdot (\texttt{q\_pih[0]} + \texttt{q\_pih[1]}) \\ res &:= \texttt{q\_r2tr}(r2, k) \end{aligned}$$

II) Falls $|k| \geq 512$ dann:

$$\begin{aligned} r1 &:= x - k \cdot \texttt{q\_pih[0]} \\ h &:= k \cdot \texttt{q\_pih[1]} \\ r2 &:= r1 \boxminus h \end{aligned}$$

a) Falls $\text{expo}(r1) = \text{expo}(r2)$ dann:

$$\begin{aligned} \text{res} := r1 \boxminus ((((&(k \cdot \texttt{q\_pih[6]} \boxplus k \cdot \texttt{q\_pih[5]}) \boxplus k \cdot \texttt{q\_pih[4]}) \\ &\boxplus k \cdot \texttt{q\_pih[3]}) \boxplus k \cdot \texttt{q\_pih[2]}) \boxplus h) \end{aligned}$$

b) Sonst:
res := $\texttt{q\_r2tr}(r2, k)$

6.1.4 Algorithmus `q_r2tr(x,k)` (Hilfsfunktion)

Der Algorithmus der Hilfsfunktion `q_r2tr` zur Argumentreduktion sieht wie folgt aus:

$r2 := r$

$h := k \cdot \texttt{q\_pih[2]}$

$r1 := r2 \boxminus h$

I) Falls expo$(r1)$ = expo$(r2)$ dann:

$$\text{res} := r2 \boxminus ((((k \cdot \texttt{q\_pih[6]} \boxplus k \cdot \texttt{q\_pih[5]}) \boxplus k \cdot \texttt{q\_pih[4]}) \boxplus k \cdot \texttt{q\_pih[3]}) \boxplus h)$$

II) Sonst:

$h := k \cdot \texttt{q\_pih[3]}$

$r2 := r1 \boxminus h$

a) Falls expo$(r1)$ = expo$(r2)$ dann:

$$\text{res} := r1 \boxminus (((k \cdot \texttt{q\_pih[6]} \boxplus k \cdot \texttt{q\_pih[5]}) \boxplus k \cdot \texttt{q\_pih[4]}) \boxplus h)$$

b) Sonst:

$h := k \cdot \texttt{q\_pih[4]}$

$r1 := r2 \boxminus h$

i. Falls expo$(r1)$ = expo$(r2)$ dann:

$$\text{res} := r1 \boxminus ((k \cdot \texttt{q\_pih[6]} \boxplus k \cdot \texttt{q\_pih[5]}) \boxplus h)$$

ii. Sonst:

$h := k \cdot \texttt{q\_pih[5]}$

$r2 := r1 \boxminus h$

A. Falls expo$(r1)$ = expo$(r2)$ dann:

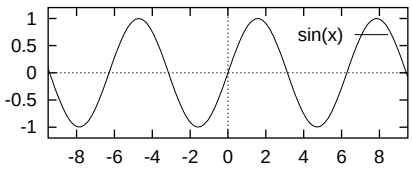
$$\text{res} := r1 \boxminus (k \cdot \texttt{q\_pih[6]} \boxplus h)$$

B. Sonst:

$$\text{res} := r2 \boxminus k \cdot \texttt{q\_pih[6]}$$

6.2 Sinusfunktion: $\sin x$

Sinusfunktion	
Math. Schreibweise	$\sin(x)$
Definitionsbereich	$\mathbb{R}$
Potenzreihe für $\lvert x\rvert \leq \infty$	$\sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)!}$ $= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + - \ldots$
1. Ableitung	$\cos(x)$
Nullstellen	$x_n = n \cdot \pi$ für $n \in \mathbb{Z}$
Minima	$x_{Tn} = -\frac{\pi}{2} + 2\pi n$ für $n \in \mathbb{Z}$
Maxima	$x_{Hn} = \frac{\pi}{2} + 2\pi n$ für $n \in \mathbb{Z}$



6.2.1 Algorithmus q_sin (Punktfunktion)

Der Gesamtalgorithmus `q_sin` zur Berechnung von $\sin x$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| >$`q_sint[2]` $= 3.3732\ldots e+09 \Longrightarrow$ Fehlermeldung: Invalid Argument

II. Argumentreduktion

$$\begin{aligned} y &:= x \boxdot \texttt{q\_pi2i} \\ k &:= \text{round}(y) \\ y &:= \texttt{q\_rtrg}(x,k) \\ n &:= k \bmod 4 \\ m &:= n \bmod 2 \end{aligned}$$

III. Approximation
ysq $:= y \boxdot y$

a) Falls $(m = 0)$ dann: (Approximation der Sinus-Funktion)

i. Falls $|x| <$ `q_sint[3]` $= 2.5809e-8$ dann:

A. Falls $(n = 0)$ dann:
res $:= y$

B. Falls $(n \neq 0)$ dann:
res $:= -y$

ii. Falls $|x| \geq$ `q_sint[3]` $= 2.5809e-8$ dann:
$q :=$ ysq $\boxdot (s_0 \boxplus$ ysq $\boxdot (s_1 \boxplus$ ysq $\boxdot (s_2 \boxplus$ ysq $\boxdot (s_3 \boxplus$ ysq $\boxdot (s_4 \boxplus$ ysq $\boxdot s_5)))))$

A. Falls $(n = 0)$ dann:
res $:= y \boxplus y \boxdot q$

B. Falls $(n \neq 0)$ dann:
res $:= -(y \boxplus y \boxdot q)$

b) Falls $(m \neq 0)$ dann: (Approximation der Cosinus-Funktion)
$q :=$ ysq $\boxdot$ ysq $\boxdot (c_0 \boxplus$ ysq $\boxdot (c_1 \boxplus$ ysq $\boxdot (c_2 \boxplus$ ysq $\boxdot (c_3 \boxplus$ ysq $\boxdot (c_4 \boxplus$ ysq $\boxdot c_5)))))$

i. Falls (ysq $\geq$ `q_sint[0]`) dann:
res $:= 0.625 \boxplus (0.375 \boxminus (0.5 \cdot$ ysq$) \boxplus q)$

ii. Falls (`q_sint[0]` $>$ ysq $\geq$ `q_sint[1]`) dann:
res $:= 0.8125 \boxplus ((0.1875 \boxminus (0.5 \cdot$ ysq$)) \boxplus q)$

iii. Falls (`q_sint[1]` $>$ ysq) dann:
res $:= 1.0 \boxminus (0.5 \cdot$ ysq $\boxminus q)$

Falls $(n = 3)$ dann:
res $:=$ res

Nun gilt `q_sin` $:=$ res $\approx \sin x$.

6.2.2 Fehlerabschätzung

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_sin}(x) \in S_N$: $\varepsilon(\texttt{q\_sin}) \leq 1.0718E-015 = 9.66 \cdot \varepsilon^*$

6.2.3 Algorithmus `j_sin` (Intervallfunktion)

Der Gesamtalgorithmus `j_sin` zur Berechnung von $\sin(X)$ für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_sinm} := (1 \bigtriangledown \varepsilon(\texttt{q\_sin})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_sinp} := (1 \bigtriangleup \varepsilon(\texttt{q\_sin})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\text{INF} < -\texttt{q\_sint[2]}$:
res.INF:= -1.0
res.SUP:= 1.0

b) $\texttt{q\_sint[2]} < x.\text{INF}$:
res.INF:= -1.0
res.SUP:= 1.0

c) Sonst:
$y := \texttt{q\_cos}(x.\text{INF})$

i. $y < 0$:
res.INF:= $y \boxdot \texttt{q\_sinp}$
res.SUP:= $y \boxdot \texttt{q\_sinm}$

ii. $y \geq 0$:
res.INF:= $y \boxdot \texttt{q\_sinm}$
res.SUP:= $y \boxdot \texttt{q\_sinp}$

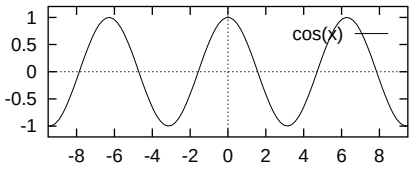
II. X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$

a) Intervall-Logik der Sinus-/Cosinusfunktion

Nun gilt $\texttt{j\_sin} := \text{res} \supseteq \sin(X)$ mit $\text{res} := [\text{res.INF}, \text{res.SUP}]$.

6.3 Cosinusfunktion: $\cos x$

Cosinusfunktion	
Math. Schreibweise	$\cos(x)$
Definitionsbereich	$\mathbb{R}$
Potenzreihe für $\lvert x\rvert \leq \infty$	$\sum_{k=0}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!}$ $= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + - \ldots$
1. Ableitung	$-\sin(x)$
Nullstellen	$x_n = \frac{\pi}{2} + n \cdot \pi$ für $n \in \mathbb{Z}$
Minima	$x_{Tn} = (2n+1)\pi$ für $n \in \mathbb{Z}$
Maxima	$x_{Hn} = 2n\pi$ für $n \in \mathbb{Z}$



6.3.1 Algorithmus q_cos (Punktfunktion)

Der Gesamtalgorithmus `q_cos` zur Berechnung von $\cos(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| >$`q_sint[2]` $= 3.3732\ldots e+09 \Longrightarrow$ Fehlermeldung: Invalid Argument

II. Argumentreduktion

$$\begin{aligned} y &:= x \boxdot \texttt{q\_pi2i} \\ k &:= \text{round}(y) \\ y &:= \texttt{q\_rtrg}(x,k) \\ n &:= (k+1) \bmod 4 \\ m &:= n \bmod 2 \end{aligned}$$

III. Approximation
ysq $:= y \boxdot y$

a) Falls $(m = 0)$ dann: (Approximation der Sinus-Funktion)

i. Falls $|x| <$ `q_sint[3]` $= 2.5809e-8$ dann:

A. Falls $(n = 0)$ dann:
res $:= y$

B. Falls $(n \neq 0)$ dann:
res $:= -y$

ii. Falls $|x| \geq$ `q_sint[3]` $= 2.5809e-8$ dann:
$q :=$ ysq $\boxdot (s_0 \boxplus$ ysq $\boxdot (s_1 \boxplus$ ysq $\boxdot (s_2 \boxplus$ ysq $\boxdot (s_3 \boxplus$ ysq $\boxdot (s_4 \boxplus$ ysq $\boxdot s_5)))))$

A. Falls $(n = 0)$ dann:
res $:= y \boxplus y \boxdot q$

B. Falls $(n \neq 0)$ dann:
res $:= -(y \boxplus y \boxdot q)$

b) Falls $(m \neq 0)$ dann: (Approximation der Cosinus-Funktion)
$q :=$ ysq$\boxdot$ysq$\boxdot(c_0 \boxplus$ysq$\boxdot(c_1 \boxplus$ysq$\boxdot(c_2 \boxplus$ysq$\boxdot(c_3 \boxplus$ysq$\boxdot(c_4 \boxplus$ysq$\boxdot c_5))))))$

i. Falls (ysq $\geq$ `q_sint[0]`) dann:
res $:= 0.625 \boxplus (0.375 \boxminus (0.5 \cdot \text{ysq}) \boxplus q)$

ii. Falls (`q_sint[0]` $>$ ysq $\geq$ `q_sint[1]`) dann:
res $:= 0.8125 \boxplus ((0.1875 \boxminus (0.5 \cdot \text{ysq})) \boxplus q)$

iii. Falls (`q_sint[1]` $>$ ysq) dann:
res $:= 1.0 \boxminus (0.5 \cdot \text{ysq} \boxminus q)$

Falls $(n = 3)$ dann:
res $:= -$res

Nun gilt `q_cos` := res $\approx \cos(x)$.

6.3.2 Fehlerabschätzung

Gesamtfehlerschranke:
Relative Fehlerschranke für `q_cos`$(x) \in S_N$: $\varepsilon($`q_cos`$) \leq 1.0718E-015 = 9.66 \cdot \varepsilon^*$

6.3.3 Algorithmus `j_cos` (Intervallfunktion)

Der Gesamtalgorithmus `j_cos` zur Berechnung von $\cos(X)$ für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

`q_sinm` := $(1 \bigtriangledown \varepsilon($`q_sin`$)) \bigtriangledown (1 \bigtriangledown$ `EpsQuer`$)$
`q_sinp` := $(1 \bigtriangleup \varepsilon($`q_sin`$)) \bigtriangleup (1 \bigtriangleup$ `EpsQuer`$)$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\text{INF} <$ `-q_sint[2]`:
res.INF:= -1.0
res.SUP:= 1.0

b) `q_sint[2]` $< x.\text{INF}$:
res.INF:= -1.0
res.SUP:= 1.0

c) Sonst:
$y :=$ `q_cos`$(x.\text{INF})$

i. $y < 0$:
res.INF:= $y \boxdot$ `q_sinp`
res.SUP:= $y \boxdot$ `q_sinm`

ii. $y \geq 0$:
res.INF:= $y \boxdot$ `q_sinm`
res.SUP:= $y \boxdot$ `q_sinp`

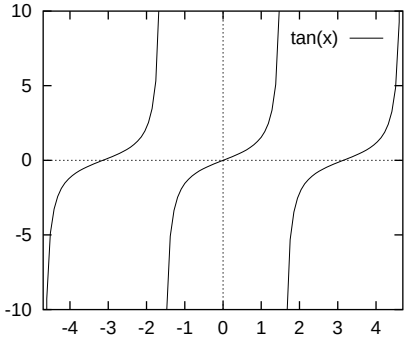
II. X ist echtes Intervall, d.h. x.INF $\neq x$.SUP

a) Intervall-Logik der Sinus-/Cosinusfunktion

Nun gilt `j_cos` := res $\supseteq \cos(X)$ mit res := [res.INF, res.SUP].

6.4 Tangensfunktion: $\tan x$

Tangensfunktion	
Math. Schreibweise	$\tan(x) = \frac{\sin(x)}{\cos(x)}$
Definitionsbereich	$\cup_{n \in Z}(-\frac{\pi}{2} + n\pi, \frac{\pi}{2} + n\pi)$
1. Ableitung	$\frac{1}{(\cos(x))^2}$
Nullstellen	$x_n = n \cdot \pi$ für $n \in \mathbb{Z}$



6.4.1 Algorithmus `q_tan` (Punktfunktion)

Der Gesamtalgorithmus `q_tan` zur Berechnung von $\tan(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number)
  $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| >$`q_sint[2]` $= 3.3732\ldots e+09$
  $\Longrightarrow$ Fehlermeldung: Invalid Argument

II. Argumentreduktion

$$\begin{aligned} y &:= x \boxdot \texttt{q\_pi2i} \\ k &:= \text{round}(y) \\ y &:= \texttt{q\_rtrg}(x, k) \\ n &:= k \bmod 4 \\ m &:= n \bmod 2 \\ \text{ysq} &:= y \boxdot y \end{aligned}$$

III. Approximation

a) Falls $|x| <$ `q_sint[4]` $= 2.5809e-8$ dann:

i. Falls $(m = 0)$ dann:
res $:= y$

ii. Falls $(m \neq 0)$ dann:
res $:= -1 \boxplus y$

b) Falls $|x| \leq$ `q_sint[4]` dann:

i. Approximation der Sinus-Funktion

A. Falls $|x| <$ `q_sint[3]` $= 1.825e-8$ dann:
1) Falls $(n = 0)$ dann:
s $:= y$
2) Falls $(n \neq 0)$ dann:
s $:= -y$

B. Falls $|x| \geq$ `q_sint[3]` dann:
$q :=$ ysq$\boxdot(s_0 \boxplus$ysq$\boxdot(s_1 \boxplus$ysq$\boxdot(s_2 \boxplus$ysq$\boxdot(s_3 \boxplus$ysq$\boxdot(s_4 \boxplus$ysq$\boxdot s_5))))))$
1) Falls $(n = 0)$ dann:
s $:= y \boxplus y \boxdot q$
2) Falls $(n \neq 0)$ dann:
s $:= -(y \boxplus y \boxdot q)$

ii. Approximation der Cosinus-Funktion
$q :=$ ysq$\boxdot$ysq$\boxdot(c_0 \boxplus$ysq$\boxdot(c_1 \boxplus$ysq$\boxdot(c_2 \boxplus$ysq$\boxdot(c_3 \boxplus$ysq$\boxdot(c_4 \boxplus$ysq$\boxdot c_5))))))$

A. Falls (ysq $\geq$ `q_sint[0]`) dann:
c $:= 0.625 \boxplus (0.375 \boxminus (0.5 \cdot$ ysq$) \boxplus q)$

B. Falls (`q_sint[0]` $>$ ysq $\geq$ `q_sint[1]`) dann:
c $:= 0.8125 \boxplus ((0.1875 \boxminus (0.5 \cdot$ ysq$)) \boxplus q)$

C. Falls (`q_sint[1]` $>$ ysq) dann:
c $:= 1.0 \boxminus (0.5 \cdot$ ysq $\boxminus q)$

Falls $(n = 3)$ dann:
c $:=$ −c

iii. Berechnung des Tangens

A. Falls $(m = 0)$ dann:
res $:= s \boxplus c$

B. Falls $(m \neq 0)$ dann:
res $:= -c \boxplus s$

Nun gilt `q_tan` $:=$ res $\approx \tan(x)$.

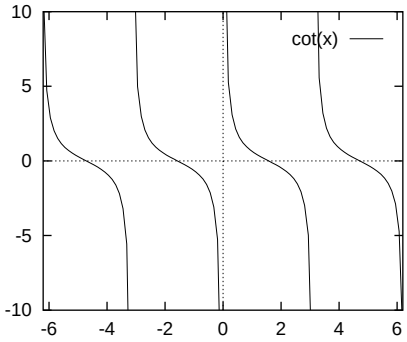
6.4.2 Fehlerabschätzung

Gesamtfehlerschranke:
Relative Fehlerschranke für `q_tan`$(x) \in S_N$: $\varepsilon($`q_tan`$) \leq 2.9777E-015 = 26.83 \cdot \varepsilon^*$

6.4.3 Algorithmus `j_tan` (Intervallfunktion)

6.5 Cotangensfunktion: $\cot x$

Cotangensfunktion	
Math. Schreibweise	$\cot(x) = \frac{\cos(x)}{\sin(x)}$
Definitionsbereich	$\cup_{n \in Z}(n\pi, (n+1)\pi)$
1. Ableitung	$-\frac{1}{(\sin(x))^2}$
Nullstellen	$x_n = \frac{\pi}{2} + n \cdot \pi$ für $n \in \mathbb{Z}$



6.5.1 Algorithmus `q_cot` (Punktfunktion)

Der Gesamtalgorithmus `q_cot` zur Berechnung von $\cot(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number)
  $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| >$`q_sint[2]` $= 3.3732\ldots e+09$
  $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| <$`q_minr`
  $\Longrightarrow$ Fehlermeldung: Invalid Argument

II. Argumentreduktion

$$\begin{aligned} y &:= x \boxdot \texttt{q\_pi2i} \\ k &:= \text{round}(y) \\ y &:= \texttt{q\_rtrg}(x,k) \\ n &:= k \bmod 4 \\ m &:= n \bmod 2 \\ \text{ysq} &:= y \boxdot y \end{aligned}$$

III. Approximation der Sinus-Funktion

a) Falls $|x| <$ `q_sint[3]` $= 2.5809e-8$ dann:

i. Falls $(n = 0)$ dann:
s $:= y$

ii. Falls $(n \neq 0)$ dann:
s $:= -y$

b) Falls $|x| \geq$ `q_sint[3]` dann:
$q :=$ ysq $\boxdot (s_0 \boxplus$ ysq $\boxdot (s_1 \boxplus$ ysq $\boxdot (s_2 \boxplus$ ysq $\boxdot (s_3 \boxplus$ ysq $\boxdot (s_4 \boxplus$ ysq $\boxdot s_5))))))$

i. Falls $(n = 0)$ dann:
s $:= y \boxplus y \boxdot q$

ii. Falls $(n \neq 0)$ dann:
s $:= -(y \boxplus y \boxdot q)$

IV. Approximation der Cosinus-Funktion
$q :=$ ysq $\boxdot$ ysq $\boxdot (c_0 \boxplus$ ysq $\boxdot (c_1 \boxplus$ ysq $\boxdot (c_2 \boxplus$ ysq $\boxdot (c_3 \boxplus$ ysq $\boxdot (c_4 \boxplus$ ysq $\boxdot c_5))))))$

a) Falls (ysq $\geq$ `q_sint[0]`) dann:
c $:= 0.625 \boxplus (0.375 \boxminus (0.5 \cdot$ ysq$) \boxplus q)$

b) Falls (`q_sint[0]` $>$ ysq $\geq$ `q_sint[1]`) dann:
c $:= 0.8125 \boxplus ((0.1875 \boxminus (0.5 \cdot$ ysq$)) \boxplus q)$

c) Falls (`q_sint[1]` $>$ ysq) dann:
c $:= 1.0 \boxminus (0.5 \cdot$ ysq $\boxminus q)$

Falls $(n = 2)$ dann:
c $:= -$c

V. Berechnung des Cotangens

a) Falls $(m = 0)$ dann:
res $:= c \boxslash s$

b) Falls $(m \neq 0)$ dann:
res $:= s \boxslash c$

Nun gilt `q_cot` $:=$ res $\approx \cot(x)$.

6.5.2 Fehlerabschätzung

Gesamtfehlerschranke:
Relative Fehlerschranke für `q_cot`$(x) \in S_N$: $\varepsilon($`q_cot`$) \leq 2.9777E-015 = 26.83 \cdot \varepsilon^*$

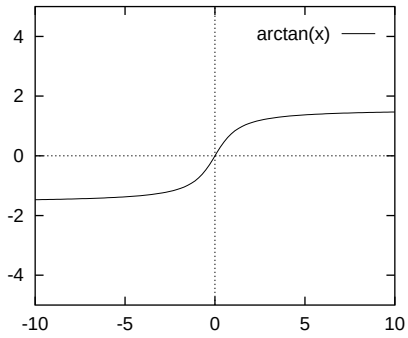
Kapitel 7

Inverse Trigonometrische Funktionen

Da die Berechnung der Funktionen $\arcsin(x)$, $\arccos(x)$ und $\operatorname{arccot}(x)$ auf die Berechnung der Funktion $\arctan(x)$ zurückgeführt wird, erfolgt zunächst die Beschreibung der Arcustangensfunktion.

7.1 Arcustangensfunktion: $\arctan(x)$

Arcustangensfunktion	
Math. Schreibweise	$\arctan x$
Definitionsbereich	$\mathbb{R}$
Potenzreihe für $\lvert x\rvert < 1$	$\sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{2k+1}$ $= x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + - \ldots$
1. Ableitung	$\frac{1}{1+x^2}$
Nullstellen	$x_0 = 0$
Symmetrie	Punktsymmetrie zu $(0,0)$
Monotonie	monoton steigend



7.1.1 Idee

Das in [28] beschriebene Verfahren wird hier modifiziert und auf das IEEE-Datenformat angewandt.

Positive Eingabeargumente $x \in I_1 := [0, \frac{1}{\gamma}]$ werden mit Hilfe der bekannten Summenformel

$$\arctan x = \arctan \frac{x - c_i}{1 + xc_i} + \arctan c_i, \quad xc_i > -1$$

und geeignet gewählter Reduktionskonstanten c_i in ein festgelegtes Approximationsintervall $I_\gamma := [-\gamma, \gamma]$ reduziert. Die benötigten Konstanten $a_i := \arctan c_i$ zur Ergebnisanpassung können im voraus berechnet und tabelliert werden.

Für Argumente $r \in I_\gamma$ wird zur Approximation des arctan im Unterschied zu [28] nicht die abgebrochene Potenzreihe

$$\arctan(r) \approx r \cdot \sum_{k=0}^{N} \frac{(-1)^k}{2k+1} r^{2k}$$

mit Polynomgrad N verwendet, sondern die polynomiale Approximation

$$\arctan(r) \approx r \cdot (1 + r^2 \cdot p_N(r^2)) \tag{7.1}$$

mit einer Bestapproximation $p_N(r^2) = \sum_{k=0}^{N} d_k \cdot r^{2k}$.

Für sehr kleine Argumente $|r| <$`q_atnt` kann die Approximation

$$\arctan(r) \approx r$$

verwendet werden. Die Berechnung für Eingabeargumente $x \in I_2 := [\frac{1}{\gamma}, \texttt{maxreal}]$ kann mit

$$\arctan(x) = \arctan(-\frac{1}{x}) + \frac{\pi}{2}$$

ebenfalls auf die Approximation (7.1) zurückgeführt werden, es gilt $r := \frac{-1}{x} \in I_\gamma$. Für negative Argumente kann die Formel

$$\arctan(-x) = -\arctan(x)$$

angewandt werden.

7.1.2 Bestimmung der benötigten Konstanten

Im folgenden wird für den Radius des Approximationsintervalls der Wert $\gamma := \frac{1}{8}$ verwendet.

I) Aufteilen von I_1 in Teilintervalle und Bestimmung der Reduktionskonstanten:
Das Intervall I_1 wird in disjunkte Teilintervalle $[b_i, b_{i+1})$ mit $\cup_i [b_i, b_{i+1}) = I_1$ unterteilt, welche mit Hilfe von geeigneten Reduktionskonstanten c_i in das Intervall I_x abgebildet werden können.
Ausgehend vom ersten Teilintervall mit $b_0 := 0$, $b_1 := \gamma$ und $c_0 := 0$ (in diesem Teilintervall wird keine Reduktion durchgeführt) können die weiteren Reduktionskonstanten c_i, sowie die weiteren Endpunkte b_i der Teilintervalle mit Hilfe der folgenden Formeln sukzessive berechnete werden:

$$\begin{aligned} c_{i-1} &:= \frac{b_{i-1} + \gamma}{1 - \gamma b_{i-1}}, \quad i = 2(1)6, \\ b_i &:= \frac{\gamma + c_{i-1}}{1 - \gamma c_{i-1}}, \quad i = 2(1)7 \end{aligned}$$

Die Berechnung endet, wenn erstmals ein b_i mit $b_i \geq \frac{1}{\gamma}$ berechnet wird. Die Konstanten werden mit Hilfe einer Langzahlarithmetik bestimmt und dann zur nächstgelegenen Gleitkommazahl des double-Formates gerundet, d.h. sie stehen maximal genau zur Verfügung.

II) Ergebnisanpassungskonstanten:
Die Berechnung erfolgt unter Verwendung einer Langzahlarithmetik über

$$a_i := \arctan(c_i),$$

die Konstanten stehen maximal genau zur Verfügung.

III) Koeffizienten des Approximationspolynoms:
Die benötigen Koeffizienten d_i werden mit Hilfe eines Computeralgebrapaketes (z.B. Maple) bestimmt und anschließend in das double-Format gerundet.

Alle benötigten Konstanten können mit dem PASCAL-XSC Programm `atankoef.p` berechnet werden, man erhält mit den Bezeichnungen `q_atna[`i`]`$:= a_i$, `q_atnb[`i`]`$:= b_i$, `q_atnc[`i`]`$:= c_i$ und `q_atnd[`i`]`$:= d_i$:

```
q_atna[0]:=  0;                                        { =  0.00000... }
q_atna[1]:=  8960721713639278.0 / 36028797018963968.0; { =  0.24871... }
q_atna[2]:=  8960721713639278.0 / 18014398509481984.0; { =  0.49742... }
q_atna[3]:=  6720541285229458.0 /  9007199254740992.0; { =  0.74613... }
q_atna[4]:=  8960721713639278.0 /  9007199254740992.0; { =  0.99484... }
q_atna[5]:=  5600451071024549.0 /  4503599627370496.0; { =  1.24355... }
q_atna[6]:=  6720541285229458.0 /  4503599627370496.0; { =  1.49226... }

q_atnb[0]:=  0;                                        { =  0.00000... }
q_atnb[1]:=  4503599627370496.0 / 36028797018963968.0; { =  0.12500... }
q_atnb[2]:=  7050717449407908.0 / 18014398509481984.0; { =  0.39139... }
q_atnb[3]:=  6454487157372003.0 /  9007199254740992.0; { =  0.71659... }
q_atnb[4]:=  5343484307627230.0 /  4503599627370496.0; { =  1.18649... }
q_atnb[5]:=  4642583338069965.0 /  2251799813685248.0; { =  2.06172... }
q_atnb[6]:=  5472919125137495.0 /  1125899906842624.0; { =  4.86093... }
q_atnb[7]:=  4503599627370496.0 /   562949953421312.0; { =  8.00000... }

q_atnc[0]:=  0;                                        { =  0.00000... }
q_atnc[1]:=  4575085335741456.0 / 18014398509481984.0; { =  0.25397... }
q_atnc[2]:=  4890523484394786.0 /  9007199254740992.0; { =  0.54296... }
q_atnc[3]:=  8326197945442628.0 /  9007199254740992.0; { =  0.92439... }
q_atnc[4]:=  6934969934934130.0 /  4503599627370496.0; { =  1.53987... }
q_atnc[5]:=  6633650527568543.0 /  2251799813685248.0; { =  2.94593... }
q_atnc[6]:=  7153270512133541.0 /   562949953421312.0; { = 12.70676... }

q_atnd[0]:= -6004799503160653.0 / 18014398509481984.0; { = -0.33333... }
q_atnd[1]:=  7205759403717662.0 / 36028797018963968.0; { =  0.20000... }
q_atnd[2]:= -5146970946471129.0 / 36028797018963968.0; { = -0.14286... }
q_atnd[3]:=  8006368526669049.0 / 72057594037927936.0; { =  0.11111... }
q_atnd[4]:= -6546869189017288.0 / 72057594037927936.0; { = -0.09086... }
q_atnd[5]:=  5323534481176937.0 / 72057594037927936.0; { =  0.07388... }

q_pih:=      7074237752028440.0 /  4503599627370496.0; { =  1.57080... }
```

7.1.3 Abschätzung des Approximationsfehlers

Da das reduzierte Argument

$$r := r(x) := \frac{x - c_i}{1 + xc_i} \quad \text{mit } x \in [b_i, b_{i+1})$$

mit der Eigenschaft $r \in I_\gamma$ auf der Maschine nicht exakt berechnet werden kann, steht nur das fehlerbehaftete reduzierte Argument

$$\tilde{r} = r + \triangle_r \quad \text{mit } | \triangle_r | \leq \triangle(r)$$

zur Verfügung. Aus diesem Grund muß im folgenden anstelle des gewünschten Approximationsintervalls I_x das etwas grössere Intervall $I_r := [-\gamma - \triangle(r), \gamma + \triangle(r)]$ betrachtet werden. Für den Fall $\gamma := \frac{1}{8}$ kann eine obere Schranke für den absoluten Fehler $\triangle(r)$ mit Hilfe des Programms `ffatan_r.p` berechnet werden:

```
Absolute Fehlerschranke der Argumentreduktion:  2.887070574022266E-016

Das reduzierte Argument liegt im Bereich:
[-1.250000000000003E-001, 1.250000000000003E-001]
```

Für $\gamma := \frac{1}{8}$ gilt also $I_r \subseteq [-0.1250000000000003, 0.1250000000000003]$.

Zunächst wird der Approximationsfehler für die folgende Approximation

$$\frac{\frac{\arctan(r)}{x} - 1}{x^2} \approx p_N(r^2)$$

mit Hilfe des Programms `atan.p` bestimmt:

```
Approximationsfehler einer polynomialen Approximation fuer
(arctan(x)/x - 1) / (x^2) =  -1/3 + 1/5 x^2 - 1/7 x^4 + 1/9 x^6...
...
Approximationsintervall = -1.250000000100000E-001 ..  1.250000000100000E-001
Funktionsauswertung ueber Gesamtintervall ergibt
[              -3.37E-001,              -3.30E-001 ]
...
Schranke fuer den Betrag des maximalen   a b s o l u t e n   Fehlers:
    4.7168481E-016
```

Es gilt somit

$$\left| \frac{\frac{\arctan(r)}{x} - 1}{x^2} - p_N(r^2) \right| \leq 4.7169 \cdot 10^{-16}$$

und daraus folgt

$$\left| \arctan(r) - r \cdot (1 + x^2 \cdot p_N(r^2)) \right| \leq 4.7169 \cdot 10^{-16} \cdot |r|^3 \leq 0.0093 \cdot 10^{-18}.$$

7.1.4 Algorithmus q_atan (Punktfunktion)

Der Gesamtalgorithmus `q_atan` zur Berechnung von $\arctan(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| \leq$`q_atnt`$= 1.807032\ldots \cdot 10^{-8} \Longrightarrow$ res $:= x$

II. Initialisierung und Argumentreduktion

a) Falls $(x < 0)$: $y := -x$, neg:=`true`
sonst: $y := x$, neg:=`false`

b) Falls $(y < 8)$: vzi:= 1, $y_m := 0$
sonst: vzi:= -1, $y_m := \frac{\pi}{2}|_{\text{IEEE}}$, $y := 1 \boxplus y$

c) Bestimme $i := \max_{k=1,2,\ldots,7}\{k | b_k \leq y\}$

d) Berechne $y := (y \boxminus c_i) \boxplus (1 \boxplus y \boxdot c_i)$

III. Approximation: $y + y \cdot (\sum_{i=2}^{7} d_i \cdot y^2)$, d.h. auf dem Rechner

a) $y_{\text{sq}} := y \boxdot y$

b) res $:= y \boxdot ((((((d_7 \boxdot y_{\text{sq}} \boxplus d_6) \boxdot y_{\text{sq}} \boxplus d_5) \boxdot y_{\text{sq}} \boxplus d_4) \boxdot y_{\text{sq}} \boxplus d_3)$
$\boxdot y_{\text{sq}} \boxplus d_2) \boxdot y_{\text{sq}} \boxplus d_1) \boxdot y_{\text{sq}}) \boxplus y$

IV. Ergebnisanpassung

a) res $:=$ res $\boxplus a_i$

b) res $:=$ vzi $\cdot$ res $\boxplus y_m$

c) Falls (neg=`true`): res $:= -$res

Nun gilt `q_atan` $:=$ res $\approx \arctan(x)$.
Die Fehlerabschätzung für den eben beschriebene Algorithmus wird in folgenden Abschnitt durchgeführt.

7.1.5 Fehlerabschätzung

Für den oben angegebenen Algorithmus kann nach Umsetzung in ein PASCAL-XSC Programm mit einer sinnvollen Unterteilung des zu untersuchenden Bereiches $[1.807031999999999E-008, 1.797693134862316E+308]$ in Teilintervalle eine relative Fehlerschranke hergeleitet werden.

Ergebnisprotokoll des Programms `ffatan_a.p`:

```
Test mit 1000 Zufallszahlen:
Minimale Groessenordnung der Fehlerschranke :  6.104482422310954E-016

Test mit x:=0.125
```

```
Minimale Groessenordnung der Fehlerschranke :  1.798440248823969E-015

[ 1.807031999999999E-008, 1.000000000000000E-007]  1.226454219578578E-015
[ 9.999999999999999E-008, 1.000000000000000E-006]  1.349448571975024E-015
[ 9.999999999999999E-007, 1.000000000000001E-005]  1.349448571977040E-015
[ 1.000000000000000E-005, 1.000000000000001E-004]  1.349448572178637E-015
[ 1.000000000000000E-004, 1.000000000000001E-003]  1.349448592338349E-015
[ 1.000000000000000E-003, 1.000000000000001E-002]  1.349450608310623E-015
[ 1.000000000000000E-002, 1.000000000000001E-001]  1.349652215947433E-015
[ 1.000000000000000E-001, 1.250000000000000E-001]  1.254093890964245E-015
[ 1.250000000000000E-001, 3.913934426229509E-001]  1.798499017044059E-015
[ 3.913934426229508E-001, 7.165920254261774E-001]  1.157386217181264E-015
[ 7.165920254261773E-001, 1.186491862010193E+000]  9.197982221441163E-016
[ 1.186491862010192E+000, 2.061721166266557E+000]  8.497853987936468E-016
[ 2.061721166266556E+000, 4.860928659711213E+000]  1.005001402033394E-015
[ 4.860928659711212E+000, 8.000000000000000E+000]  5.381909041457355E-016
[ 8.000000000000000E+000, 1.797693134862316E+308]  4.777603396332967E-016

Relative Fehlerschranke der ffarctan-Funktion:  1.798499017044059E-015

Definitionsbereich fuer x: [ 1.807031999999999E-008, 1.797693134862316E+308]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 1.798499017044059E-015;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999978E-001 */
double q_...m  = 9007199254740972.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000003E+000 */
double q_...p  = 4503599627370508.0 / 4503599627370496.0;
```

Für $|x| \leq 1.807032\ldots E-008$ wird die Approximation $\arctan(x) \approx x$ verwendet. Mit einfacher Handrechnung erhält man den für Ergebnisse im Bereich der normalisierten Zahlen gültigen relativen Approximationsfehler (siehe [28], Seite 50):

$$\left|\frac{\arctan x - x}{\arctan x}\right| \leq \frac{x^2}{3} \cdot \frac{1}{1-x^2} \cdot \frac{1}{1-(1/3)x^2} \leq 1.0885e-16 := \varepsilon(\mathrm{app}_N)$$

Im Bereich der denormalisierten Zahlen gilt der absolute Fehler:

$$|\arctan x - x| \leq \frac{x^2}{3} \cdot \frac{1}{1-x^2} \leq 1.66e-616 \leq \texttt{dMinReal} := \triangle(\mathrm{app}_D)$$

Gesamtfehlerschranke:
Relative Fehlerschranke für `q_atan`$(x) \in S_N$: $\varepsilon(\texttt{q\_atan}) \leq 1.7985E-015 = 16.20 \cdot \varepsilon^*$
Absolute Fehlerschranke für `q_atan`$(x) \in S_D$: $\triangle(\texttt{q\_atan}) \leq 4.9497E-324$

7.1.6 Algorithmus `j_atan` (Intervallfunktion)

Der Gesamtalgorithmus `j_atan` zur Berechnung von $\arctan(X)$ für Intervallargumente $X := [x.\mathrm{INF}, x.\mathrm{SUP}]$ stellt sich wie folgt dar:

Benötigte Konstanten:

$$\begin{aligned}
\texttt{q_atnt} &:= 1.807032\ldots\cdot 10^{-8}\\
\texttt{q_ctnm} &:= (1 \bigtriangledown \varepsilon(\texttt{q_atan})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})\\
\texttt{q_ctnp} &:= (1 \bigtriangleup \varepsilon(\texttt{q_atan})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})
\end{aligned}$$

I. X ist Punktintervall, d.h. x.INF = x.SUP

a) x.INF $\leq -$q_atnt:
$y :=$ q_atan(x.INF)
res.INF:= $y \boxdot$ q_ctnp
res.SUP:= $y \boxdot$ q_ctnm
Falls (res.INF$<$ x.INF) dann: res.INF:= x.INF

b) $-$q_atnt $< x$.INF < 0:
res.INF:= x.INF
res.SUP:= succ(x.INF)

c) $0 \leq x$.INF $<$ q_atnt:
Falls (x.INF $= 0$) dann: res.INF:= 0
sonst: res.INF:= pred(x.INF)
res.SUP:= x.INF

d) q_atnt $\leq x$.INF:
$y :=$ q_atan(x.INF)
res.INF:= $y \boxdot$ q_ctnm
res.SUP:= $y \boxdot$ q_ctnp
Falls (res.SUP$<$ x.INF) dann: res.SUP:= x.INF

II. X ist echtes Intervall, d.h. x.INF $\neq x$.SUP

a) Berechnung der Unterschranke res.INF:

i. x.INF $\leq -$q_atnt:
res.INF:= q_atan(x.INF) $\boxdot$ q_ctnp
Falls (res.INF$<$ x.INF) dann: res.INF:= x.INF

ii. $-$q_atnt $< x$.INF ≤ 0:
res.INF:= x.INF

iii. $0 < x$.INF $<$ q_atnt:
res.INF:= pred(x.INF)

iv. q_atnt $\leq x$.INF:
res.INF:= q_atan(x.INF) $\boxdot$ q_ctnm

b) Berechnung der Oberschranke res.SUP:

i. x.SUP $\leq -$q_atnt:
res.SUP:= q_atan(x.SUP) $\boxdot$ q_ctnm

ii. $-$q_atnt $< x$.SUP < 0:
res.SUP:= succ(x.SUP)

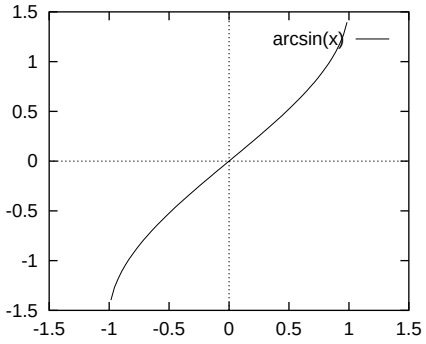
iii. $0 \leq x.\text{SUP} <$ `q_atnt`:
res.SUP:= x.SUP

iv. `q_atnt` $\leq x$.SUP:
res.SUP:= `q_atan`(x.SUP) $\boxdot$ `q_ctnp`
Falls (res.SUP$> x$.SUP) dann: res.SUP:= x.SUP

Nun gilt `j_atan` := res $\supseteq \arctan(X)$ mit res := [res.INF, res.SUP].

7.2 Arcussinusfunktion: arcsin(x)

Arcussinusfunktion	
Math. Schreibweise	$\arcsin x$
Definitionsbereich	$[-1, 1]$
1. Ableitung	$\frac{1}{\sqrt{1-x^2}}$
Nullstellen	$x_0 = 0$
Minima	$x_T = -1$
Symmetrie	Punktsymmetrie zu $(0, 0)$
Monotonie	monoton steigend



7.2.1 Idee

Die Berechnung wird mit Hilfe der Formel

$$\arcsin(x) := \arctan\left(\frac{x}{\sqrt{(1+x)(1-x)}}\right)$$

auf die Berechnung es Arkustangens zurückgeführt. Für sehr kleine Argumente $x <$ `q_atnt` wird die Approximation

$$\arcsin(x) \approx x$$

verwendet. Die Eingabeargumente $x = \pm 1$ werden als Sonderfall behandelt, für $x = +1$ wird $\arcsin := \frac{\pi}{2}\Big|_{\text{IEEE}}$, für $x = -1$ wird $\arcsin := -\frac{\pi}{2}\Big|_{\text{IEEE}}$ gesetzt.

7.2.2 Algorithmus `q_asin` (Punktfunktion)

Der Gesamtalgorithmus `q_asin` zur Berechnung von $\arcsin(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x < -1.0$ oder $x > 1.0$ (Test des Definitionsbereiches) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x = -1 \Longrightarrow$ res $:= -\frac{\pi}{2}\Big|_{\text{IEEE}}$

- $x = 1 \Longrightarrow \text{res} := \frac{\pi}{2}\Big|_{\text{IEEE}}$
- $|x| \leq$`q_atnt`$= 1.807032\ldots \cdot 10^{-8} \Longrightarrow \text{res} := x$

II. Berechnung mit Hilfe des Arkustangens
res := `q_atan`$(x \boxplus$ `q_sqrt`$((1.0 \boxplus x) \boxdot (1.0 \boxminus x)))$

Nun gilt `q_asin` := res $\approx \arcsin(x)$.
Hinweis: In der ANSI-C Implementierung wird aus Laufzeitgründen nicht die Funktion `q_sqrt`, sondern die Funktion `sqrt` verwendet. Ebenso wird anstelle der Funktion `q_atan` die Funktion `q_atn1` aufgerufen, in dieser Funktion werden im Unterschied zu `q_atan` bestimmte Spezialfälle nicht abgeprüft.

Die Fehlerabschätzung für den eben beschriebene Algorithmus wird in folgenden Abschnitt durchgeführt.

7.2.3 Fehlerabschätzung

Die Fehlerabschätzung für Argumente $x \in [\text{succ}(-1.0), -$`q_atnt`$] \cup [$`q_atnt`$, \text{pred}(1.0)]$ wird mit Hilfe des PASCAL-XSC Programms `ffasin.p` durchgeführt, das Ergebnisprotokoll ist im folgenden angegeben:

```
Test mit 1000 Zufallszahlen:
Minimale Groessenordnung der Fehlerschranke :  2.575656450000008E-015

[-9.999999999999999E-001,-9.999999998999999E-001]  2.201640632253669E-015
[-9.999999999000000E-001,-9.999000000000000E-001]  2.575072504195729E-015
[-9.999000000000001E-001,-9.000000000000000E-001]  2.571973678196994E-015
[-9.000000000000001E-001,-5.000000000000000E-001]  2.473982614671442E-015
[-5.000000000000000E-001,-1.000000000000000E-001]  2.585009878638179E-015
[-1.000000000000001E-001,-1.000000000000000E-002]  2.591266638712048E-015
[-1.000000000000001E-002,-1.000000000000000E-003]  2.591130768054251E-015
[-1.000000000000001E-003,-1.000000000000000E-004]  2.591112474187027E-015
[-1.000000000000001E-004,-1.000000000000000E-005]  2.591110597724613E-015
[-1.000000000000001E-005,-9.999999999999999E-007]  2.591110409608210E-015
[-1.000000000000000E-006,-9.999999999999999E-008]  2.591110390789878E-015
[-1.000000000000000E-007,-1.807031999999999E-008]  2.583441691724714E-015
[ 1.807031999999999E-008, 1.000000000000000E-007]  2.587334312588489E-015
[ 9.999999999999999E-008, 1.000000000000000E-006]  2.587246905590743E-015
[ 9.999999999999999E-007, 1.000000000000001E-005]  2.587246919681883E-015
[ 1.000000000000000E-005, 1.000000000000001E-004]  2.590144713415197E-015
[ 1.000000000000000E-004, 1.000000000000001E-003]  2.590146471621126E-015
[ 1.000000000000000E-003, 1.000000000000001E-002]  2.590163583485816E-015
[ 1.000000000000000E-002, 1.000000000000001E-001]  2.590287690402380E-015
[ 1.000000000000000E-001, 5.000000000000000E-001]  2.585009878638189E-015
[ 5.000000000000000E-001, 9.000000000000001E-001]  2.473982614671445E-015
[ 9.000000000000000E-001, 9.999000000000001E-001]  2.173435210172913E-015
[ 9.999000000000000E-001, 9.999999990000001E-001]  2.423557170868157E-015
[ 9.999999990000000E-001, 9.999999999000000E-001]  1.798524415263348E-015
[ 9.999999998999999E-001, 9.999999999999999E-001]  2.201569143709489E-015

Relative Fehlerschranke der ffarcsin-Funktion:  2.591266638712048E-015
```

```
Definitionsbereich fuer x: [-9.999999999999999E-001,-1.807031999999999E-008]
                       und [ 1.807031999999999E-008, 9.999999999999999E-001]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 2.591266638712048E-015;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999970E-001 */
double q_...m  = 9007199254740965.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000003E+000 */
double q_...p  = 4503599627370511.0 / 4503599627370496.0;
```

Für $|x| \leq 1.807032\ldots E-008$ wird die Approximation $\arcsin(x) \approx x$ verwendet. Mit einfacher Handrechnung erhält man den für Ergebnisse im Bereich der normalisierten Zahlen gültigen relativen Approximationsfehler (siehe [13], Seite 78):

$$\left|\frac{\arcsin x - x}{\arcsin x}\right| \leq \frac{x^1}{3} \cdot \frac{1}{2} \cdot \frac{x^2}{1-x^2} \leq 5.4423e-17 := \varepsilon(\mathrm{app}_N)$$

Im Bereich der denormalisierten Zahlen gilt der absolute Fehler:

$$|\arcsin x - x| \leq \frac{x^1}{3} \cdot \frac{1}{2} \cdot \frac{|x|^3}{1-x^2} \leq \texttt{dMinReal} := \triangle(\mathrm{app}_D)$$

Bei den beiden Spezialfällen $x = -1.0$ und $x = 1.0$ ist der relative Fehler jeweils kleiner oder gleich `Eps53`.

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_asin}(x) \in S_N$: $\varepsilon(\texttt{q\_asin}) \leq 2.5913E-015 = 23.34 \cdot \varepsilon^*$
Absolute Fehlerschranke für $\texttt{q\_asin}(x) \in S_D$: $\triangle(\texttt{q\_asin}) \leq 4.9497E-324$

7.2.4 Algorithmus `j_asin` (Intervallfunktion)

Der Gesamtalgorithmus `j_asin` zur Berechnung von $\arcsin(X)$ für Intervallargumente $X := [x.\mathrm{INF}, x.\mathrm{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\begin{aligned}
\texttt{q_atnt} &:= 1.807032\ldots \cdot 10^{-8} \\
\texttt{q_csnm} &:= (1 \bigtriangledown \varepsilon(\texttt{q_asin})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer}) \\
\texttt{q_csnp} &:= (1 \bigtriangleup \varepsilon(\texttt{q_asin})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})
\end{aligned}$$

I. X ist Punktintervall, d.h. $x.\mathrm{INF} = x.\mathrm{SUP}$

a) $x.\mathrm{INF} \leq -\texttt{q\_atnt}$:
$y := \texttt{q\_asin}(x.\mathrm{INF})$
res.INF:= $y \boxdot \texttt{q\_csnp}$
res.SUP:= $y \boxdot \texttt{q\_csnm}$
Falls (res.SUP> x.INF) dann: res.SUP:= x.INF

b) $-\texttt{q\_atnt} < x.\mathrm{INF} < 0$:
res.INF:= pred(x.INF)
res.SUP:= x.INF

c) $0 \leq x.\mathrm{INF} < \texttt{q\_atnt}$:
res.INF:= x.INF Falls (x.INF = 0) dann: res.SUP:= 0
sonst: res.SUP:= succ(x.INF)

d) $\texttt{q\_atnt} \leq x.\mathrm{INF}$:
$y :=$ `q_asin`(x.INF)
res.INF:= $y \boxdot$ `q_ctnm`
res.SUP:= $y \boxdot$ `q_ctnp`
Falls (res.INF$<$ x.INF) dann: res.INF:= x.INF

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\mathrm{INF} \leq -\texttt{q\_atnt}$:
res.INF:= `q_asin`(x.INF) $\boxdot$ `q_csnp`
Falls (res.INF$<$ x.INF) dann: res.INF:= x.INF

ii. $-\texttt{q\_atnt} < x.\mathrm{INF} < 0$:
res.INF:= pred(x.INF)

iii. $0 \leq x.\mathrm{INF} < \texttt{q\_atnt}$:
res.INF:= x.INF

iv. $\texttt{q\_atnt} \leq x.\mathrm{INF}$:
res.INF:= `q_asin`(x.INF) $\boxdot$ `q_csnm`
Falls (res.INF$<$ x.INF) dann: res.INF:= x.INF

b) Berechnung der Oberschranke res.SUP:

i. $x.\mathrm{SUP} \leq -\texttt{q\_atnt}$:
res.SUP:= `q_asin`(x.SUP) $\boxdot$ `q_csnm`

ii. $-\texttt{q\_atnt} < x.\mathrm{SUP} \leq 0$:
res.SUP:= x.SUP

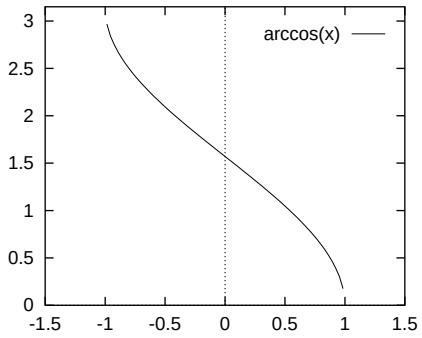
iii. $0 < x.\mathrm{SUP} < \texttt{q\_atnt}$:
res.SUP:= succ(x.SUP)

iv. $\texttt{q\_atnt} \leq x.\mathrm{SUP}$:
res.SUP:= `q_asin`(x.SUP) $\boxdot$ `q_csnp`

Nun gilt `j_asin` := res $\supseteq \arcsin(X)$ mit res := [res.INF, res.SUP].

7.3 Arcuscosinusfunktion: arccos(x)

Arcuscosinusfunktion	
Math. Schreibweise	$\arccos x$
Definitionsbereich	$[-1, 1]$
1. Ableitung	$-\frac{1}{\sqrt{1-x^2}}$
Nullstellen	$x_0 = 1$
Minima	$x_T = 1$
Monotonie	monoton fallend



7.3.1 Idee

Zuerst wird überprüft, ob das Funktionsargument x innerhalb des Definitionsbereiches $D := [-1, 1]$ liegt. Für x mit $|x| > 1$ wird eine Fehlermeldung ausgegeben.
Für $-1 \leq x \leq -10^{-17}$ erfolgt die Berechnung über

$$\arccos(x) = \pi\Big|_{\text{IEEE}} + \arctan\left(\frac{\sqrt{(1+x)(1-x)}}{x}\right).$$

Für $10^{-17} \leq x \leq 1$ wird die Formel

$$\arccos(x) = \arctan\left(\frac{\sqrt{(1+x)(1-x)}}{x}\right)$$

ausgewertet.
Für $|x| < 10^{-17}$ wird die Approximation

$$\arccos(x) = \frac{\pi}{2}\Big|_{\text{IEEE}}$$

verwendet.

7.3.2 Algorithmus `q_acos` (Punktfunktion)

Der Gesamtalgorithmus `q_acos` zur Berechnung von $\arccos(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x < -1.0$ oder $x > 1.0$ (Test des Definitionsbereiches) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x|$ <1e-17$\Longrightarrow$ res := $\frac{\pi}{2}\Big|_{\text{IEEE}}$

II. Berechnung mit Hilfe des Arkustangens

a) $x > 1e-17$:
res := `q_atan`(`q_sqrt`$((1.0 \boxplus x) \boxdot (1.0 \boxminus x)) \boxslash x$)

b) $x < -1e-17$:
res := $\pi\big|_{\text{IEEE}} \boxplus$ `q_atan`(`q_sqrt`$((1.0 \boxplus x) \boxdot (1.0 \boxminus x)) \boxslash x$)

Nun gilt `q_acos` := res $\approx \arccos(x)$.

7.3.3 Fehlerabschätzung

Die Fehlerabschätzung für Argumente $x \in [\text{succ}(-1.0), -1e-17] \cup [1e-17, \text{pred}(1.0)]$ wird mit Hilfe des PASCAL-XSC Programms `ffacos.p` durchgeführt, das Ergebnisprotokoll ist im folgenden angegeben:

```
Test mit 1000 Zufallszahlen:
Minimale Groessenordnung der Fehlerschranke :  2.475551026406628E-015

Test mit x:=pred(1.0)
Minimale Groessenordnung der Fehlerschranke :  2.575656450000005E-015

[-9.999999999999999E-001,-9.999900000000000E-001]  1.282114837290963E-015
[-9.999900000000001E-001,-9.000000000000000E-001]  9.580552724543393E-016
[-9.000000000000001E-001,-5.000000000000000E-001]  1.500503938154643E-015
[-5.000000000000000E-001,-1.000000000000000E-004]  2.372926040646923E-015
[-1.000000000000001E-004,-1.000000000000000E-008]  2.247547425868874E-015
[-1.000000000000001E-008,-9.999999999999999E-013]  2.242589995783036E-015
[-1.000000000000000E-012,-1.000000000000000E-017]  2.242589500494861E-015
[ 1.000000000000000E-017, 1.000000000000000E-012]  1.798500000494959E-015
[ 9.999999999999999E-013, 1.000000000000001E-008]  1.798500496788968E-015
[ 1.000000000000000E-008, 1.000000000000001E-004]  1.803468016941919E-015
[ 1.000000000000000E-004, 5.000000000000000E-001]  2.139073788707985E-015
[ 5.000000000000000E-001, 9.000000000000001E-001]  2.477636204606855E-015
[ 9.000000000000000E-001, 9.900000000000000E-001]  2.571918939192369E-015
[ 9.899999999999999E-001, 9.990000000000000E-001]  2.581165609583567E-015
[ 9.989999999999999E-001, 9.999000000000001E-001]  2.582092410855208E-015
[ 9.999000000000000E-001, 9.999900000000001E-001]  2.582185128574106E-015
[ 9.999900000000000E-001, 9.999990000000000E-001]  2.582194269949106E-015
[ 9.999989999999999E-001, 9.999999000000001E-001]  2.582194231640566E-015
[ 9.999999000000000E-001, 9.999999900000000E-001]  2.582194308189621E-015
[ 9.999999899999999E-001, 9.999999990000001E-001]  2.582323329940879E-015
[ 9.999999990000000E-001, 9.999999999000000E-001]  2.582967036195150E-015
[ 9.999999998999999E-001, 9.999999999900000E-001]  2.590450110913059E-015
[ 9.999999999899999E-001, 9.999999999900000E-001]  2.575656449989645E-015
...  (Test mit Punktintervallen)
[ 9.999999999999998E-001, 9.999999999999999E-001]  2.575656450000005E-015

Relative Fehlerschranke der ffarccos-Funktion:  2.590450110913059E-015

Definitionsbereich fuer x: [-9.999999999999999E-001,-1.000000000000000E-017]
                       und [ 1.000000000000000E-017, 9.999999999900000E-001]

------------------------------------------------------------
```

```
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 2.590450110913058E-015;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999970E-001 */
double q_...m  = 9007199254740965.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000003E+000 */
double q_...p  = 4503599627370511.0 / 4503599627370496.0;
```

Gesamtfehlerschranke:
Relative Fehlerschranke für `q_acos`$(x) \in S_N$: $\varepsilon(\texttt{q\_acos}) \leq 2.5905E-015 = \cdot 23.34\varepsilon^*$

7.3.4 Algorithmus `j_acos` (Intervallfunktion)

Der Gesamtalgorithmus `j_acos` zur Berechnung von $\arccos(X)$ für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_ccsm} := (1 \bigtriangledown \varepsilon(\texttt{q\_acos})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_ccsp} := (1 \bigtriangleup \varepsilon(\texttt{q\_acos})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

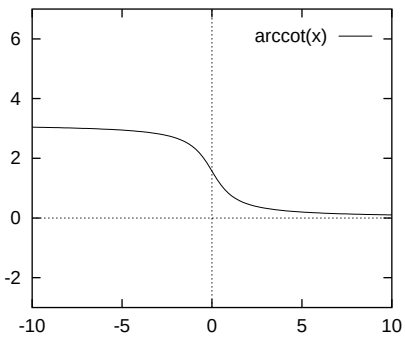
I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$
$y :=$ `q_acos`$(x.\text{INF})$
res.INF:= $y \boxdot$ `q_ccsm`
res.SUP:= $y \boxdot$ `q_ccsp`

II. X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$
res.INF:= `q_acos`$(x.\text{SUP}) \boxdot$ `q_ccsm`
res.SUP:= `q_acos`$(x.\text{INF}) \boxdot$ `q_ccsp`

Nun gilt `j_acos` := res $\supseteq \arccos(X)$ mit res := [res.INF, res.SUP].

7.4 Arcuscotangensfunktion: arccot(x)

Arcuscotangensfunktion	
Math. Schreibweise	$\operatorname{arccot} x$
Definitionsbereich	$\mathbb{R}$
Potenzreihe für $\lvert x\rvert < 1$	$\frac{\Pi}{2} - \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{2k+1}$ $= \frac{\Pi}{2} - (x - \frac{x^3}{3} + \frac{x^5}{5} - + \ldots)$
1. Ableitung	$-\frac{1}{1+x^2}$
Nullstellen	keine
Symmetrie	Punktsymmetrie zu $(0,0)$
Monotonie	monoton fallend



7.4.1 Idee

Die Berechnung wird für negative Argumente $x \leq -10^{-17}$ mit Hilfe der Formel

$$\text{arccot} := \pi\big|_{\text{IEEE}} + \arctan(\frac{1.0}{x}),$$

für positive Argumente $10^{-17} \leq x < 10^{10}$ mit der Formel

$$\text{arccot} := \arctan(\frac{1.0}{x})$$

auf die Berechung des Arkustangens zurückgeführt. Für große Argumente $10^{10} \leq x$ erfolgt die Approximation mit

$$\text{arccot} := \frac{1.0}{x},$$

für $|x| < 10^{-17}$ wird

$$\text{arccot}\,(x) := \frac{\pi}{2}\Big|_{\text{IEEE}}$$

verwendet.

7.4.2 Algorithmus `q_acot` (Punktfunktion)

Der Gesamtalgorithmus `q_acot` zur Berechnung von $\text{arccot}\,(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x|$ <1e-17$\Longrightarrow$ res := $\frac{\pi}{2}\big|_{\text{IEEE}}$

II. Approximation der Funktion

a) $x \leq -1e-17$:
res := $\pi\big|_{\text{IEEE}} \boxplus$ `q_atan`$(1.0 \boxplus x)$

b) $1e-17 \leq x < 1e10$:
res := `q_atan`$(1.0 \boxplus x)$

c) $1e10 \leq x$:
res := $1.0 \boxplus x$

Nun gilt `q_acot` := res $\approx \text{arccot}\,(x)$.

7.4.3 Fehlerabschätzung

Die Fehlerabschätzung für Argumente $x \in [-\texttt{maxreal}, -1e-17] \cup [1e-17, 1e10]$ wird mit Hilfe des PASCAL-XSC Programms `ffacot.p` durchgeführt, das Ergebnisprotokoll ist im folgenden angegeben:

```
Test mit 1000 Zufallszahlen:
Minimale Groessenordnung der Fehlerschranke :  2.242589500000005E-015

[-1.797693134862316E+308,-1.000000000000000E+000]  1.137828164246288E-015
[-1.000000000000000E+000,-1.000000000000000E-004]  2.242554849088074E-015
[-1.000000000000001E-004,-1.000000000000000E-008]  2.242589660833739E-015
[-1.000000000000001E-008,-9.999999999999999E-013]  2.242589500016087E-015
[-1.000000000000000E-012,-1.000000000000000E-017]  2.242589500000019E-015
[ 1.000000000000000E-017, 1.000000000000000E-012]  1.798500000000144E-015
[ 9.999999999999999E-013, 1.000000000000001E-008]  1.798500001426446E-015
[ 1.000000000000000E-008, 1.000000000000001E-004]  1.798514265345811E-015
[ 1.000000000000000E-004, 1.000000000000001E-001]  1.813579929327843E-015
[ 1.000000000000000E-001, 1.000000000000000E+001]  2.021066184397226E-015
[ 1.000000000000000E+001, 1.000000000000000E+002]  2.200019387426096E-015
[ 1.000000000000000E+002, 1.000000000000000E+003]  2.202369798627842E-015
[ 1.000000000000000E+003, 1.000000000000000E+004]  2.202393483738031E-015
[ 1.000000000000000E+004, 1.000000000000000E+005]  2.202393720607381E-015
[ 1.000000000000000E+005, 1.000000000000000E+006]  2.202393722976076E-015
[ 1.000000000000000E+006, 1.000000000000000E+007]  2.202393722999764E-015
[ 1.000000000000000E+007, 1.000000000000000E+008]  2.202393723000000E-015
[ 1.000000000000000E+008, 1.000000000000000E+009]  2.202393723000003E-015
[ 1.000000000000000E+009, 1.000000000000000E+010]  2.202393723000003E-015

Relative Fehlerschranke der ffarccot-Funktion:  2.242589660833739E-015

Definitionsbereich fuer x: [-1.797693134862316E+308,-1.000000000000000E-017]
                       und [ 1.000000000000000E-017, 1.000000000000000E+010]

-------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 2.242589660833738E-015;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999973E-001 */
double q_...m  = 9007199254740968.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000003E+000 */
double q_...p  = 4503599627370510.0 / 4503599627370496.0;
```

Für $|x| \leq 1e-17$ wird die Approximation $\operatorname{arccot}(x) \approx \frac{\pi}{2}\big|_{\text{IEEE}}$ verwendet. Mit einfacher Handrechnung erhält man den für Ergebnisse im Bereich der normalisierten Zahlen gültigen relativen Approximationsfehler (siehe auch [28], Seite 56):

$$\left|\frac{\operatorname{arccot} x - \frac{\pi}{2}}{\operatorname{arccot} x}\right| = \left|\frac{\arctan x}{\operatorname{arccot} x}\right| \leq |x| \cdot \frac{1}{1-(1/3)x^2} \leq 1.0001e-17$$

Für $1e10 \leq x$ wird die Approximation $1/x$ verwendet:

$$\left|\frac{\operatorname{arccot} x - 1/x}{\operatorname{arccot} x}\right| \leq \frac{(1/x)^2}{3} \cdot \frac{1}{1-(1/x)^2} \cdot \frac{1}{1-(1/3)\cdot(1/x)^2} \leq 5.4423e-17 := \varepsilon(\text{app}_N)$$

Zusätzlich muß hier noch der Rundungsfehler bei Ausführung der Division $1 \boxplus x$ berücksichtigt werden. Im Bereich der denormalisierten Zahlen gilt der absolute Fehler:

$$\left|\frac{\operatorname{arccot} x - 1/x}{\operatorname{arccot} x}\right| \leq \frac{(1/x)^2}{3} \cdot \frac{1}{1-(1/x)^2} \cdot \frac{1}{1-(1/3)\cdot(1/x)^2} \leq \texttt{dMinReal} := \triangle(\mathrm{app}_D)$$

Zusätzlich muß in den beiden letzten Fällenhier der Rundungsfehler bei Ausführung der Division $1 \boxplus x$ berücksichtigt werden.

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_acot}(x) \in S_N$: $\varepsilon(\texttt{q\_acot}) \leq 2.2426E-015 = 20.20 \cdot \varepsilon^*$
Absolute Fehlerschranke für $\texttt{q\_acot}(x) \in S_D$: $\triangle(\texttt{q\_acot}) \leq 4.9497E-324$

7.4.4 Algorithmus `j_acot` (Intervallfunktion)

Der Gesamtalgorithmus `j_acot` zur Berechnung von arccot (X) für Intervallargumente $X := [x.\mathrm{INF}, x.\mathrm{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\begin{aligned} \texttt{q\_cctm} &:= (1 \bigtriangledown \varepsilon(\texttt{q\_acot})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer}) \\ \texttt{q\_cctp} &:= (1 \bigtriangleup \varepsilon(\texttt{q\_acot})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer}) \end{aligned}$$

I. X ist Punktintervall, d.h. $x.\mathrm{INF} = x.\mathrm{SUP}$
$y := \texttt{q\_acot}(x.\mathrm{INF})$
res.INF:= $y \boxdot \texttt{q\_cctm}$
res.SUP:= $y \boxdot \texttt{q\_cctp}$

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$
res.INF:= $\texttt{q\_acot}(x.\mathrm{SUP}) \boxdot \texttt{q\_cctm}$
res.SUP:= $\texttt{q\_acot}(x.\mathrm{INF}) \boxdot \texttt{q\_cctp}$

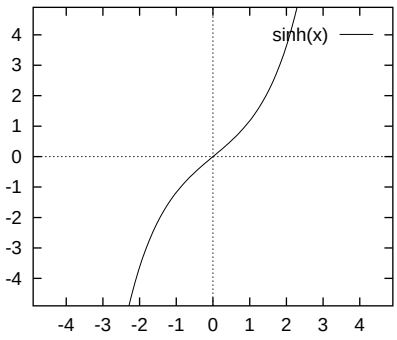
Nun gilt `j_acot` := res $\supseteq$ arccot (X) mit res := [res.INF, res.SUP].

Kapitel 8

Hyperbolische Funktionen

8.1 Sinus Hyperbolicus - Funktion: sinh(x)

hyperbolische Sinusfunktion	
Math. Schreibweise	$\sinh x = \frac{e^x - e^{-x}}{2}$
Definitionsbereich	$\mathbb{R}$
Potenzreihe für $\lvert x\rvert < \infty$	$\sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}$ $= x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \ldots$
1. Ableitung	$\cosh(x)$
Nullstellen	$x_0 = 0$
Symmetrie	Punktsymmetrie zu $(0, 0)$
Monotonie	monoton steigend



8.1.1 Idee

Die sinh-Funktion wird mit Hilfe der folgenden Formeln auf die exp- bzw. expm1-Funktion zurückgeführt.
Für $|x| \geq$ tpkt1:

$$\sinh(x) = \text{sign}(x) \cdot 0.5 \cdot (e^{|x|} - \frac{1}{e^{|x|}})$$

Für tpkt2$\leq |x| <$tpkt1:

$$\sinh(x) = \text{sign}(x) \cdot 0.5 \cdot ((e^{|x|} - 1) + \frac{(e^{|x|} - 1)}{(e^{|x|} - 1) + 1})$$

Für $|x| <$tpkt2 wird die Approximation

$$\sinh(x) = x$$

verwendet.

Der Teilpunkt tpkt2 wird so gewählt, daß für den relativen Fehler $|\frac{\sinh(x)-x}{\sinh(x)}| \leq \varepsilon^*$ gilt. (Hinweis: Die Punktfunktion liefert dann im entsprechenden Bereich ein maximalgenaues Ergebnis)

Der Teilpunkt tpkt1 wird so gewählt, daß der relative Fehler in den Teilbereichen links und rechts von tpkt1 ungefähr gleichgroß ist.

Konkret werden die folgenden beiden Werte verwendet: tpkt1= 0.662 und tpkt2= $2.5783798e-8$.

8.1.2 Algorithmus q_sinh (Punktfunktion)

Der Gesamtalgorithmus `q_sinh` zur Berechnung von $\sinh(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x >$`q_ex2a`$= 709.78... \Longrightarrow$ Fehlermeldung: Overflow (Überlauf)
- $|x| < 2.5783798e-8 \Longrightarrow \text{res} := x$

II. Berechnung mit Hilfe der Exponentialfunktionen

a) $|x| \geq 0.662$:
$h :=$ `q_exp`$(|x|)$
$\text{res} := \text{sign}(x) \cdot 0.5 \cdot (h \boxminus 1.0 \boxslash h)$

b) $|x| < 0.662$:
$h :=$ `q_expm1`$(|x|)$
$\text{res} := \text{sign}(x) \cdot 0.5 \cdot (h \boxplus h \boxslash (h \boxplus 1.0))$

Nun gilt `q_sinh` $:= \text{res} \approx \sinh(x)$.

8.1.3 Fehlerabschätzung

Es wird nur der Fall $x \geq 0$ betrachtet, da die Funktion punktsymmetrisch zum Ursprung ist (Vorzeichenwechsel ist fehlerfrei möglich). Die Fehlerabschätzung wird mit Hilfe des Programms `ffsinh.p`, durchgeführt, es wird das folgende Ergebnisprotokoll ausgegeben:

```
Test mit 1000 Zufallszahlen:
Minimale Groessenordnung der Fehlerschranke :  7.040422463697968E-016

Test mit x:=0.662
Minimale Groessenordnung der Fehlerschranke :  7.093087374122772E-016

[ 2.578379799999999E-008, 1.000000000000000E-007]  7.037543112958781E-016
[ 9.999999999999999E-008, 1.000000000000000E-006]  7.054594518766065E-016
[ 9.999999999999999E-007, 1.000000000000001E-005]  7.054594697955710E-016
[ 1.000000000000000E-005, 1.000000000000001E-004]  7.054596632433388E-016
[ 1.000000000000000E-004, 1.000000000000001E-003]  7.054615972928992E-016
[ 1.000000000000000E-003, 1.000000000000001E-002]  7.054808215656063E-016
[ 1.000000000000000E-002, 1.000000000000001E-001]  7.056614837102176E-016
```

```
[ 1.000000000000000E-001, 6.620000000000001E-001]  7.054676898859712E-016
[ 6.620000000000000E-001, 7.000000000000000E-001]  7.093147399533771E-016
[ 6.999999999999999E-001, 1.000000000000000E+001]  6.990794400370803E-016
[ 1.000000000000000E+001, 1.000000000000000E+002]  5.481396262403598E-016
[ 1.000000000000000E+002, 7.097827128933840E+002]  7.077487262633923E-016

Relative Fehlerschranke der ffsinh-Funktion:  7.093147399533771E-016

Definitionsbereich fuer x: [ 2.578379799999999E-008, 7.097827128933840E+002]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 7.093147399533771E-016;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999989E-001 */
double q_...m  = 9007199254740982.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000002E+000 */
double q_...p  = 4503599627370503.0 / 4503599627370496.0;
```

Für $|x| \leq 2.5783798e-8$ wird die Approximation $\sinh(x) \approx x$ verwendet. Mit einfacher Handrechnung erhält man den für Ergebnisse im Bereich der normalisierten Zahlen gültigen relativen Approximationsfehler:

$$|\frac{\sinh(x)-x}{\sinh(x)}| \leq |\frac{x \cdot \sum_{n=1}^{\infty} \frac{x^{2n}}{(2n+1)!}}{\sinh(x)}| \leq |\sum_{n=1}^{\infty} \frac{x^{2n}}{(2n+1)!}| \leq \frac{x^2}{3!} \cdot \frac{1}{1-x^2} \leq 1.10801e-16$$

Im Bereich der denormalisierten Zahlen gilt der absolute Fehler:

$$|\sinh x - x| \leq |x \sum_{n=1}^{\infty} \frac{x^{2n}}{(2n+1)!}| \leq \frac{x^3}{3!} \cdot \frac{1}{1-x^2} \leq \texttt{dMinReal}$$

Gesamtfehlerschranke:
relative Fehlerschranke für $\texttt{q\_sinh}(x) \in S_N$: $\varepsilon(\texttt{q\_sinh}) \leq 7.0932E-016 = 6.39 \cdot \varepsilon^*$
absolute Fehlerschranke für $\texttt{q\_sinh}(x) \in S_D$: $\triangle(\texttt{q\_sinh}) \leq 4.9497E-324$

8.1.4 Algorithmus `j_sinh` (Intervallfunktion)

Der Gesamtalgorithmus `j_sinh` zur Berechnung von $\sinh(X)$ für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_snhm} := (1 \bigtriangledown \varepsilon(\texttt{q\_sinh})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_snhp} := (1 \bigtriangleup \varepsilon(\texttt{q\_sinh})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\text{INF} \leq -\texttt{q\_minr}$:
$y := \texttt{q\_sinh}(x.\text{INF})$
res.INF:= $y \boxdot \texttt{q\_snhp}$
res.SUP:= $y \boxdot \texttt{q\_snhm}$
Falls (res.SUP> x.INF) dann: res.SUP:= x.INF

b) $-\texttt{q\_minr} < x.\mathrm{INF} < 0$:
res.INF:= pred(x.INF)
res.SUP:= x.INF

c) $0 \leq x.\mathrm{INF} < \texttt{q\_minr}$:
res.INF:= x.INF
Falls (x.INF = 0) dann: res.SUP:= 0
sonst: res.SUP:= succ(x.INF)

d) $\texttt{q\_minr} \leq x.\mathrm{INF}$:
y := $\texttt{q\_sinh}(x.\mathrm{INF})$
res.INF:= $y \boxdot \texttt{q\_snhm}$
res.SUP:= $y \boxdot \texttt{q\_snhp}$
Falls (res.INF< x.INF) dann: res.INF:= x.INF

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\mathrm{INF} \leq -\texttt{q\_minr}$:
res.INF:= $\texttt{q\_sinh}(x.\mathrm{INF}) \boxdot \texttt{q\_snhp}$

ii. $-\texttt{q\_minr} < x.\mathrm{INF} < 0$:
res.INF:= pred(x.INF)

iii. $0 \leq x.\mathrm{INF} < \texttt{q\_minr}$:
res.INF:= x.INF

iv. $\texttt{q\_minr} \leq x.\mathrm{INF}$:
res.INF:= $\texttt{q\_sinh}(x.\mathrm{INF}) \boxdot \texttt{q\_snhm}$
Falls (res.INF< x.INF) dann: res.INF:= x.INF

b) Berechnung der Oberschranke res.SUP:

i. $x.\mathrm{SUP} \leq -\texttt{q\_minr}$:
res.SUP:= $\texttt{q\_sinh}(x.\mathrm{SUP}) \boxdot \texttt{q\_snhm}$
Falls (res.SUP> x.SUP) dann: res.SUP:= x.SUP

ii. $-\texttt{q\_minr} < x.\mathrm{SUP} le 0$:
res.SUP:= x.SUP

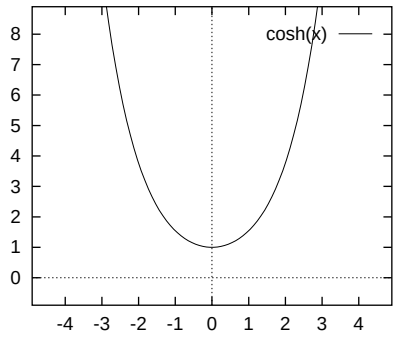
iii. $0 < x.\mathrm{SUP} < \texttt{q\_minr}$:
res.SUP:= succ(x.SUP)

iv. $\texttt{q\_sinh} \leq x.\mathrm{SUP}$:
res.SUP:= $\texttt{q\_sinh}(x.\mathrm{SUP}) \boxdot \texttt{q\_snhp}$

Nun gilt $\texttt{j\_sinh} := \mathrm{res} \supseteq \sinh(X)$ mit res := [res.INF, res.SUP].

8.2 Cosinus Hyperbolicus - Funktion: cosh(x)

hyperbolische Cosinusfunktion	
Math. Schreibweise	$\cosh x = \frac{e^x + e^{-x}}{2}$
Definitionsbereich	$\mathbb{R}$
Potenzreihe für $\lvert x\rvert < \infty$	$\sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$ $= 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots$
1. Ableitung	$\sinh(x)$
Nullstellen	keine
Minima	$x_T = 0$
Symmetrie	Symmetrie zur y-Achse
Monotonie	monoton fallend für $x < 0$ monoton steigend für $x > 0$



8.2.1 Idee

Die cosh-Funktion wird auf die exp-Funktion zurückgeführt. Es gilt:

$$\cosh(x) = 0.5 \cdot (e^{|x|} + e^{-|x|}) = 0.5 \cdot (e^x + e^{-x})$$

8.2.2 Algorithmus `q_cosh` (Punktfunktion)

Der Gesamtalgorithmus `q_cosh` zur Berechnung von $\cosh(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| >$`q_ex2c`$= 709.089... \Longrightarrow$ Fehlermeldung: Overflow (Überlauf)

II. Berechnung mit Hilfe der Exponentialfunktion
res $:= 0.5 \cdot (\texttt{q\_exp}(x) \boxplus \texttt{q\_exp}(-x))$

Nun gilt `q_cosh` := res $\approx \cosh(x)$.

8.2.3 Fehlerabschätzung

Es wird nur der Fall $x \geq 0$ betrachtet, da die Funktion achsensymmetrisch zur y-Achse ist. Die Fehlerabschätzung wird mit Hilfe des Programms `ffcosh.p` durchgeführt, es wird das folgende Ergebnisprotokoll ausgebegen:

```
Test mit 1000 Zufallszahlen:
Minimale Groessenordnung der Fehlerschranke :  4.578447000000007E-016

[ 0.000000000000000E+000, 1.000000000000000E+000]  4.580736795901527E-016
[ 1.000000000000000E+000, 1.000000000000000E+001]  4.579133818615485E-016
[ 1.000000000000000E+001, 5.000000000000000E+001]  4.581500315658812E-016
```

```
[ 5.000000000000000E+001, 1.000000000000000E+002]  4.580736795901290E-016
[ 1.000000000000000E+002, 5.000000000000000E+002]  4.589907438747163E-016
[ 5.000000000000000E+002, 7.097827128933840E+002]  4.588061871968667E-016

Relative Fehlerschranke der ffcosh-Funktion:  4.589907438747163E-016

Definitionsbereich fuer x: [ 0.000000000000000E+000, 7.097827128933840E+002]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 4.589907438747163E-016;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999991E-001 */
double q_...m  = 9007199254740984.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000001E+000 */
double q_...p  = 4503599627370502.0 / 4503599627370496.0;
```

Gesamtfehlerschranke:
relative Fehlerschranke für $\texttt{q\_cosh}(x) \in S_N$: $\varepsilon(\texttt{q\_cosh}) \leq 4.5900E-016 = 4.13 \cdot \varepsilon^*$

8.2.4 Algorithmus `j_cosh` (Intervallfunktion)

Der Gesamtalgorithmus `j_cosh` zur Berechnung von $\cosh(X)$ für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_cshm} := (1 \bigtriangledown \varepsilon(\texttt{q\_cosh})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_cshp} := (1 \bigtriangleup \varepsilon(\texttt{q\_cosh})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. $x.\text{SUP} < 0$

a) X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$
$y := \texttt{q\_cosh}(x.\text{INF})$
res.INF:= $y \boxdot \texttt{q\_cshm}$
res.SUP:= $y \boxdot \texttt{q\_cshp}$

b) X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$
res.INF:= $\texttt{q\_cosh}(x.\text{SUP}) \boxdot \texttt{q\_cshm}$
res.SUP:= $\texttt{q\_cosh}(x.\text{INF}) \boxdot \texttt{q\_cshp}$

Falls (res.INF< 1.0) dann: res.INF:= 1.0

II. $x.\text{INF} > 0$

a) X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$
$y := \texttt{q\_cosh}(x.\text{INF})$
res.INF:= $y \boxdot \texttt{q\_cshm}$
res.SUP:= $y \boxdot \texttt{q\_cshp}$

b) X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$
res.INF:= `q_cosh`$(x.\text{INF}) \boxdot$ `q_cshm`
res.SUP:= `q_cosh`$(x.\text{SUP}) \boxdot$ `q_cshp`

Falls (res.INF< 1.0) dann: res.INF:$= 1.0$

III. $0 \in X$

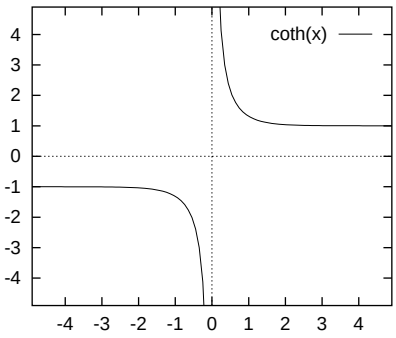
a) $-x.\text{INF} > x.\text{SUP}$
res.INF:$= 1.0$
res.SUP:= `q_cosh`$(x.\text{INF}) \boxdot$ `q_cshp`

b) $-x.\text{INF} \leq x.\text{SUP}$
res.INF:$= 1.0$
res.SUP:= `q_cosh`$(x.\text{SUP}) \boxdot$ `q_cshp`

Nun gilt `j_acosh` := res $\supseteq \cosh(X)$ mit res := [res.INF, res.SUP].

8.3 Cotangens Hyperbolicus - Funktion: coth(x)

hyperbolische Cotangensfunktion	
Math. Schreibweise	$\coth x = \frac{e^x+e^{-x}}{e^x-e^{-x}}$
Definitionsbereich	$(-\infty, 0) \cup (0, \infty)$
1. Ableitung	$-\frac{1}{(\sinh(x))^2}$
Nullstellen	keine
Symmetrie	Punktsymmetrie zu $(0, 0)$
Monotonie	monoton fallend



8.3.1 Idee

Die coth-Funktion wird für $|x| < \frac{\ln 2}{2}$ mit

$$\coth(x) = \text{sign}(x) \cdot (1 + \frac{2}{\text{expm1}(2 \cdot |x|)})$$

und für $|x| \geq \frac{\ln 2}{2}$ mit

$$\coth(x) = \text{sign}(x) \cdot (1 + \frac{2}{e^{2 \cdot |x|} - 1})$$

auf die Berechnung der exp- bzw. expm1-Funktion zurückgeführt. Für $|x| > 22.875$ kann als Ergebnis direkt $+1$ bzw. -1 gesetzt werden.

8.3.2 Algorithmus q_coth (Punktfunktion)

Der Gesamtalgorithmus `q_coth` zur Berechnung von $\coth(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| < 5.56268...e-309 \Longrightarrow$ Fehlermeldung: Overflow (Überlauf)
- $|x| > 22.875 \Longrightarrow \text{res} := \text{sign}(x)$

II. Berechnung mit Hilfe der Exponentialfunktionen

a) $|x| \geq$ `q_ln2h` $= 0.34657\ldots$:
$\text{res} := \text{sign}(x) \cdot (1 \boxplus 2 \boxslash (\texttt{q\_exp}(2 \cdot |x|) \boxminus 1))$

b) $|x| <$ `q_ln2h`:
$\text{res} := \text{sign}(x) \cdot (1 \boxplus 2 \boxslash \texttt{q\_expm1}(2 \cdot |x|))$

Nun gilt `q_coth` $:= \text{res} \approx \coth(x)$.

8.3.3 Fehlerabschätzung

Es wird nur der Fall $x \geq 0$ betrachtet, da die Funktion punktsymmetrisch zum Ursprung ist (Vorzeichenwechsel ist fehlerfrei möglich). Die Fehlerabschätzung wird im Intervall $[5.56268\ldots e-309, 22.875]$ mit Hilfe der Programme `ffcoth.p` durchgeführt.

```
Test mit 1000 Zufallszahlen:
Minimale Groessenordnung der Fehlerschranke :  8.202139303014012E-016

[ 5.562684646268013E-309, 1.000000000000000E-308]  8.882494592480248E-016
[ 9.999999999999999E-309, 1.112536929253601E-308]  6.911300005978952E-016
[ 1.112536929253600E-308, 2.225073858507202E-308]  8.882674277110517E-016
[ 2.225073858507201E-308, 1.000000000000001E-307]  7.900522053740879E-016
[ 1.000000000000000E-307, 1.000000000000001E-306]  7.340312742583798E-016
[ 1.000000000000000E-306, 1.000000000000001E-305]  8.155935556340593E-016
...
[ 1.000000000000000E-004, 1.000000000000001E-003]  8.148710359082969E-016
[ 1.000000000000000E-003, 1.000000000000001E-002]  8.143923301000462E-016
[ 1.000000000000000E-002, 1.000000000000001E-001]  8.095978332097705E-016
[ 1.000000000000000E-001, 3.465735902799727E-001]  6.779492434414347E-016
[ 3.465735902799726E-001, 7.000000000000000E-001]  8.328083581166275E-016
[ 6.999999999999999E-001, 2.287500000000000E+001]  7.885211217079232E-016

Relative Fehlerschranke der ffcoth-Funktion:  8.882674277110517E-016

Definitionsbereich fuer x: [ 3.465735902799726E-001, 3.465735902799727E-001]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 8.882674277110517E-016;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999987E-001 */
```

```
double q_...m  = 9007199254740980.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.0000000000000002E+000 */
double q_...p  = 4503599627370504.0 / 4503599627370496.0;
```

Gesamtfehlerschranke:
relative Fehlerschranke für q_coth(x) $\in S_N$: ε(q_coth) $\leq 8.8827E-016 = 8.01 \cdot \varepsilon^*$

Hinweis: Nach Einschränkung des zulässigen Definitionsbereiches auf Argumente $x \in S_N$ läst sich eine etwas kleinere relative Fehlerschranke herleiten, es gilt dann auf jeden Fall ε(q_coth) $< 8.0 \cdot \varepsilon^*$.

8.3.4 Algorithmus j_coth (Intervallfunktion)

Der Gesamtalgorithmus j_coth zur Berechnung von coth(x) für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_cthm} := (1 \bigtriangledown \varepsilon(\texttt{q\_coth})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_cthp} := (1 \bigtriangleup \varepsilon(\texttt{q\_coth})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. $x.\text{SUP} < 0$

a) X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$
$y :=$ q_coth(x.INF)
res.INF:= $y \boxdot$ q_cthp
res.SUP:= $y \boxdot$ q_cthm

b) X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$
res.INF:= q_coth(x.SUP) $\boxdot$ q_cthp
res.SUP:= q_coth(x.INF) $\boxdot$ q_cthm

Falls (res.SUP> -1.0) dann: res.SUP:$= -1.0$

II. $x.\text{INF} > 0$

a) X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$
$y :=$ q_coth(x.INF)
res.INF:= $y \boxdot$ q_cthm
res.SUP:= $y \boxdot$ q_cthp

b) X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$
res.INF:= q_coth(x.SUP) $\boxdot$ q_cthm
res.SUP:= q_coth(x.INF) $\boxdot$ q_cthp

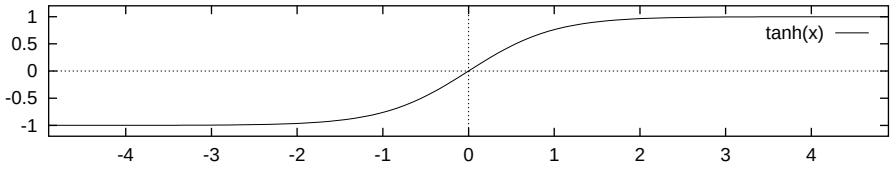
Falls (res.INF< 1.0) dann: res.INF:$= 1.0$

III. $0 \in X \Longrightarrow$ Fehlermeldung: Invalid Argument

Nun gilt j_coth := res $\supseteq$ coth(X) mit res := [res.INF, res.SUP].

8.4 Tangens Hyperbolicus - Funktion: tanh(x)

hyperbolische Tangensfunktion	
Math. Schreibweise	$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Definitionsbereich	$\mathbb{R}$
1. Ableitung	$\frac{1}{(\cosh(x))^2}$
Nullstellen	$x_0 = 0$
Symmetrie	Punktsymmetrie zu $(0, 0)$
Monotonie	monoton steigend



8.4.1 Idee

Die tanh-Funktion wird für $|x| \leq 1e-10$ mit

$$\tanh(x) = \frac{1}{\coth(x)}$$

auf die Berechnung der coth-Funktion zurückgeführt. Für $|x| < 1e-10$ wird die Approximation

$$\tanh(x) \approx x$$

verwendet.

8.4.2 Algorithmus `q_tanh` (Punktfunktion)

Der Gesamtalgorithmus `q_tanh` zur Berechnung von $\tanh(x)$ stellt sich wie folgt dar:

I. Abprüfen von Spezialfälle und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| < 1e-10 \Longrightarrow$ res $:= x$

II. Berechnung mit Hilfe der hyperbolischen Cotangensfunktion
res $:= 1 \not\boxplus$ (`q_coth`(x)

Nun gilt `q_tanh` $:=$ res $\approx \tanh(x)$.

8.4.3 Fehlerabschätzung

Es wird nur der Fall $x \geq 0$ betrachtet, da sowohl tanh, wie auch die Funktion coth punktsymmetrisch zum Ursprung sind (Vorzeichenwechsel ist fehlerfrei möglich).

Die Fehlerschranke für $x \in [1e-10, \texttt{maxreal}]$ läst sich am schnellsten mit folgender Handrechnung herleiten:

$$1 \boxplus \widetilde{\coth(x)} = \frac{1}{\coth(x)} \frac{1+\varepsilon}{1+\varepsilon_{\texttt{q\_coth}}} \leq \frac{1}{\coth(x)} (1 + \overline{\varepsilon} + 1.1 \cdot \varepsilon(\texttt{q\_coth}) + 1.1 \cdot \overline{\varepsilon} \cdot \varepsilon(\texttt{q\_coth}))$$

Das numerische Ergebnis, sowie die Konstanten für das ANSI-C Programm werden mit Hilfe des Programms `fftanh.p` berechnet:

```
Relative Fehlerschranke der fftanh-Funktion:  1.119197400000001E-015

Definitionsbereich fuer x: [ 1.000000000000000E-010, 1.797693134862316E+308]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 1.119197400000001E-015;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999984E-001 */
double q_...m  = 9007199254740978.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000002E+000 */
double q_...p  = 4503599627370505.0 / 4503599627370496.0;
```

Für $|x| \leq 1e-10$ wird die Approximation $\tanh(x) \approx x$ verwendet. Ebenfalls mit Handrechnung erhält man den für Ergebnisse im Bereich der normalisierten Zahlen gültigen relativen Approximationsfehler (siehe auch [13], Seite 107):

$$\left| \frac{\tanh x - x}{\tanh x} \right| \leq \frac{5}{3} \cdot \frac{1}{1 - \frac{x^2}{3}} \cdot x^2 \leq 1.67e-20 := \varepsilon(\mathrm{app}_N)$$

Im Bereich der denormalisierten Zahlen gilt der absolute Fehler:

$$|\tanh x - x| = |-\frac{1}{3}x^3 + \frac{2}{15}x^5 - + \ldots| \leq \texttt{dMinReal} := \triangle(\mathrm{app}_D)$$

Gesamtfehlerschranke:
relative Fehlerschranke für $\texttt{q\_tanh}(x) \in S_N$: $\varepsilon(\texttt{q\_tanh}) \leq 1.1192E-015 = 10.09 \cdot \varepsilon^*$
absolute Fehlerschranke für $\texttt{q\_tanh}(x) \in S_D$: $\triangle(\texttt{q\_tanh}) \leq 4.9497E-324$

8.4.4 Algorithmus `j_tanh` (Intervallfunktion)

Der Gesamtalgorithmus `j_tanh` zur Berechnung von $\tanh(X)$ für Intervallargumente $X := [x.\mathrm{INF}, x.\mathrm{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_tnhm} := (1 \bigtriangledown \varepsilon(\texttt{q\_tanh})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_tnhp} := (1 \bigtriangleup \varepsilon(\texttt{q\_tanh})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. X ist Punktintervall, d.h. $x.\mathrm{INF} = x.\mathrm{SUP}$

a) $x.\mathrm{INF} \leq -\texttt{q\_minr}$:
$y := \texttt{q\_tanh}(x.\mathrm{INF})$
res.INF:= $y \boxdot \texttt{q\_tnhp}$
res.SUP:= $y \boxdot \texttt{q\_tnhm}$
Falls (res.INF$< x.$INF) dann: res.INF:= $x.$INF

b) $-\texttt{q\_minr} < x.\mathrm{INF} < 0$:
res.INF:= $x.$INF
res.SUP:= succ($x.$INF)

c) $0 \leq x.\mathrm{INF} < \texttt{q\_minr}$:
Falls ($x.\mathrm{INF} = 0$) dann: res.INF:= 0
sonst: res.INF:= pred($x.$INF)
res.SUP:= $x.$INF

d) $\texttt{q\_minr} \leq x.\mathrm{INF}$:
$y := \texttt{q\_tanh}(x.\mathrm{INF})$
res.INF:= $y \boxdot \texttt{q\_tnhm}$
res.SUP:= $y \boxdot \texttt{q\_tnhp}$
Falls (res.SUP$> x.$INF) dann: res.SUP:= $x.$INF

II. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\mathrm{INF} \leq -\texttt{q\_minr}$:
res.INF:= $\texttt{q\_tanh}(x.\mathrm{INF}) \boxdot \texttt{q\_tnhp}$
Falls (res.INF$< x.$INF) dann: res.INF:= $x.$INF

ii. $-\texttt{q\_minr} < x.\mathrm{INF} \leq 0$:
res.INF:= $x.$INF

iii. $0 < x.\mathrm{INF} < \texttt{q\_minr}$:
res.INF:= pred($x.$INF)

iv. $\texttt{q\_minr} \leq x.\mathrm{INF}$:
res.INF:= $\texttt{q\_tanh}(x.\mathrm{INF}) \boxdot \texttt{q\_tnhm}$

b) Berechnung der Oberschranke res.SUP:

i. $x.\mathrm{SUP} \leq -\texttt{q\_minr}$:
res.SUP:= $\texttt{q\_tanh}(x.\mathrm{SUP}) \boxdot \texttt{q\_tnhm}$

ii. $-\texttt{q\_minr} < x.\mathrm{SUP} < 0$:
res.SUP:= succ($x.$SUP)

iii. $0 \leq x.\mathrm{SUP} < \texttt{q\_minr}$:
res.SUP:= $x.$SUP

iv. $\texttt{q\_minr} \leq x.\mathrm{SUP}$:
res.SUP:= $\texttt{q\_tanh}(x.\mathrm{SUP}) \boxdot \texttt{q\_tnhp}$
Falls (res.SUP$> x.$SUP) dann: res.SUP:= $x.$SUP

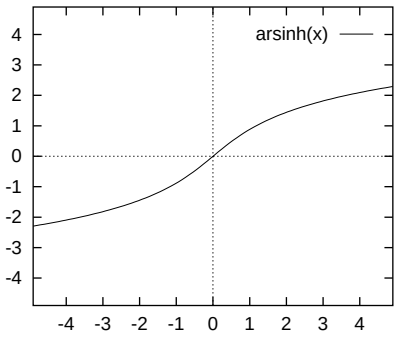
Falls (res.INF< -1.0) dann: res.INF:= -1.0
Falls (res.SUP> 1.0) dann: res.SUP:= 1.0
Nun gilt $\texttt{j\_tanh} :=$ res $\supseteq \tanh(X)$ mit res := [res.INF, res.SUP].

Kapitel 9

Inverse Hyperbolische Funktionen

9.1 Areasinus: arsinh(x)

Areasinusfunktion	
Math. Schreibweise	$\operatorname{arsinh} x = \ln(x + \sqrt{x^2 + 1})$
Definitionsbereich	$\mathbb{R}$
1. Ableitung	$\frac{1}{\sqrt{1+x^2}}$
Nullstellen	$x_0 = 0$
Symmetrie	Punktsymmetrie zu $(0, 0)$
Monotonie	monoton steigend



9.1.1 Idee

Für Argumente x mit $2.5e-8 < |x| < 1.25$ wird die Formel

$$\operatorname{arsinh}(x) = \operatorname{sign}(x) \cdot \operatorname{ln1p}\left(x + \frac{x}{\sqrt{1 + \left(\frac{1}{x}\right)^2} + \frac{1}{x}}\right)$$

verwendet. Für Argumente $1.25 \leq |x| \leq 10^{150}$ wird

$$\operatorname{arsinh}(x) = \operatorname{sign}(x) \cdot \ln(x + \sqrt{x^2 + 1})$$

angewandt. Für große Argumente $|x| > 10^{150}$ kann die Approximation

$$\operatorname{arsinh}(x) \approx \operatorname{sign}(x) \cdot \ln(2 \cot x) = \ln 2 + \ln(x)$$

benutzt werden. Kleine Argumente $|x| < 2.5 \cdot 10^{-8}$ können mit

$$\operatorname{arsinh}(x) \approx x$$

approximiert werden.

9.1.2 Algorithmus q_asnh (Punktfunktion)

I. Abprüfen von Spezialfällen und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| \leq 2.5e-8 \Longrightarrow$ res $:= x$

II. Initialisierung
Falls $x < 0$ dann: $x := -x$ und neg := `true`
Sonst: neg := `false`

III. Fallunterscheidung

a) $x > 1e150$:
res := $\texttt{q\_l2} \boxplus \texttt{q\_log}(x)$ mit $\texttt{q\_l2} \approx \ln 2$

b) $2.5 \leq x \leq 1e150$:
res := $\texttt{q\_log}(x \boxplus \texttt{q\_sqrt}((x \boxdot x) \boxplus 1))$

c) $x < 2.5$:
$h := 1/x$
res := $\texttt{q\_lg1p}(x \boxplus x \boxslash (\texttt{q\_sqrt}((1 \boxplus h \boxdot h) \boxplus h))$

Falls neg = `true` dann: res := $-$res

Nun gilt `q_asnh` := res $\approx \operatorname{arsinh}(x)$.

9.1.3 Fehlerabschätzung

Die Fehlerabschätzung wird im Intervall $[2.5e-8, \texttt{maxreal}]$ mit Hilfe des Programms `ffasnh.p` durchgeführt.

```
Test mit 1000 Zufallszahlen:
Minimale Groessenordnung der Fehlerschranke :  5.636156198771210E-016

Test mit x:=2.5
Minimale Groessenordnung der Fehlerschranke :  5.637562879207643E-016

[ 2.499999999999999E-008, 1.000000000000000E-007]  4.742833049799722E-016
[ 9.999999999999999E-008, 1.000000000000001E-005]  5.196783540998713E-016
[ 1.000000000000000E-005, 1.000000000000001E-003]  5.196831137784061E-016
[ 1.000000000000000E-003, 1.000000000000001E-001]  5.201585860973911E-016
[ 1.000000000000000E-001, 1.000000000000000E+000]  5.646009139541352E-016
[ 1.000000000000000E+000, 1.250000000000000E+000]  5.616072199058608E-016
[ 1.250000000000000E+000, 2.500000000000000E+000]  5.593305183271121E-016
[ 2.500000000000000E+000, 3.000000000000000E+000]  5.638506148995451E-016
[ 3.000000000000000E+000, 5.000000000000000E+000]  5.411984648287889E-016
[ 5.000000000000000E+000, 1.000000000000000E+001]  4.896184980575515E-016
[ 1.000000000000000E+001, 1.000000000000000E+002]  4.673654895459556E-016
[ 1.000000000000000E+002, 1.000000000000000E+003]  3.921765351288440E-016
[ 1.000000000000000E+003, 1.000000000000000E+004]  3.624301872473180E-016
[ 1.000000000000000E+004, 1.000000000000000E+005]  3.465153573016590E-016
```

```
[ 1.000000000000000E+005, 1.000000000000000E+006]  3.366049517580098E-016
[ 1.000000000000000E+006, 1.000000000000000E+007]  3.298401923797823E-016
[ 1.000000000000000E+007, 1.000000000000000E+008]  3.249285302716885E-016
[ 1.000000000000000E+008, 1.000000000000000E+009]  3.212002543483568E-016
[ 1.000000000000000E+009, 1.000000000000000E+010]  3.182736694044263E-016
...
[ 9.999999999999999E+146, 1.000000000000000E+148]  3.385289839186767E-016
[ 9.999999999999999E+147, 1.000000000000000E+149]  3.382285882712561E-016
[ 9.999999999999999E+148, 1.000000000000000E+150]  3.379322166502932E-016
[ 9.999999999999999E+149, 1.000000000000001E+151]  5.190915384273793E-016
[ 1.000000000000000E+151, 1.000000000000000E+152]  5.190712686484660E-016
[ 9.999999999999999E+151, 1.000000000000000E+153]  5.190512650500100E-016
...
[ 9.999999999999999E+303, 1.000000000000000E+305]  5.175394795407465E-016
[ 9.999999999999999E+304, 9.999999999999999E+305]  5.175345179474600E-016
[ 9.999999999999998E+305, 9.999999999999999E+306]  5.175295887510173E-016
[ 9.999999999999998E+306, 1.797693134862316E+308]  5.179524139906275E-016

Relative Fehlerschranke der ffarsinh-Funktion:  5.646009139541352E-016

Definitionsbereich fuer x: [ 2.499999999999999E-008, 1.797693134862316E+308]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 5.646009139541352E-016;                   */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999990E-001 */
double q_...m  = 9007199254740983.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000001E+000 */
double q_...p  = 4503599627370502.0 / 4503599627370496.0;
```

Für $|x| > 1e150$ wird bei der Berechung unter anderem die Approximation

$$2 \cdot x = x + x \approx x + \sqrt{x^2 + 1}$$

benutzt. Unter Berücksichtigung der Monotonieeigenschaft von $h(x) := \sqrt{x^2+1} - x$ gilt für den absoluten Fehler

$$|\sqrt{x^2+1} - x| = |h(x)| \leq h(1e150) \leq 5.00001e - 151.$$

Der rechtsstehende Wert der letzten Ungleichung kann leicht mit Hilfe einer Langzahlarithmetik bestimmt werden (siehe z.B. Programm `ffasnh_1.p`). Diese Fehlerabschätzung geht im Programm `ffasnh.p` in der Funktion `maxfehler3` in die Berechnung mit ein.

Für $|x| \leq 2.5e{-}8$ wird die Approximation $\operatorname{arsinh}(x) \approx x$ verwendet. Mit Handrechnung erhält man den für Ergebnisse im Bereich der normalisierten Zahlen gültigen relativen Approximationsfehler (siehe auch [28], Seite 60):

$$\left|\frac{\operatorname{arsinh}x - x}{\operatorname{arsinh}x}\right| \leq \frac{\frac{1}{2\cdot 3}x^2}{1 - \frac{1}{2\cdot 3}x^2} \leq 1.0417e - 16 := \varepsilon(\mathrm{app}_N)$$

Im Bereich der denormalisierten Zahlen gilt der absolute Fehler:

$$|\operatorname{arsinh}x - x| = |-\frac{1}{2\cdot 3}x^3 + \frac{1\cdot 3}{2\cdot 4\cdot 5}x^5 - + \ldots| \leq \texttt{dMinReal} := \triangle(\mathrm{app}_D)$$

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_asnh}(x) \in S_N$: $\varepsilon(\texttt{q\_asnh}) \leq 5.6461E-016 = 5.09 \cdot \varepsilon^*$
Absolute Fehlerschranke für $\texttt{q\_asnh}(x) \in S_D$: $\triangle(\texttt{q\_asnh}) \leq 4.9497E-324$

9.1.4 Algorithmus `j_asnh` (Intervallfunktion)

Der Gesamtalgorithmus `j_asnh` zur Berechnung von arsinh(X) für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_asnm} := (1 \bigtriangledown \varepsilon(\texttt{q\_asnh})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_asnp} := (1 \bigtriangleup \varepsilon(\texttt{q\_asnh})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\text{INF} \leq -\texttt{q\_minr}$:
$y := \texttt{q\_asnh}(x.\text{INF})$
res.INF:= $y \boxdot \texttt{q\_asnp}$
res.SUP:= $y \boxdot \texttt{q\_asnm}$
Falls (res.INF< x.INF) dann: res.INF:= x.INF

b) $-\texttt{q\_minr} < x.\text{INF} < 0$:
res.INF:= x.INF
res.SUP:= succ(x.INF)

c) $0 \leq x.\text{INF} < \texttt{q\_minr}$:
Falls (x.INF = 0) dann: res.INF:= 0
sonst: res.INF:= pred(x.INF)
res.SUP:= x.INF

d) $\texttt{q\_minr} \leq x.\text{INF}$:
$y := \texttt{q\_asnh}(x.\text{INF})$
res.INF:= $y \boxdot \texttt{q\_asnm}$
res.SUP:= $y \boxdot \texttt{q\_asnp}$
Falls (res.SUP> x.INF) dann: res.SUP:= x.INF

II. X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\text{INF} \leq -\texttt{q\_minr}$:
res.INF:= $\texttt{q\_asnh}(x.\text{INF}) \boxdot \texttt{q\_asnp}$
Falls (res.INF< x.INF) dann: res.INF:= x.INF

ii. $-\texttt{q\_minr} < x.\text{INF} \leq 0$:
res.INF:= x.INF

iii. $0 < x.\text{INF} < \texttt{q\_minr}$:
res.INF:= pred(x.INF)

iv. $\texttt{q\_minr} \leq x.\text{INF}$:
res.INF:= $\texttt{q\_asnh}(x.\text{INF}) \boxdot \texttt{q\_asnm}$

b) Berechnung der Oberschranke res.SUP:

i. $x.\mathrm{SUP} \leq -$`q_minr`:
res.SUP:= `q_asnh`$(x.\mathrm{SUP}) \boxdot$ `q_asnm`

ii. $-$`q_minr` $< x.\mathrm{SUP} < 0$:
res.SUP:= succ$(x.\mathrm{SUP})$

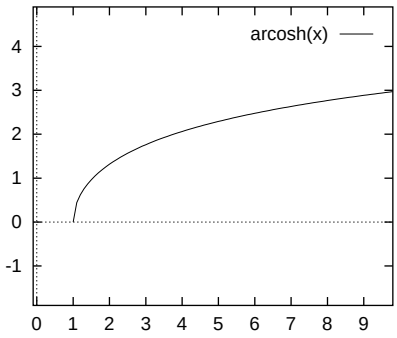
iii. $0 \leq x.\mathrm{SUP} <$ `q_minr`:
res.SUP:= $x.\mathrm{SUP}$

iv. `q_minr` $\leq x.\mathrm{SUP}$:
res.SUP:= `q_asnh`$(x.\mathrm{SUP}) \boxdot$ `q_asnp`
Falls (res.SUP$> x.\mathrm{SUP}$) dann: res.SUP:= $x.\mathrm{SUP}$

Nun gilt `j_asnh` := res $\supseteq \mathrm{arsinh}(X)$ mit res := [res.INF, res.SUP].

9.2 Areacosinus: arcosh(x)

Areacosinusfunktion	
Math. Schreibweise	$\mathrm{arcosh}\, x = \ln(x + \sqrt{x^2 - 1})$
Definitionsbereich	$[1, \infty)$
1. Ableitung	$\frac{1}{\sqrt{x^2-1}}$
Nullstellen	$x_0 = 1$
Minima	$x_T = 1$
Monotonie	monoton steigend



9.2.1 Idee

Falls das Eingabeargument x im zulässigen Definitionsbereich $D :=$ [1, `maxreal`] liegt (andernfalls wird eine Fehlermeldung ausgegeben), werden die folgenden drei Fälle unterschieden:

- $x < 1.025$:

$$\mathrm{arcosh} := \ln(1 + y) \quad \text{mit} \quad y := x - 1 + \sqrt{(x-1)(x+1)}$$

- $1.025 \leq x \leq 10^{150}$:

$$\mathrm{arcosh}(x) := \ln(x + \sqrt{(x-1)(x+1)})$$

- $10^{150} < x$:

$$\mathrm{arcosh}(x) := \ln 2\big|_{\mathrm{IEEE}} + \ln(x)$$

Im ersten Fall muß zur Berechnung die Funktion $\ln(x+1)$ (d.h. `q_lg1p`) verwendet werden. Eine Berechnung mit der Logarithmusfunktion $\ln(x)$ würde zu Auslöschung und in der Folge zu unbrauchbaren Ergebnissen führen.

Der Wert $\ln 2\big|_{\mathrm{IEEE}}$ wird im voraus berechnet und als Konstante abgespeichert.

9.2.2 Algorithmus q_acsh (Punktfunktion)

I. Abprüfen von Spezialfällen und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $x < 1 \Longrightarrow$ Fehlermeldung: Invalid Argument

II. Fallunterscheidung

a) $x < 1.025$:
res := $\texttt{q\_lg1p}(x \boxminus 1 \boxplus \texttt{q\_sqrt}((x \boxminus 1) \boxtimes (x \boxplus 1)))$

b) $1.025 \leq x \leq 1e150$:
res := $\texttt{q\_log}(x \boxplus \texttt{q\_sqrt}((x \boxminus 1) \boxtimes (x \boxplus 1)))$

c) $x > 1e150$:
res := $\texttt{q\_l2} \boxplus \texttt{q\_log}(x)$ mit $\texttt{q\_l2} \approx \ln 2$

Nun gilt q_acsh := res $\approx \operatorname{arcosh}(x)$.

9.2.3 Fehlerabschätzung

Die Fehlerabschätzung wird im Intervall $[1, 1e150]$ mit Hilfe des Programms ffacsh.p durchgeführt.

```
Test mit x:=1.025
Minimale Groessenordnung der Fehlerschranke :  1.736839599140470E-015

[ 1.000000000000000E+000, 1.000000000001001E+000]  1.027976446056508E-015
[ 1.000000000001000E+000, 1.000000000100001E+000]  1.643097720126808E-015
[ 1.000000000100000E+000, 1.000000010000000E+000]  1.643081274585971E-015
[ 1.000000009999999E+000, 1.000001000000000E+000]  1.642973945026761E-015
[ 1.000000999999999E+000, 1.000100000000000E+000]  1.641896657154981E-015
[ 1.000099999999999E+000, 1.025000000000000E+000]  1.415110177259564E-015
[ 1.024999999999999E+000, 1.025000000001001E+000]  1.736839599140470E-015
[ 1.025000000001000E+000, 1.025000000100000E+000]  1.736839599121260E-015
[ 1.025000000099999E+000, 1.025000010000001E+000]  1.736839597219637E-015
[ 1.025000010000000E+000, 1.025001000000001E+000]  1.736839407057038E-015
[ 1.025001000000000E+000, 1.025100000000000E+000]  1.736820391306869E-015
[ 1.025099999999999E+000, 1.100000000000001E+000]  1.736527399913530E-015
[ 1.100000000000000E+000, 2.000000000000000E+000]  1.167659422253004E-015
[ 2.000000000000000E+000, 1.000000000000000E+001]  6.651631410395162E-016
[ 1.000000000000000E+001, 1.000000000000000E+002]  7.707611547418609E-016
[ 1.000000000000000E+002, 1.000000000000000E+003]  5.625097936632048E-016
[ 1.000000000000000E+003, 1.000000000000000E+004]  4.811559461906476E-016
[ 1.000000000000000E+004, 1.000000000000000E+005]  4.376370322213856E-016
[ 1.000000000000000E+005, 1.000000000000000E+006]  4.105371979677345E-016
[ 1.000000000000000E+006, 1.000000000000000E+007]  3.920390797127829E-016
[ 1.000000000000000E+007, 1.000000000000000E+008]  3.786082240977235E-016
[ 1.000000000000000E+008, 1.000000000000000E+009]  1.374921873456078E-015
```

```
[ 1.000000000000000E+009, 1.000000000000000E+010]  1.258704435821751E-015
...
[ 9.999999999999999E+146, 1.000000000000000E+148]  3.548956019845365E-016
[ 9.999999999999999E+147, 1.000000000000000E+149]  3.544848455522561E-016
[ 9.999999999999999E+148, 1.000000000000000E+150]  3.540795915124546E-016

Relative Fehlerschranke der ffarcosh-Funktion:  1.736839599140470E-015

Definitionsbereich fuer x: [ 1.000000000000000E+000, 1.000000000000000E+001]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 1.736839599140469E-015;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999979E-001 */
double q_...m  = 9007199254740973.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000002E+000 */
double q_...p  = 4503599627370507.0 / 4503599627370496.0;
```

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_acsh}(x) \in S_N$: $\varepsilon(\texttt{q\_acsh}) \leq 1.7369E-015 = 15.65 \cdot \varepsilon^*$

9.2.4 Algorithmus `j_acsh` (Intervallfunktion)

Der Gesamtalgorithmus `j_acsh` zur Berechnung von arcosh (X) für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_acsm} := (1 \bigtriangledown \varepsilon(\texttt{q\_acsh})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_acsp} := (1 \bigtriangleup \varepsilon(\texttt{q\_acsh})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. Abprüfen des Definitionsbereiches
$x.\text{INF} < -1 \Longrightarrow$ Fehlermeldung: Invalid Argument

II. X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$

a) $x.\text{INF} = 1$:
res.INF:= 0
res.SUP:= 0

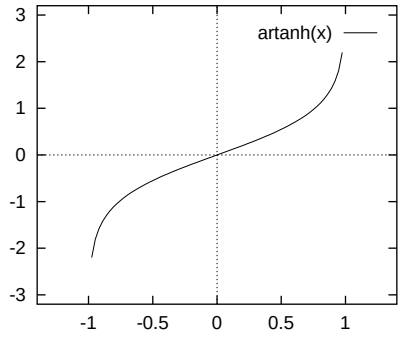
b) $x.\text{INF} > 1$:
$y :=$ `q_acsh`$(x.\text{INF})$
res.INF:= $y \boxdot$ `q_acsm`
res.SUP:= $y \boxdot$ `q_acsp`

III. X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$
res.INF:= `q_acsh`$(x.\text{INF}) \boxdot$ `q_acsm`
res.SUP:= `q_acsh`$(x.\text{SUP}) \boxdot$ `q_acsp`

Nun gilt `j_acsh` := res $\supseteq$ arcosh (X) mit res := [res.INF, res.SUP].

9.3 Areatangens: artanh(x)

Areatangensfunktion	
Math. Schreibweise	$\operatorname{artanh} x = \frac{1}{2}\ln(\frac{1+x}{1-x})$
Definitionsbereich	$(-1, 1)$
1. Ableitung	$\frac{1}{1-x^2}$
Nullstellen	$x_0 = 1$
Symmetrie	Punktsymmetrie zu $(0, 0)$
Monotonie	monoton steigend



9.3.1 Idee

Für kleine positive Argumente x mit $0 < x \leq \tau < \frac{1}{3}$ wird die Formel

$$\operatorname{artanh}(x) := \frac{1}{2}\ln(1 + \frac{2x}{1-x})$$

angewandt, alle anderen zulässigen positiven Argumente $x < 1$ werden mit

$$\operatorname{artanh}(x) := \frac{1}{2}\ln(\frac{1+x}{1-x})$$

berechnet. Für negative Argumente werden die Symmetrieeigenschaften des Areatangens ausgenutzt.

9.3.2 Algorithmus `q_atnh` (Punktfunktion)

I. Abprüfen von Spezialfällen und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| \geq 1 \Longrightarrow$ Fehlermeldung: Invalid Argument

II. Fallunterscheidung

a) $|x| < \bigtriangledown(\frac{1}{3})$:
res := $0.5 * \texttt{ln1p}((2 * |x|)/(1 - |x|)$
Falls $x < 0$: res := $-$res

b) $|x| \geq \bigtriangledown(\frac{1}{3})$:
res := $0.5 * \texttt{ln}((1 + |x|)/(1 - |x|)$
Falls $x < 0$: res := $-$res

Nun gilt `q_atnh` := res $\approx \operatorname{artanh}(x)$.

9.3.3 Fehlerabschätzung

Aufgrund der Symmetrieeigenschaften werden im folgenden nur positive Argumente betrachtet. Die abschließende Multiplikation mit dem Faktor $\frac{1}{2}$ kann in binären Gleitkommasystemen fehlerfrei ausgeführt werden und braucht daher nicht in der Fehlerabschätzung berücksichtigt werden.

I) Fehlerabschätzung für $0 < x \leq \tau < \frac{1}{3}$:
Der Ausdruck

$$u := \frac{2x}{1-x}$$

wird auf der Maschine ausgewertet, man erhält

$$\tilde{u} := (2 \boxdot x) \boxplus (1 \boxminus x) = (2 \cdot x) \boxplus (1 \boxminus x) = \frac{2x}{1-x}(1+\varepsilon_u).$$

Für den relativen Fehler ε_u gilt nach der Formel 1.2.6 aus [28]

$$|\varepsilon_u| \leq \varepsilon(u) := 2.02\overline{\varepsilon}$$

Wegen $0 < x < \tau$ gilt $0 \leq u \leq \frac{2\tau}{1-\tau} < 1$ und damit gilt nach Abschnitt 3.4.9 für die Auswertung der Funktion Logarithmus-minus-Eins mit fehlerbehaftetem Argument:

$$\left|\frac{\mathrm{ln1p}(u) - \mathrm{ln1p}(\tilde{u})}{\mathrm{ln1p}(u)}\right| \leq \varepsilon(\mathrm{ln1p}) + (1+\varepsilon(\mathrm{ln1p})) \cdot \varepsilon(u) \cdot \frac{2}{1-\varepsilon(u)}$$

Für $\overline{\varepsilon} :=$ EPS52 und $\tau := \bigtriangledown\frac{1}{3}$ kann die Fehlerschranke mit den folgenden Pascal-XSC Anweisungen berechnet werden:

```
Eps52   := 2.220447E-16;
EpsLn1p := 2.5082E-16;
EpsU    := 2.02 *> Eps52;
relerr1 := EpsLn1p +> (1 +> EpsLn1p) *> EpsU *> (2 /> (1 -< EpsU));
```

Man erhält als Ergebnis die Fehlerschranke

$$\varepsilon(\mathrm{artanh}_I) := \texttt{relerr1} = 1.147880588000001E-015.$$

II) Fehlerabschätzung für $0 < \tau \leq x < 1$:
Der Ausdruck

$$u := u(x) := \frac{1+x}{1-x}$$

wird auf der Maschine ausgewertet, man erhält

$$\tilde{u} := (1 \boxplus x) \boxslash (1 \boxminus x) = \frac{1+x}{1-x}(1+\varepsilon_u).$$

Für den relativen Fehler gilt nach der Formel 1.2.6 aus [28]

$$|\varepsilon_u| \leq \varepsilon(u) := 3.03\overline{\varepsilon}$$

Wegen $\tau \leq x < 1$ gilt $u \geq \frac{1+\tau}{1-\tau} > 1$ und damit kann die Abschätzung aus Abschnitt 3.4.8 für die Auswertung der Logarithmusfunktion mit fehlerbehaftetem Argument angewandt werden:

$$\left|\frac{\ln(u) - \tilde{\ln}(\tilde{u})}{\ln(u)}\right| \quad \leq \quad \varepsilon(\ln) + (1+\varepsilon(\ln)) \cdot \varepsilon(u) \cdot \frac{1}{1-\varepsilon(u)} \cdot \frac{1}{\ln(u(\tau))}$$

Hierbei wird ausgenutzt, daß $u(x)$ monoton wachsend in x ist.

Mit $\overline{\varepsilon} :=$ EPS52 und $\tau := \bigtriangledown\frac{1}{3}$ kann mit den Pascal-XSC Anweisungen

```
Eps52   := 2.220447E-16;
EpsLn   := 2.9398E-16;
EpsLn1p := 2.5082E-16;
EpsU    := 3.03 *> Eps52;
tau     := 1 /< 3;
LnWert  := inf( ln(intval( (1 +< tau) /< (1 -> tau) )) );
relerr2 := EpsLn +> (1 +> EpsLn) *> EpsU *> (1 /> (1 -< EpsU))
           *> 1 /> LnWert;
```

die folgende Fehlerschranke berechnet werden:

$$\varepsilon(\text{artanh}_{II}) := \texttt{relerr2} = 1.2646186462634 05E-015$$

III) Gesamtfehlerabschätzung:
Der relative Gesamtfehler ergibt sich durch

$$\varepsilon(\text{artanh}) := \max\{\varepsilon(\text{artanh}_I), \varepsilon(\text{artanh}_{II})\} \leq 1.2647E-015$$

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_atnh}(x) \in S_N$: $\varepsilon(\texttt{q\_atnh}) \leq 1.2647E-015 = 11.40 \cdot \varepsilon^*$
Absolute Fehlerschranke für $\texttt{q\_atnh}(x) \in S_D$: $\triangle(\texttt{q\_atnh}) \leq 4.9497E-324$

9.3.4 Algorithmus `j_atnh` (Intervallfunktion)

Der Gesamtalgorithmus `j_atnh` zur Berechnung von artanh (X) für Intervallargumente $X := [x.\mathrm{INF}, x.\mathrm{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_atnm} := (1 \bigtriangledown \varepsilon(\texttt{q\_atnh})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_atnp} := (1 \bigtriangleup \varepsilon(\texttt{q\_atnh})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. Abprüfen des Definitionsbereiches
$x.\mathrm{INF} \leq -1$ oder $1 \geq x.\mathrm{SUP} \Longrightarrow$ Fehlermeldung: Invalid Argument

II. X ist Punktintervall, d.h. $x.\mathrm{INF} = x.\mathrm{SUP}$

a) $x.\mathrm{INF} \leq -$`q_minr`:
$y :=$ `q_atnh`$(x.\mathrm{INF})$
res.INF:= $y \boxdot$ `q_atnp`
res.SUP:= $y \boxdot$ `q_atnm`
Falls (res.SUP$>x.\mathrm{INF}$) dann: res.SUP:= $x.\mathrm{INF}$

b) $-$`q_minr` $< x.\mathrm{INF} < 0$:
res.INF:= pred$(x.\mathrm{INF})$
res.SUP:= $x.\mathrm{INF}$

c) $0 \leq x.\mathrm{INF} <$ `q_minr`:
res.INF:= $x.\mathrm{INF}$
Falls $(x.\mathrm{INF} = 0)$ dann: res.SUP:= 0
sonst: res.SUP:= succ$(x.\mathrm{INF})$

d) `q_minr` $\leq x.\mathrm{INF}$:
$y :=$ `q_atnh`$(x.\mathrm{INF})$
res.INF:= $y \boxdot$ `q_atnm`
res.SUP:= $y \boxdot$ `q_atnp`
Falls (res.INF$<x.\mathrm{INF}$) dann: res.INF:= $x.\mathrm{INF}$

III. X ist echtes Intervall, d.h. $x.\mathrm{INF} \neq x.\mathrm{SUP}$

a) Berechnung der Unterschranke res.INF:

i. $x.\mathrm{INF} \leq -$`q_minr`:
res.INF:= `q_atnh`$(x.\mathrm{INF}) \boxdot$ `q_atnp`

ii. $-$`q_minr` $< x.\mathrm{INF} < 0$:
res.INF:= pred$(x.\mathrm{INF})$

iii. $0 \leq x.\mathrm{INF} <$ `q_minr`:
res.INF:= $x.\mathrm{INF}$

iv. `q_minr` $\leq x.\mathrm{INF}$:
res.INF:= `q_atnh`$(x.\mathrm{INF}) \boxdot$ `q_atnm`
Falls (res.INF$<x.\mathrm{INF}$) dann: res.INF:= $x.\mathrm{INF}$

b) Berechnung der Oberschranke res.SUP:

i. $x.\mathrm{SUP} \leq -$`q_minr`:
res.SUP:= `q_atnh`$(x.\mathrm{SUP})$ $\boxdot$ `q_atnm`
Falls (res.SUP$> x.$SUP) dann: res.SUP:= $x.$SUP

ii. $-$`q_minr` $< x.\mathrm{SUP}le0$:
res.SUP:= $x.$SUP

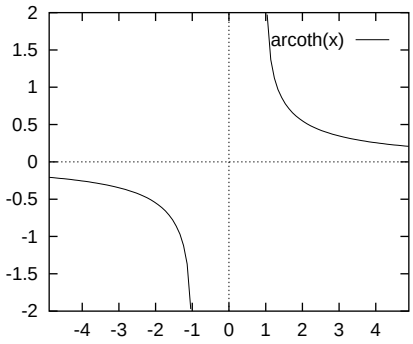
iii. $0 < x.\mathrm{SUP} <$ `q_minr`:
res.SUP:= succ$(x.\mathrm{SUP})$

iv. `q_sinh` $\leq x.$SUP:
res.SUP:= `q_atnh`$(x.\mathrm{SUP})$ $\boxdot$ `q_atnp`

Nun gilt `j_atnh` := res $\supseteq$ artanh(X) mit res := [res.INF, res.SUP].

9.4 Areacotangens: arcoth(x)

Areacotangensfunktion	
Math. Schreibweise	$\operatorname{arcoth} x = \frac{1}{2}\ln(\frac{x+1}{x-1})$
Definitionsbereich	$(-\infty,-1) \cup (1,\infty)$
1. Ableitung	$-\frac{1}{x^2-1}$
Nullstellen	keine
Symmetrie	Punktsymmetrie zu $(0,0)$
Monotonie	monoton fallend



9.4.1 Idee

Für alle Funktionsargumente $x \in D := [-$`maxreal`$,$ `maxreal`$]\backslash[-1,1]$ soll die Formel

$$\mathrm{arcoth} := \mathrm{sign}(x) \cdot 0.5 \cdot \ln\left(\frac{|x|+1}{|x|-1}\right) = \mathrm{sign}(x) \cdot 0.5 \cdot \mathrm{ln1p}\left(\frac{2}{|x|-1}\right)$$

herangezogen werden.

9.4.2 Algorithmus `q_acth` (Punktfunktion)

I. Abprüfen von Spezialfällen und deren Behandlung

- x ist NaN (not a number) $\Longrightarrow$ Fehlermeldung: Invalid Argument
- $|x| \leq 1 \Longrightarrow$ Fehlermeldung: Invalid Argument

II. Fallunterscheidung

a) $x > 1$:
res := $0.5 *$ `ln1p`$(2/(x-1)$

b) $x < -1$:
res := $-0.5 * \texttt{ln1p}(2/(|x| - 1)$

Nun gilt `q_acth` := res $\approx \operatorname{arcoth}(x)$.

9.4.3 Fehlerabschätzung

Aufgrund der Symmetrieeigenschaften des arcoth genügt es, den Bereich $1 < x \leq$ `maxreal` zu untersuchen. Aus beweistechnischen Gründen wird dieser Bereich aufgeteilt in die zwei Teilbereiche $1 < x \leq 3$ und $3 < x \leq$ `maxreal`.

I) Teilbereich I mit $1 < x \leq 3$:
Die Fehlerabschätzung wird mit Hilfe des Programms `ffacth_a.p` und des Moduls `eps_ari` durchgeführt.
Als Ergebnis erhält man die relative Fehlerschranke:

$$\varepsilon(\operatorname{arcoth}_I) := 5.711660947777037E - 016$$

II) Teilbereich II mit $3 < x \leq$ `maxreal`:
Der Ausdruck

$$u := \frac{2}{x - 1}$$

wird auf der Maschine ausgewertet, man erhält

$$\tilde{u} := 2 \boxplus (x \boxminus 1) = \frac{2}{x-1}(1 + \varepsilon_u).$$

Für den relativen Fehler ε_u gilt nach der Formel 1.2.6 aus [28]

$$|\varepsilon_u| \leq \varepsilon(u) := 2.02\overline{\varepsilon}$$

Wegen $x > 3$ gilt $0 < u < 1$ und damit gilt nach Abschnitt 3.4.9 für die Auswertung der „Logarithmusfunktion Minus Eins“ mit fehlerbehaftetem Argument:

$$\left| \frac{\operatorname{ln1p}(u) - \operatorname{ln1p}(\tilde{u})}{\operatorname{ln1p}(u)} \right| \leq \varepsilon(\operatorname{ln1p}) + (1 + \varepsilon(\operatorname{ln1p})) \cdot \varepsilon(u) \cdot \frac{2}{1 - \varepsilon(u)}$$

Für $\overline{\varepsilon}$:= EPS52 kann die Fehlerschranke mit den folgenden Pascal-XSC Anweisungen berechnet werden:

```
Eps52   := 2.220447E-16;
EpsLn1p := 2.5082E-16;
EpsU    := 2.02 *> Eps52;
relerr2 := EpsLn1p +> (1 +> EpsLn1p) *> EpsU *> (2 /> (1 -< EpsU));
```

Man erhält als Ergebnis die Fehlerschranke

$$\varepsilon(\text{arcoth}_{II}) := \texttt{relerr2} = 1.147880588000001E-015.$$

III) Gesamtfehlerabschätzung:
Der relative Gesamtfehler ergibt sich durch

$$\varepsilon(\text{arcoth}) := \max\{\varepsilon(\text{arcoth}_I), \varepsilon(\text{arcoth}_{II})\} \leq 1.1479E-015.$$

Ergebnisprotokoll:

```
Relative Fehlerschranke der ffacoth-Funktion:  1.147880588000002E-015

Definitionsbereich fuer x: [ 1.000000000000000E+000, 1.797693134862316E+308]

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 1.147880588000001E-015;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999984E-001 */
double q_...m  = 9007199254740978.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000002E+000 */
double q_...p  = 4503599627370505.0 / 4503599627370496.0;
```

Gesamtfehlerschranke:
Relative Fehlerschranke für $\texttt{q\_acth}(x) \in S_N$: $\varepsilon(\texttt{q\_acth}) \leq 1.1479E-015 = 10.34 \cdot \varepsilon^*$
Absolute Fehlerschranke für $\texttt{q\_acth}(x) \in S_D$: $\triangle(\texttt{q\_acth}) \leq 4.9497E-324$

9.4.4 Algorithmus `j_acth` (Intervallfunktion)

Der Gesamtalgorithmus `j_acth` zur Berechnung von arcoth(x) für Intervallargumente $X := [x.\text{INF}, x.\text{SUP}]$ stellt sich wie folgt dar:
Benötigte Konstanten:

$$\texttt{q\_actm} := (1 \bigtriangledown \varepsilon(\texttt{q\_acth})) \bigtriangledown (1 \bigtriangledown \texttt{EpsQuer})$$
$$\texttt{q\_actp} := (1 \bigtriangleup \varepsilon(\texttt{q\_acth})) \bigtriangleup (1 \bigtriangleup \texttt{EpsQuer})$$

I. Falls $x.\text{SUP} < -1$

a) X ist Punktintervall, d.h. $x.\text{INF} = x.\text{SUP}$
$y := \texttt{q\_acth}(x.\text{INF})$
res.INF:= $y \boxdot \texttt{q\_actp}$
res.SUP:= $y \boxdot \texttt{q\_actm}$

b) X ist echtes Intervall, d.h. $x.\text{INF} \neq x.\text{SUP}$
res.INF:= $\texttt{q\_acth}(x.\text{SUP}) \boxdot \texttt{q\_actp}$
res.SUP:= $\texttt{q\_acth}(x.\text{INF}) \boxdot \texttt{q\_actm}$

II. Falls x.INF > 1

a) X ist Punktintervall, d.h. x.INF $= x$.SUP
$y :=$ `q_acth`$(x$.INF$)$
res.INF:= $y \boxdot$ `q_actm`
res.SUP:= $y \boxdot$ `q_actp`

b) X ist echtes Intervall, d.h. x.INF $\neq x$.SUP
res.INF:= `q_acth`$(x$.SUP$) \boxdot$ `q_actm`
res.SUP:= `q_acth`$(x$.INF$) \boxdot$ `q_actp`

III. Sonst $\Longrightarrow$ Fehlermeldung: Invalid Argument

Nun gilt `j_acth` := res $\supseteq$ arcoth(X) mit res := [res.INF, res.SUP].

Kapitel 10

Intervallarithmetik in ANSI-C

10.1 Definitionen und Mathematischer Hintergrund

Eine Menge $A := [\underline{a}, \overline{a}] := \{x \in I\!R | \underline{a} \leq x \leq \overline{a}\}$ mit $\underline{a}, \overline{a} \in I\!R$ heißt abgeschlossenes reelles Intervall. Dabei bezeichnet $\underline{a} = \inf A$ die Unterschranke und $\overline{a} = \sup A$ die Oberschranke von A. Die Menge aller solcher Intervalle wird mit $I\!I\!R := \{A = [\underline{a}, \overline{a}] \mid \underline{a} \leq \overline{a}\}$ bezeichnet.

Die intervallarithmetischen Operationen für A, $B \in I\!I\!R$ werden gemäß

$$A \circ B := \{a \circ b \mid a \in A, b \in B\}$$

definiert, mit $\circ \in \{+, -, \cdot, /\}$. Man kann zeigen, daß

$$\begin{aligned}
A + B &= [\underline{a} + \underline{b}, \overline{a} + \overline{b}] \\
A - B &= [\underline{a} - \overline{b}, \overline{a} - \underline{b}] \\
A \cdot B &= [\min\{\underline{a} \cdot \underline{b}, \underline{a} \cdot \overline{b}, \overline{a} \cdot \underline{b}, \overline{a} \cdot \overline{b}\}, \max\{\overline{a} \cdot \overline{b}, \underline{a} \cdot \overline{b}, \overline{a} \cdot \underline{b}, \overline{a} \cdot \overline{b}\}] \\
A/B &= [\underline{a}, \overline{a}] \cdot [1/\overline{b}, 1/\underline{b}] \text{ für } 0 \notin B
\end{aligned}$$

gelten.

Beim Rechnen mit Intervallen auf einer Rechenanlage stehen statt der Menge der reellen Intervalle $I\!I\!R$ nur die Menge der Maschinenintervalle IS zur Verfügung. Ein Maschinenintervall $X \in IS$

$$X := [\underline{x}, \overline{x}] := \{x \in I\!R | \underline{x} \leq x \leq \overline{x}; \underline{x}, \overline{x} \in S(b, n)\}$$

ist dabei ein Intervall, dessen Endpunkte Maschinenzahlen des auf dem Rechner zugrundeliegenden Gleitkommarasters $S(b, n)$ sind. Man beachte jedoch, daß ein Maschinenintervall sämtliche reellen Werte zwischen den Endpunkten repräsentiert, d.h. also nach wie vor das gesamte Kontinuum abdeckt.

Bei der Verknüpfung der Maschinenintervalle können die Endpunkte mit Hilfe von Gleitkommaoperationen mit gerichteter Rundung optimal berechnet werden. So kann die Einschließungseigenschaft der Intervallrechnung aufrecht erhalten werden. Dies sollte folgendermassen geschehen:

$$\begin{aligned}
A \boxplus B &= [\underline{a} \bigtriangledown \underline{b}, \overline{a} \bigtriangleup \overline{b}] \\
A \boxminus B &= [\underline{a} \bigtriangledown \overline{b}, \overline{a} \bigtriangleup \underline{b}] \\
A \boxdot B &= [\min\{\underline{a} \bigtriangledown \underline{b}, \underline{a} \bigtriangledown \overline{b}, \overline{a} \bigtriangledown \underline{b}, \overline{a} \bigtriangledown \overline{b}\}, \max\{\overline{a} \bigtriangleup \overline{b}, \underline{a} \bigtriangleup \overline{b}, \overline{a} \bigtriangleup \underline{b}, \overline{a} \bigtriangleup \overline{b}\}] \\
A \boxslash B &= [\underline{a}, \overline{a}] \boxdot [1 \bigtriangledown \overline{b}, 1 \bigtriangleup \underline{b}] \text{ für } 0 \notin B.
\end{aligned}$$

Weitere Einzelheiten und zusätzliche Erläuterungen können z.B. Alefeld/Herzberger [2] entnommen werden.

Die benötigten Operationen mit gerichteten Rundungen werden im IEEE-Standard 754 [4] gefordert und werden heute von vielen Prozessoren zur Verfügung gestellt. Allerdings sind diese Operationen in der ANSI-Norm für die Programmiersprache C [3, 22] nicht vorgesehen und werden von den üblichen C- und C++ - Compilern auch nicht zur Verfügung gestellt.

Aus diesem Grund setzen wir im folgenden nur voraus, daß von Rechner und Compiler ein Datenformat `double` verwendet wird, daß dem im IEEE-Standard 754 beschriebenen Format (Darstellung einer Gleitkommazahl mit 64 Bit, Mantissenlänge 53 Bit) entspricht. Für die Verknüpfung zweier Gleitkommazahlen wird ebenfalls vorausgesetzt, daß sie dem IEEE-Standard 754 entspricht. Dies bedeutet insbesondere, daß für alle Rundungsmodi die folgende Beziehung gilt:

$$\bigwedge_{\circ \in \{+,-,\cdot,/\}} \quad \bigwedge_{\substack{a,b \in S \text{ mit} \\ |a \circ b| \in [\texttt{MinReal},\texttt{MaxReal}]}} \left| \frac{a \circ b - a \boxed{\circ} b}{a \circ b} \right| \leq 2 \cdot \varepsilon^* \; . \tag{10.1}$$

Dabei bezeichnet ε^* die relative Maschinengenauigkeit. Für das Gleitkommaraster $S(b,l)$ gilt $\varepsilon^* := \frac{1}{2}2^{1-l}$.

Diese Voraussetzungen werden heute von fast allen Rechnern und Compilern erfüllt. Insbesondere ist es bei dieser Wahl der Voraussetzungen gleichgültig, ob der Compiler den Rundungsmodus so setzt, daß das Ergebnis einer Gleitkommaverknüpfung zur nächstgelegenen Maschinenzahl gerundet wird, oder ob der Compiler den Rundungsmodus überhaupt nicht steuert und einfach den Zufall überläßt. Im letzten Fall ist bei einem IEEE-konformen Prozessor dann einer der folgenden Modi aktiv: Rundung zur nächstgelegenen Zahl, Rundung in Richtung $-\infty$, Rundung in Richtung ∞ oder Rundung zur 0.

10.2 Implementierung in ANSI-C

Es müssen zunächst zwei Hilfsfunktionen pred(x) und succ(x) bereitgestellt werden. Die Funktion pred(x) liefert den Vorgänger einer Double-Zahl x im Gleitkommaraster, die Funktion succ(x) liefert entsprechend den Nachfolger.

Zur Speicherung von Intervallen wird der neue Datentyp `a_intv` mit zwei Komponenten `INF` und `SUP` bereitgestellt:

```
typedef struct a_intv { double INF, SUP; } a_intv;
```

Der Namen dieses Datentyps wurde aus Kompabilitätsgründen mit dem Laufzeitsystem von Pascal–XSC so gewählt. Prinzipiell kann der Name dieses Datentyps mit einem zur Verfügung gestellten Preprozessor im Quelltext abgeaendert werden. Mit dieser Methode wird z.B. in der Version für den FTP-Server dem Benutzer folgender Typname vorgeben:

```
typedef struct interval { double INF, SUP; } interval;
```

Mit diesem Namen, der auch in dem bereitgestellten C++ – Interface verwendet wird, ist sicher eine eingängigere Programmierung möglich.

Zusammen mit den Voraussetzungen des vorhergehenden Abschnittes läßt sich nun eine einfache Intervallgrundarithmetik wie folgt realisieren:

$$
\begin{aligned}
A \boxplus B &= [\mathrm{pred}(\underline{a} \boxplus \underline{b}), \mathrm{succ}(\overline{a} \boxplus \overline{b})] \\
A \boxminus B &= [\mathrm{pred}(\underline{a} \boxminus \overline{b}), \mathrm{succ}(\overline{a} \boxminus \underline{b})] \\
A \boxdot B &= [\mathrm{pred}(\min\{\underline{a} \boxdot \underline{b}, \underline{a} \boxdot \overline{b}, \overline{a} \boxdot \underline{b}, \overline{a} \boxdot \overline{b}\}), \\
&\qquad \mathrm{succ}(\max\{\overline{a} \boxdot \overline{b}, \underline{a} \boxdot \overline{b}, \overline{a} \boxdot \underline{b}, \overline{a} \boxdot \overline{b}\})] \\
A \boxslash B &= [\mathrm{pred}(\min\{\underline{a} \boxslash \underline{b}, \underline{a} \boxslash \overline{b}, \overline{a} \boxslash \underline{b}, \overline{a} \boxslash \overline{b}\}), \\
&\qquad \mathrm{succ}(\max\{\overline{a} \boxslash \overline{b}, \underline{a} \boxslash \overline{b}, \overline{a} \boxslash \underline{b}, \overline{a} \boxslash \overline{b}\})]
\end{aligned}
$$

Die Operationen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$ zweier Gleitkommazahlen können dabei mit einem der im IEEE-Standard beschrieben Rundungsmodi ausgeführt werden.

Damit bei der späteren Verwendung dieser Intervallarithmetik Verifikationsverfahren erfolgreich ablaufen können, müssen bei der Implementierung einige Sonderfälle (z.B. richtige Vorzeichenerkennung) beachtet werden. Nähere Details können dem C-Quellcode im File `q_ari.c` entnommen werden.

Die beschriebenen Operationen zur Verknüpfung zweier Intervalle werden in der Bibliothek `fi_lib.a` zur Verfügung gestellt. Zusätzlich werden in dieser Bibliothek Verknüpfungen für ein Intervall mit einer Double-Zahl bereitgestellt.

Da in ANSI-C kein Operatorkonzept verfügbar ist, mußten die Operationen als Funktionen bereitgestellt werden. Dies führt natürlich im Anwendungsprogramm zu etwas unübersichtlichen Funktionsaufrufen (siehe das nachfolgende Beispiel). Aus diesem Grund wurde versucht, die Namensgebung möglichst einfach zu gestalten. Die Funktionen beginnen mit einer Namensabkürzung für die Operation (`add`, `sub`, `mult` oder `div`), gefolgt von einem Strich (`_`) und zwei Buchstaben (`i` oder `d`), die den Datentyp der Operanden (`a_intv` oder `double`) angeben. Z.B. heißt die Funktion zur Addition zweier Intervalle `add_ii`, die Funktion zur Multiplikation eines Intervalls als erster Operand mit einer Double-Zahl als zweiter Operand heißt `mul_id`.

Die folgende Intervalloperationen werden in `fi_ari.h` für den Intervalldatentyp

```
typedef struct a_intv { double INF, SUP;} a_intv ;
```

zur Verfügung gestellt (evtl. mit Hilfe eines Preprozessors in `interval` abgeändert):

Funktionsdeklaration	**Bedeutung**	
`a_intv add_ii(a_intv x,a_intv y)`	addiert zwei Intervalle x und y	**Addition**
`a_intv add_di(double x,a_intv y)`	addiert `double`-Zahl x zu Intervall y	
`a_intv add_id(a_intv x,double y)`	addiert Intervall x zu `double`-Zahl y	
`a_intv sub_ii(a_intv x,a_intv y)`	subtrahiert Intervall y von Intervall x	**Subtraktion**
`a_intv sub_di(double x,a_intv y)`	subtrahiert von `double`-Zahl x das Intervall y	
`a_intv sub_id(a_intv x,double y)`	subtrahiert von Intervall x die `double`-Zahl y	
`a_intv mul_ii(a_intv x,a_intv y)`	multipliziert zwei Intervalle x und y	**Multiplikation**
`a_intv mul_di(double x,a_intv y)`	multipliziert `double`-Zahl x mit Intervall y	
`a_intv mul_id(a_intv x,double y)`	multipliziert Intervall x mit `double`-Zahl y	
`a_intv div_ii(a_intv x,a_intv y)`	dividiert Intervall x durch Intervall y	**Division**
`a_intv div_di(double x,a_intv y)`	dividiert `double`-Zahl x durch Intervall y	
`a_intv div_id(a_intv x,double y)`	dividiert Intervall x durch `double`-Zahl y	
`a_intv eq_ii(a_intv y)`	Zuweisungsoperator für Intervalle (kann auch weggelassen werden)	**Zuweisungen**
`a_intv eq_di(double y)`	`double`-Zahl wird einem Punktintervall zugewiesen	
`int in_ii(a_intv x,a_intv y)`	liefert 1, falls Intervall $x \subset$ Intervall y, sonst 0	**logische Operatoren**
`int in_di(double x,a_intv y)`	liefert 1, falls $x \in y$, sonst 0	
`int ieq_ii(a_intv x,a_intv y)`	liefert 1, falls beide Intervalle identisch, sonst 0	
`int is_ii(a_intv x,a_intv y)`	liefert 1, falls $x.INF < y.INF$ und $x.SUP < y.SUP$, sonst 0	
`int ig_ii(a_intv x,a_intv y)`	liefert 1, falls $x.INF > y.INF$ und $x.SUP > y.SUP$, sonst 0	
`int ise_ii(a_intv x,a_intv y)`	liefert 1, falls $x.INF \leq y.INF$ und $x.SUP \leq y.SUP$, sonst 0	
`int ige_ii(a_intv x,a_intv y)`	liefert 1, falls $x.INF \geq y.INF$ und $x.SUP \geq y.SUP$, sonst 0	
`int dis_ii(a_intv x,a_intv y)`	liefert 1, falls $x \cap y = \emptyset$, sonst 0	
`double t_mid(a_intv x)`	liefert Mittelpunkt des Intervalls	**sonstige Funktionen**
`double t_diam(a_intv x)`	liefert Durchmesser des Intervalls	
`a_intv hull(a_intv x,a_intv y)`	liefert konvexe Hülle der Intervalle x und y	
`a_intv intsec(a_intv x,a_intv y)`	liefert das Intervall $x \cap y$, falls es existiert	

Kapitel 11

Bereitgestellte Versionen

Es hat sich herausgestellt, daß für die verschiedenen, vorhandenen Anwendungswünsche unterschiedliche Versionen der Bibliothek benötigt werden. Um bei späteren Änderungen und Korrekturen die Fehleranfälligkeit zu minimieren, wurde eine sogenannte Urversion implementiert (Ursprungsquelltext) aus der dann (größtenteils automatisch) mit Hilfe von Skripten und eines Preprozessors die anderen benötigten Versionen generiert werden können. Änderungen an der Bibliothek sollten damit immer an der Urversion erfolgen. Hinweise zur Generierung der abgeleiteten Versionen können dem Anhang entnommen werden.

11.1 ANSI-C Version

11.1.1 Implementierung der Bibliothek

Die ANSI-C Version benötigt keine Funktionen aus dem Pascal-XSC Laufzeitsystems. Zum Übersetzen muß ein eigenes Headerfile `o_defs.h` verwendet werden.

Die Headerfiles `q_defs.h`, `q_fcth.h` und `q_errm.h` unterscheiden sich in einigen Punkten von denen der Pascal-XSC Version. Da nicht auf die Funktionen für das Fehlerhandling von Pascal-XSC zurückgegriffen werden kann, werden die Ausgabe der Fehlermeldung, sowie eine Funktion zum Programmabruch im File `q_errm.c` zur Verfügung gestellt. Der Benutzer kann in diesem File sehr leicht verändern, wie sich seine Programme im Fehlerfall verhalten sollen (siehe auch Kapitel 12).

Zusätzlich steht für diese Version eine einfache und schnelle Intervallgrundarithmetik zur Verfügung. Der hierfür benötigte Quellcode befindet sich in den Files `fi_lib.h` und `fi_lib.c`.

11.1.2 ANSI-C Version für den FTP-Server

Zur Bereitstellung auf dem FTP-Server wurde die oben beschriebene ANSI-C Version modifiziert. Befehle, die in dieser Version nicht benötigt werden, wurden mit Hilfe eines Preprozessors entfernt. Dies betrifft insbesondere die Fehlerbehandlung (vergleiche Kapitel 12). Weiter wurden an den Anfang jeder Datei Kommentarzeilen mit der Versionsnummer, Hinweisen zum Copyright usw. eingefügt. Alle benötigten Headerfiles wurden

zu einer Datei names `fi_lib.h` zusammengefasst. Der in der ursprünglichen ANSI-C Version aus Kompatibilitätsgründen gewählte Name `a_intv` für den Intervalldatentyp wurde in `interval` umbenannt.

Ziel dieser Modifikationen war, eine größere Übersichtlichkeit im Programmcode zu erreichen und die Benutzung der Bibliothek zu erleichtern.

Zusätzlich wurde diese Version um eine einfach Intervallgrundarithmetik (siehe Kapitel 10) und erste Anwendungsbeispiele (siehe Anhang D) ergäenzt.

Die erste Version wurde am 20.8.1997 zur freien Benutzung zur Verfügung gestellt und ist unter der Adresse

```
iamk4515.mathematik.uni-karlsruhe.de
```

im Verzeichnis

```
pub\iwrmm\software
```

per „anonymous“ FTP erhältlich. Alternativ kann auch im WWW über die Seiten des Institus für Angewandte Mathematik (IAM) bzw. die Seiten des Instituts für Wissenschaftliches Rechnen und Mathematische Modellbildung (IWRMM)

```
http://www.uni-karlsruhe.de/~iam/lehrstuhl2.html
http://www.mathematik.uni-karlsruhe.de/~iwrmm
```

auf dem FTP-Server zugegriffen werden.

11.2 Version mit C++ Schnittstelle

Die ANSI-C Version für den FTP-Server wurde um eine C++ Schnittstelle erweitert. Durch die in C++ vorgesehenen Konzepte der Funktionsüberladung und der Operatordefinition kann die Benutzung der Bibliothek im Anwendungsprogramm wesentlich vereinfacht werden. Dies wird auf eindrucksvolle Weise durch die ANSI-C und C++ Beispielprogramm im Anhang D belegt. Das bereitgestellte Headerfile mit den entsprechenden Definitionen und Deklarationen ist im Anhang C abgedruckt.

11.3 Pascal–XSC Versionen

In diesen Versionen werden Funktionen des Laufzeitsystem von Pascal–XSC mitbenutzt. So werden z.B. für die Bildung des Vorgängers bzw. Nachfolgers einer Gleitkommazahl die in Pascal–XSC bereits vorhanden Funktionen benutzt oder im Fehlerfall wird das Fehlerhandling des Laufzeitsystems aufgerufen.

11.3.1 Modul ff_ari zu Pascal–XSC

Um auch in den älteren Versionen von Pascal–XSC das Arbeiten mit den neuen Standardfunktionen zu ermöglichen, wurde ein Modul namens `ff_ari.p` erstellt. Nach dem Einbinden dieses Moduls in ein Anwendungsprogramm mit dem Kommando `use` können die neuen Funktionen verwendet werden. Beim Aufruf muß vor den bekannten Funktionsnamen ein `ff` davorgestellt werden, also z.B. `ffexp` statt `exp` für den Aufruf der

Exponentialfunktion. Ein Vorteil dieser Methode ist, daß die neuen und die bisherigen Standardfunktionen in einem Programm gleichzeitig verwendet werden können. Auf diese Weise ist auch ein sehr einfaches Testen der neuen Funktionen durch Vergleich mit den bisherigen Funktionen möglich.

11.3.2 Version für Laufzeitsystem

In die neueren Versionen des Laufzeitsystems von Pascal–XSC wurden die Standardfunktionen direkt integriert (Quellcode der Dateien `q_????.c` und `j_????.c`). Bei Aufruf einer Standardfunktion in einem Programm werden dann automatisch die neuen schnellen Funktonen verwendet.

11.4 C–XSC Version

Bei einem bereits geplanten Redesign von C–XSC sollen die neuen Funktionen auch hier eingebunden werden. Hierzu soll die ANSI-C Version verwendet werden.

11.5 Fortran–XSC Version

In Zusammenarbeit mit Prof. Dr. W. Walter (Universität Dresden) und Dr. R. Lohner (Universität Karlsruhe) wurde eine Version der Bibliothek in das neu entstehende Fortran–XSC integriert.

Als Schnittstelle zwischen Fortran und ANSI-C wurde hierzu ein Interface `xscstd.f9p` als Fortran Modul von Dr. R. Lohner entwickelt. An der Bibliothek selbst mußten mehrere Änderungen durchgeführt werden. So wurden alle Funktions- und Dateinamen umbenannt.

ANSI-C	Fortran–XSC
q_acos	racos
q_acot	racot
q_acsh	racosh
q_acth	racoth
q_asin	rasin
q_asnh	rasinh
q_atan	ratan
q_atn1	ratn1
q_atnh	ratanh
q_cos	rcos
q_cos1	rcos1
q_cosh	rcosh
q_cot	rcot
q_coth	rcoth
q_cth1	rcth1
q_ep1	rep1
q_epm1	repm1
q_ex10	rexp10
q_exp	rexp
q_exp2	rexp2
q_expm	rexpm1
q_lg10	rlog10
q_lg1p	rlog1p
q_log	rlog
q_log1	rlog1
q_log2	rlog2
q_sin	rsin
q_sin1	rsin1
q_sinh	rsinh
q_sqr	rsqr
q_sqrt	rsqrt
q_tan	rtan
q_tanh	rtanh

ANSI-C	Fortran–XSC
j_acos	iacos
j_acot	iacot
j_acsh	iacosh
j_acth	iacoth
j_asin	iasin
j_asnh	iasinh
j_atan	iatan
j_atnh	iatanh
j_cos	icos
j_cosh	icosh
j_cot	icot
j_coth	icoth
j_ex10	iexp10
j_exp	iexp
j_exp2	iexp2
j_expm	iexpm1
j_lg10	ilog10
j_lg1p	ilog1p
j_log	ilog
j_log2	ilog2
j_sin	isin
j_sinh	isinh
j_sqr	isqr
j_sqrt	isqrt
j_tan	itan
j_tanh	itanh

Der Intervalldatentyp `a_intv` wurde in `interval` umbenannt. Alle benötigten Deklarationen wurden in einem Headerfile zusammengefaßt, die entsprechenden `#include` Anweisungen in den einzelnen Dateien angepaßt.

Weiter mußten Aufgrund der unterschiedlichen Übergabemechanismen des verwendeten C–Compilers und Fortran–Compilers bei der Argumentübergabe die Parameterlisten abgeändert werden. Dabei wurden alle Wertaufrufe durch Referenzaufrufe ersetzt.

Die Umbenennungen können mit Hilfe eines Skripts automatisch durchgeführt werden. Die Änderungen in der Parameterliste der einzelnen Funktionen kann zur Zeit nicht vollautomatisch ausgeführt werden, hier ist bei den einzelnen Funktionen ein manuelles Eingreifen erforderlich.

Für die Fortran–XSC Version wurden zusätzlich spezielle Testprogramme erstellt bzw. angepaßt, damit bei einer Übertragung auf eine andere Rechnerplattform die Korrektheit der Bibliothek überprüft werden kann.

Kapitel 12

Fehlerbehandlung („error handling“)

12.1 Allgemeine Bemerkungen und Hinweise

Da die Funktionsbibliothek einerseits eigenständig verfügbar sein soll (ANSI–C), andererseits aber auch gemeinsam mit bestehenden Laufzeitsystemen (Pascal–XSC, C–XSC, Fortran–XSC) verwendet werden soll, mußten Wahlmöglichkeiten für verschiedene Arten der Fehlerbehandlung (error handling) vorgesehen werden.

Aus diesem Grund wurden die eigentlichen Funktionen im Ursprungsquelltext in den folgenden Programmrahmen integriert:

```
...   /* Einbinden von Headerfiles, Kommentare usw. */

#ifdef LINT_ARGS
local double q_name(double x)
#else
local double q_name(x)

double x;
#endif
{

#ifdef PXSCTRAP
  E_SPUSH("q_name");
#endif

...  /* Eigentlicher Quellcode der Funktion */

#ifdef PXSCTRAP
  E_SPOPP("q_sqrt");
#endif

}
```

Bei gemeinsamer Verwendung der Bibliothek mit dem Laufzeitsystem von Pascal–XSC kann mit Hilfe des Preprozessors durch Setzen von `PXSCTRAP` erreicht werden, daß zu Beginn einer Funktion der Funktionsname auf einen Stapelspeicher gelegt wird. Dadurch ist im Fehlerfall eine Rückverfolgung der Fehlerursache möglich. Nach dem korrekten

Verlassen einer Funktion (d.h. es trat während der Abarbeitung kein Fehler auf) wird der entsprechende Funktionsname wieder vom Stapelspeicher entfernt. Zur Manipulation des Stapelspeichers werden die Routinen `E_SPUSH` und `E_SPOPP` aus dem Laufzeitsystem von Pascal–XSC verwendet.

Falls der oben genannte Stapelspeicher und damit die Rückverfolgung der Fehlerursache nicht benutzt wird, besteht die Möglichkeit, die entsprechenden Befehle aus dem Ursprungsquelltext mittels eines eigenen Preprozessors zu entfernen. Der Programmtext wird hierdurch übersichtlicher und einfacher lesbar. Bei der Bibliotheksversion, die per FTP-Server zur Verfügung gestellt wird (ANSI–C Version) wird dies verwendet.

12.2 ANSI–C Version

Es wurde bewußt nur eine einfache Art der Fehlerbehandlung implementiert. Dem Benutzer wird dadurch auch die Möglichkeit gegeben, die Fehlerbehandlung an seine Bedürfnisse anzupassen. Änderungen müssen in diesem Fall in den beiden Dateien `q_errm.h` und `q_errm.c` durchgeführt werden.

Die implementierten Standardfunktionen rufen im Fehlerfall eine der folgenden Funktionen auf:

```
double q_abortnan(int n, double *x, int fctn)
double q_abortr1(int n, double *x, int fctn)
a_intv q_abortr2(int n, double *x1, double *x2, int fctn)
```

Beim Aufruf wird eine „Funktionsnummer“ `fctn` mit übergeben. Anhand dieser Funktionsnummer läßt sich identifizieren, in welcher Standardfunktion der Fehler aufgetreten ist. Die Zuordnung dieser Funktionsnummern kann der folgenden Tabelle entnommen werden.

Nummer	Name	Funktion
0	`q_sqrt`	Wurzelfunktion
1	`q_sqr`	Quadratfunktion
2	`q_exp`	Exponentialfunktion
3	`q_expm`	Exponentialfkt.-minus-Eins
4	`q_exp2`	Exponentialfkt. Basis 2
5	`q_ex10`	Exponentialfkt. Basis 10
6	`q_log`	Logarithmusfunktion
7	`q_lg1p`	Logarithmusfkt. von $(x+1)$
8	`q_log2`	Logarithmusfkt. Basis 2
9	`q_lg10`	Logarithmusfkt. Basis 10
10	`q_sin`	Sinusfunktion
11	`q_cos`	Cosinusfunktion
12	`q_tan`	Tangensfunktion
13	`q_cot`	Cotangensfunktion

Nummer	Name	Funktion
14	`q_asin`	Arkussinus
15	`q_acos`	Arkuscosinus
16	`q_atan`	Arkustangens
17	`q_actn`	Arkuscotangens
18	`q_sinh`	Sinus hyperbolicus
19	`q_cosh`	Cosinus hyperbolicus
20	`q_tanh`	Tangens hyperbolicus
21	`q_coth`	Cotanges hyperbolicus
22	`q_asnh`	Areasinus
23	`q_acsh`	Areacosinus
24	`q_atnh`	Areatangens
25	`q_acth`	Areacotangens
26	`q_comp`	Mant., Expo., Vz.

Für die Intervallfunktionen `j_????` werden die gleichen Funktionsnummern verwendet, wie für die zugehörigen Punktfunktionen `q_????`.

Die Fehlerart wird durch den Parameter `n` bestimmt. Die folgenden Fehlerarten werden in der Datei `o_defs.h` definiert und verwendet:

Nummer	Name	Bedeutung
1	INV_ARG	Unzulässiges Eingabeargument
2	OVER_FLOW	Ergebnis im Überlaufbereich

12.3 Übersicht Fehlerverhalten

Im folgenden ist eine Tabelle angegeben, aus der das Verhalten bei den wichtigsten Fehlerursachen für die elementaren Punktfunktionen entnommen werden kann.

Funktion	sNaN/qNaN	$-\infty$	∞	$x \notin D$
sqr	Error: NaN	OVER_FLOW	OVER_FLOW	OVER_FLOW
sqrt	Error: NaN	INV_ARG	∞	INV_ARG
exp	Error: NaN	0	OVER_FLOW	OVER_FLOW
expm1	Error: NaN	-1	OVER_FLOW	OVER_FLOW
log	Error: NaN	INV_ARG	INV_ARG	INV_ARG
log1p	Error: NaN	INV_ARG	INV_ARG	INV_ARG
sinh	Error: NaN	OVER_FLOW	OVER_FLOW	OVER_FLOW
cosh	Error: NaN	OVER_FLOW	OVER_FLOW	OVER_FLOW
tanh	Error: NaN	-1	$+1$	INV_ARG
coth	Error: NaN	-1	$+1$	INV_ARG
arsinh	Error: NaN	INV_ARG	INV_ARG	INV_ARG
arcosh	Error: NaN	INV_ARG	INV_ARG	INV_ARG
artanh	Error: NaN	INV_ARG	INV_ARG	INV_ARG
arcoth	Error: NaN	INV_ARG	INV_ARG	INV_ARG
sin	Error: NaN	INV_ARG	INV_ARG	INV_ARG
cos	Error: NaN	INV_ARG	INV_ARG	INV_ARG
tan	Error: NaN	INV_ARG	INV_ARG	INV_ARG
cot	Error: NaN	INV_ARG	INV_ARG	INV_ARG
arcsin	Error: NaN	INV_ARG	INV_ARG	INV_ARG
arccos	Error: NaN	INV_ARG	INV_ARG	INV_ARG
arctan	Error: NaN	INV_ARG	INV_ARG	INV_ARG
arccot	Error: NaN	INV_ARG	INV_ARG	INV_ARG
exp2	Error: NaN	0	OVER_FLOW	OVER_FLOW
exp10	Error: NaN	0	OVER_FLOW	OVER_FLOW
log2	Error: NaN	INV_ARG	INV_ARG	INV_ARG
log10	Error: NaN	INV_ARG	INV_ARG	INV_ARG

Der Tabelleneintrag „Error: NaN“ bedeutet, daß das Programm abgebrochen wird. In der vorhandenen Implementierung wird dabei nicht zwischen sNaN und qNaN („signaling“

und „quiet“ NaN) unterschieden. Diese Unterscheidung und ein daraus resultierendes unterschiedliches Verhalten kann allerdings leicht in den Quellcode der beiden Dateien `q_errm.h` und `q_errm.c` eingebaut werden.

Kapitel 13

Tests und Laufzeitvergleiche

13.1 Testen der Standardfunktionen

Zunächst wurden die Routinen der neuen Bibliothek `fi_lib` gegen bestehende XSC–Versionen und gegen Langzahlintervallversionen (unter Verwendung von `mpi_ari`) getestet (Inklusionstest). Siehe hierzu auch Abschnitt F.3.

Desweiteren wurde ein Testprogramm (z.B. ANSI–C Version: `fi_test.c`) zum Überprüfen der Korrektheit einer Installation entwickelt.

13.2 Laufzeitmessungen

Aufgrund der beobachteten starken Schwankungen bei den Laufzeitmessungen sollten die folgenden Tabellen nur als Anhaltspunkt interpretiert werden. Die Tests wurden mit der Pascal-XSC Version `ff_ari` realisiert (siehe Abschnitt 11.3.1).

13.2.1 Pascal-XSC Version auf PC unter LINUX

```
Geschwindigkeit der ff-Funktionen (Punktversionen)
Mit wieviel Zufallszahlen ? 100

Fkt-Name        ff_ari       P-XSC       Faktor
-----------------------------------------------
exp             0.2794      69.9000     250.1790
exp2            0.2754      72.6000     263.6166
exp10           0.2731      72.3000     264.7382
ln              0.5636      52.1000      92.4414
log2            0.8163      54.3000      66.5197
log10           0.8298      55.8000      67.2451
sin             0.4403     138.5000     314.5583
cos             0.6607     175.5000     265.6274
tan             0.8092     100.0000     123.5788
cot             0.8256      99.7000     120.7607
arcsin          0.3537      90.2000     255.0184
```

```
arccos          0.3390      95.9000     282.8909
arctan          0.2832      34.0000     120.0565
arccot          0.3127      46.6000     149.0246
sinh            0.2696      76.6000     284.1246
cosh            0.4549      72.0000     158.2765
tanh            0.3007      79.4000     264.0505
coth            0.6183      74.3000     120.1682
arsinh          0.6057      82.3000     135.8758
arcosh          0.8137     104.3000     128.1799
artanh          0.4339      83.4000     192.2102
arcoth          0.7022      56.4000      80.3190
expm1           0.2828      72.9000     257.7793
ln1p            0.3008      16.2000      53.8564
sqrt            0.3277      54.5000     166.3106

Geschwindigkeit der ff-Funktionen (Intervallversionen mit Punktintervallen)
Mit wieviel Zufallszahlen ? 100

Fkt-Name        ff_ari       P-XSC       Faktor
-----------------------------------------------
exp             0.4452     151.2000     339.6226
exp2            0.4601     161.1000     350.1413
exp10           0.4906     160.5000     327.1504
ln              0.6616     116.0000     175.3325
log2            0.9047     124.2000     137.2831
log10           0.9409     129.1000     137.2091
sin             0.6523     272.7000     418.0592
cos             0.8874     331.6000     373.6759
tan             1.2839     195.0000     151.8810
cot             1.1845     197.4000     166.6526
arcsin          0.2580     182.3000     706.5891
arccos          0.3815     195.4000     512.1887
arctan          0.2445      76.4000     312.4744
arccot          0.3517     103.8000     295.1379
sinh            0.3767     156.9000     416.5118
cosh            0.5533     149.5000     270.1970
tanh            0.4164     170.8000     410.1825
coth            0.7107     157.9000     222.1753
arsinh          0.7279     173.1000     237.8074
arcosh          0.8530     213.4000     250.1758
artanh          0.6111     166.5000     272.4595
arcoth          0.9057     121.6000     134.2608
expm1           0.4088     160.0000     391.3894
ln1p            0.3892      72.1000     185.2518
sqrt            0.8388      57.6000      68.6695

Geschwindigkeit der ff-Funktionen (Intervallversionen)
Mit wieviel Zufallszahlen ? 100
```

```
Fkt-Name        ff_ari       P-XSC       Faktor
-------------------------------------------------

exp             0.7452     148.8000     199.6779
exp2            0.8243     159.6000     193.6188
exp10           0.8828     159.7000     180.9017
ln              1.2616     117.3000      92.9772
log2            1.7449     124.0000      71.0642
log10           1.7341     123.1000      70.9878
sin             1.4352     261.3000     182.0652
cos             1.5748     312.9000     198.6919
tan             2.3081     194.1000      84.0951
cot             2.5268     196.2000      77.6476
arcsin          0.3431     190.6000     555.5232
arccos          0.7800     201.3000     258.0769
arctan          0.3113      77.7000     249.5985
arccot          0.6926     102.9000     148.5706
sinh            0.7395     158.0000     213.6579
cosh            1.0933     151.7000     138.7542
tanh            0.7313     160.1000     218.9252
coth            1.3272     151.3000     113.9994
arsinh          1.3961     174.7000     125.1343
arcosh          1.5687     211.8000     135.0163
artanh          1.1693     170.9000     146.1558
arcoth          1.6068     117.2000      72.9400
expm1           0.8230     158.3000     192.3451
ln1p            0.7329      68.8000      93.8737
sqrt            1.2515     106.7000      85.2577
```

13.2.2 Pascal-XSC Version auf SUN-Workstation

```
Geschwindigkeit der ff-Funktionen (Punktversionen)
Mit wieviel Zufallszahlen ? 100

Fkt-Name        ff_ari       P-XSC       Faktor
-------------------------------------------------
exp             0.4460     120.3000     269.7309
exp2            0.4444     125.8000     283.0783
exp10           0.4379     126.6000     289.1071
ln              1.1155      89.8000      80.5020
log2            1.6150      93.0000      57.5851
log10           1.6314      95.3000      58.4161
sin             0.9230     234.9000     254.4962
cos             1.1086     293.1000     264.3875
tan             0.9337     167.2000     179.0725
cot             1.2697     166.9000     131.4484
arcsin          0.6135     155.2000     252.9747
arccos          0.4804     168.1000     349.9167
```

```
arctan          0.5187      59.8000     115.2882
arccot          0.4442      82.7000     186.1774
sinh            0.4497     119.5000     265.7327
cosh            0.8874     110.1000     124.0703
tanh            0.7432     128.3000     172.6319
coth            0.6590     116.6000     176.9347
arsinh          0.8278     149.7000     180.8408
arcosh          0.9994     184.0000     184.1105
artanh          0.7819     140.8000     180.0742
arcoth          1.0526      98.6000      93.6728
expm1           0.7513     122.4000     162.9176
ln1p            0.5514      24.9000      45.1578
sqrt            0.7703      97.9000     127.0933

Geschwindigkeit der ff-Funktionen (Intervallversionen)
Mit wieviel Zufallszahlen ? 100

Fkt-Name        ff_ari       P-XSC       Faktor
-----------------------------------------------
exp             1.6990     252.9000     148.8523
exp2            1.1413     269.8000     236.3971
exp10           1.7972     273.9000     152.4037
ln              1.6925     189.3000     111.8464
log2            3.4329     206.6000      60.1824
log10           2.5904     205.8000      79.4472
sin             1.3631     384.8000     282.2977
cos             2.2152     443.9000     200.3882
tan             0.8836     293.3000     331.9375
cot             3.0283     297.5000      98.2399
arcsin          0.4414     314.5000     712.5057
arccos          1.1325     342.2000     302.1634
arctan          0.4120     128.3000     311.4078
arccot          1.0601     176.7000     166.6824
sinh            1.3475     246.3000     182.7829
cosh            1.4725     227.7000     154.6350
tanh            1.5304     258.3000     168.7794
coth            2.1322     237.4000     111.3404
arsinh          1.9517     291.5000     149.3570
arcosh          2.0986     364.3000     173.5919
artanh          1.8760     277.0000     147.6546
arcoth          2.1742     192.9000      88.7223
expm1           1.4311     263.2000     183.9145
ln1p            1.5785     110.9000      70.2566
sqrt            1.8696     187.9000     100.5028
```

13.2.3 Pascal-XSC Version auf der SP/2

```
Geschwindigkeit der ff-Funktionen (Punktversionen)
```

```
Mit wieviel Zufallszahlen ? 100

Fkt-Name        ff_ari       P-XSC       Faktor
-----------------------------------------------
exp             0.4300      36.9000      85.8140
exp2            0.4311      38.2000      88.6105
exp10           0.4257      38.2000      89.7346
ln              0.5413      28.0000      51.7273
log2            0.9485      28.4000      29.9420
log10           0.9460      28.0000      29.5983
sin             0.5793      73.7000     127.2225
cos             0.5901      90.8000     153.8722
tan             0.5871      50.6000      86.1863
cot             0.6307      51.1000      81.0211
arcsin          0.4632      44.0000      94.9914
arccos          0.4602      47.4000     102.9987
arctan          0.4289      18.2000      42.4341
arccot          0.4417      24.4000      55.2411
sinh            0.4430      38.2000      86.2302
cosh            0.5137      36.3000      70.6638
tanh            0.4442      41.3000      92.9761
coth            0.4965      37.5000      75.5287
arsinh          0.6249      43.9000      70.2512
arcosh          0.6739      52.3000      77.6080
artanh          0.5336      43.2000      80.9595
arcoth          0.6396      29.8000      46.5916
expm1           0.4257      37.3000      87.6204
ln1p            0.4578       8.6000      18.7855
sqrt            0.4620      24.9000      53.8961

Geschwindigkeit der ff-Funktionen (Intervallversionen mit Punktintervallen)
Mit wieviel Zufallszahlen ? 100

Fkt-Name        ff_ari       P-XSC       Faktor
-----------------------------------------------
exp             0.5404      77.2000     142.8571
exp2            0.5613      82.9000     147.6929
exp10           0.5619      81.2000     144.5097
ln              0.6096      60.0000      98.4252
log2            1.0133      64.4000      63.5547
log10           1.0107      63.4000      62.7288
sin             0.3099     137.6000     444.0142
cos             0.7233     166.0000     229.5037
tan             0.7854      98.0000     124.7772
cot             1.0813      98.5000      91.0941
arcsin          0.2304      89.7000     389.3229
arccos          0.5150      96.6000     187.5728
arctan          0.2200      40.3000     183.1818
```

arccot	0.5039	52.2000	103.5920
sinh	0.5217	79.7000	152.7698
cosh	0.5951	75.5000	126.8694
tanh	0.5428	84.1000	154.9374
coth	0.5637	78.1000	138.5489
arsinh	0.8106	91.8000	113.2494
arcosh	0.7528	105.0000	139.4793
artanh	0.6275	90.6000	144.3825
arcoth	0.7159	62.6000	87.4424
expm1	0.5095	79.6000	156.2316
ln1p	0.5289	35.8000	67.6877
sqrt	0.7957	26.8000	33.6810

Anhang A

Tabellen und Daten zu den Standardfunktionen

A.1 Verzeichnisstruktur der Urversion

Alle im Verlauf dieses Projekts entstandenen Programme und Dateien sind in der gepackten Datei `ff.tgz` enthalten. Beim Entpacken mit

```
gunzip ff.tgz
tar xf ff.tar
```

wird die folgende Verzeichnisstruktur (beginnend im aktuellen Verzeichnis) angelegt:

```
Verzeichnisstruktur der Urversion:
----------------------------------

fastfunc
   |
   |- ansi_c              Verzeichnis der ANSI-C Version
   |      |
   |      |- compile      Batch-Files ('compall.cc','compall.gcc', ...) zum
   |      |               Erzeugen der Bibliothek
   |      |- hp           Enthaelt uebersetzte Version der Bibliothek
   |      |
   |      |- linux        Enthaelt uebersetzte Version der Bibliothek
   |      |
   |      |- sun_gcc      Enthaelt uebersetzte Version der Bibliothek
   |      |               (fuer SUN-Sparc mit SUNOS 5.3, GNU-C-Compiler 2.6.2)
   |      |- test         Verschiedene Testprogramme
   |
   |- doku                Dokumentation
   |
   |- estimate            Programme, die fuer die Fehlerabschaetzungen
   |      |               verwendet wurden
   |      |
   |      |- approx       Bestimmung des mathematischen Approximationsfehlers
   |      |
   |      |- eps52        Programme zur Fehlerabschaetzung
   |      |
```

```
|      |- eps53          Programme zur Fehlerabschaetzung
|      |
|      |- modules        Module, die fuer die Fehlerabschaetzungen
|                        verwendet wurden
|
|- f_xsc                 Testprogramme fuer Version zu Fortran-XSC
|      |                 Headerfile libstdf.h
|      |
|      |- gen_utils      Skripte und Preprozessordatei zur Erzeugung der
|                        Version zu Fortran-XSC
|
|- ftp                   Testprogramme fuer FTP-Version, Headerfile
|      |                 fi_lib.h, C++ Interface
|      |
|      |- example        Erste Beispielprogramme
|      |
|      |- gen_utils      Skripte und Preprozessordatei zur Erzeugung der
|                        Version fuer FTP-Server
|
|- misc                  Programme zur Konstantengenerierung, Hilfsprogramme
|
|- p_xsc
|      |
|      |- compile        Batch-Files ("compall.cc','compall.gcc', ...)
|      |      |          zum Erzeugen der Bibliothek
|      |      |- rts
|      |      |
|      |      |- rts_hp
|      |      |
|      |      |- rts_sun
|      |
|      |- hp             Enthaelt uebersetzte Version der Bibliothek
|      |
|      |- linux          Enthaelt uebersetzte Version der Bibliothek
|      |
|      |- sun_gcc        Enthaelt uebersetzte Version der Bibliothek
|      |
|      |- test           Testprogramme
|
|- source                Enthaelt die Quellfiles der Funktionen
       |
       |- source_c       Enthaelt die fuer die ANSI-C-Version benoetigten
       |                 Header-Files und weitere Funktionen
       |- source_p       Quellfiles, die nur fuer die PASCAL-XSC-Version
       |                 benoetigt werden
       |- source_f       Quellfiles, fuer die Version zu Fortran-XSC
       |                 (geaenderte Parameterliste bei Argumentuebergabe)
```

Zusätzlich werden noch die Programme `aktuell.bat`, `gen_ftp.bat` und `gen_fxsc.bat` zur Generierung von verschiedenen Versionen bereitgestellt.

A.2 Erzeugen einer Bibliothek für ANSI-C

Verwendet werden hierzu die Quellfiles in `source` und in `source/source_c`, sowie die entsprechende `compall`-Datei in `ansi_c/compile`.
Vorgehensweise:

I) Im Verzeichnis `source/source_c` muß die Datei `o_defs.h` editiert werden. Bei PCs muß in Zeile 7 der Eintrag `#define INTEL 1`, auf anderen Rechnern muß der Eintrag `#define INTEL 0` stehen.

II) In das Verzeichnis `ansi_c/compile` wechseln

   a) HP-Workstation: Aufruf von `compall.hp`

   b) SUN-Workstation: Aufruf von `compall.sun`

   c) PC unter LINUX: Aufruf von `compall.cc` oder `compall.gcc` (je nach Aufruf des C-Compilers mit `cc` oder `gcc`)

   d) Sonst: Aufruf von `compall.cc` bzw. `compall.gcc`. Bei Problemen müssen eventuell die Compileraufrufe in der Datei `compall.cc` bzw. —verb—compall.gcc— abgeaendert werden oder es müssen noch Compileroptionen hinzugefügt werden.

III) Es müssen die folgenden Dateien erzeugt worden sein:
`fi_ari.h` und `fi_ari.a`

IV) Erster Test der Funktionen:
Hierzu die Dateien `fi_ari.*` in das Verzeichnis `ansi_c/test` kopieren und das Testprogramm `testen.c` übersetzen und starten. Dies kann mit Hilfe der Batchprogramme `testen.cc`, `testen.gcc` bzw. `testen.hp` durchgeführt werden. Auf dem Bildschirm muß die Meldung `Test der Funktionen: Alles OK!` ausgegeben werden.

A.3 Erzeugen einer Bibliothek für C++

Das Vorgehen ist völlig analog zur Erzeugung einer Bibliothek für ANSI-C.

A.4 README–File der `fi_lib`–Installation

```
/*****************************************************************/
/*                                                               */
/*  fi_lib  --- A fast interval library (Version 1.1)            */
/*                                                               */
/*  Authors:                                                     */
/*  --------                                                     */
/*  Werner Hofschuster, Walter Kraemer                           */
/*  Institut fuer Wissenschaftliches Rechnen                     */
/*           und  Mathematische Modellbildung (IWRMM)    and     */
```

```
/*  Institut fuer Angewandte Mathematik                                 */
/*  Universitaet Karlsruhe (TH)                                         */
/*                                                                      */
/*  Disclaimer:                                                         */
/*  -----------                                                         */
/*  This Library is distributed in the hope that it will be useful,     */
/*  but WITHOUT ANY WARRANTY; without even the implied warranty of      */
/*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.                */
/*  Neither the Institut fuer Angewandte Mathematik nor any other       */
/*  facility are responsible for this software. This software does      */
/*  not grant for anything and you can't make the autors responsible    */
/*  for any possible damages or errors!                                 */
/*                                                                      */
/*  Copyright:                                                          */
/*  ----------                                                          */
/*  This package is proprietary of the authors. It may be freely        */
/*  distributed but must not be changed or used for commercial          */
/*  purposes without permission of the authors.                         */
/*                                                                      */
/************************************************************************/


Some remarks about the ANSI-C library fi_lib:
---------------------------------------------

The main features of the new library, called fi_lib
(fast interval library) are:
-  Fast table look-up algortihms are used for the basic functions
   like arctan, exp or log
-  All elementary function routines are supplied with reliable relative
   error bounds of high quality.
   The error estimates cover rounding errors, errors introduced by
   not exactly representable constants as well as
   approximation errors (best approximations with reliable error bounds).
-  All error estimates are reliable worst-case estimates, which
   have been derived using interval methods.
-  We only insist in a faithful computer arithmetic. The routines do not
   manipulate the rounding mode of basic operations (setting the rounding
   mode may be rather expensive).
-  No higher precision internal data format is used. All computations
   are done using the IEEE-double format (64 bit).
-  A C++ interface for easier use is also supplied with the library.
-  To get good portability all programs are written in ANSI-C.
-  Source code and some applications are available via anonymous ftp
   (public domain):
          FTP://iamk4515.mathematik.uni-karlsruhe.de
          Directory: /pub/iwrmm/software

Improvements and extensions of the library are still in progress.
Please contact from time to time our FTP server.

Please send bugs and comments per e-mail to
          werner.hofschuster@math.uni-karlsruhe.de   or
          walter.kraemer@math.uni-karlsruhe.de
```

```
How to unpack the source code of the library fi_lib?
----------------------------------------------------
Use the commands

   gunzip fi_lib.tgz
   tar xf fi_lib.tar

They generate the directory tree

   fi_lib               (source code and makefile)
    |
    |------ test        (integrity test          )
    |
    |------ example     (some simple applications)

To compile the source code, change the directory

   cd fi_lib

and select the correct entries in the configuration section
at the beginning of file 'fi_lib.h', then use the command

   make all

If you have problems with the make command on MS-DOS/Windows PC's you
can use the batch file 'compall.bat'.

If you have problems:
---------------------
1. Check the configuration section in file 'fi_lib.h'
2. Check the call of the C-compiler in the Makefile   (e.g. CC  = cc   or
                                                            = gcc  or ... )
3. Check the call of the C++ compiler in the Makefile (e.g. CPP = CC   or
                                                            = g++  or ... )
4. Check the options for the compiler in the Makefile
   (e.g. on HP-Workstations you must probably use CFLAGS= -Aa and COPT = -O)
   or in the batch file compall.bat
5. Compile the library without any optimization options (e.g. without -O or
   -O2). Some compilers do some nasty things here.

History
-------
1997/08/20  (Version 1.0) First version available on the FTP-server.
1998/03/18  (Version 1.1) Some modifications in the interval routines
            for the basic operations and in the error handling routines
            have been made (e.g. the input/output of intervals and the
            test program fi_test have been modified).
            Also some simple application programs have been added.
```

A.5 Erzeugen einer Bibliothek für Pascal-XSC

Verwendet werden hierzu die Quellfiles in `source` und in `source/source_p`, sowie die entsprechende `compall`-Datei in `p_xsc/compile`.
Vorgehensweise:

I) In das Verzeichnis `p_xsc/compile` wechseln

   a) HP-Workstation: Aufruf von `compall.hp`

   b) SUN-Workstation: Aufruf von `compall.sun`

   c) PC unter LINUX: Aufruf von `compall.cc` oder `compall.gcc`
   (je nach Aufruf des C-Compilers mit `cc` oder `gcc`)

   d) Sonst: In das Unterverzeichnis `rts` wechseln.
   Die Datei `o_slct.h` editieren, es muß hier das richtige „Installation Macro“ (in Abhängigkeit von Rechner und Compiler) durch setzen von „1“ ausgewählt werden, alle anderen Macros müssen mit „0“ gekennzeichnet sein.
   Anschließend die Datei `p88rts.h` editieren, im Abschnitt „configuring section“ müssen für `USING_LINT_ARGS`, `USING_AREALSTRU` und `USING_64_BITSYS` geeignete Werte (siehe Kommentarzeilen) eingetragen werden.
   Danach wieder in das übergeordnete Verzeichnis `compile` wechseln und dort `compall.cc` bzw. `compall.gcc` aufrufen. Bei Problemen müssen eventuell die Compileraufrufe in der Datei `compall.cc` bzw. `compall.gcc` abgeändert werden oder es müssen noch Compileroptionen hinzugefügt werden.

II) Es müssen die folgenden Dateien erzeugt worden sein:
`ff_ari.h`, `ff_ari.mod` und `ff_ari.a`

III) Erster Test der Funktionen:
Hierzu die Dateien `ff_ari.*` in das Verzeichnis `p_xsc/test` kopieren und die Testprogramme `ff_ti.p` (für Punktfunktionen) und `ff_tii.p` (für Intervallfunktionen) mit `mxsc` übersetzen und starten. Auf dem Bildschirm muß in allen Fällen `TRUE` ausgegeben werden.

A.6 Erzeugen einer Bibliothek für C-XSC

Für C–XSC braucht keine eigene Bibliothek erzeugt werden, da die Standardfunktionen in den neuen Versionen (ab 1998) der C–XSC–Bibliothek enthalten sein werden. Verwendet werden hierfür im Prinzip die gleichen Dateien wie bei der ANSI–C Version.

A.7 Übersicht der Filenamen

A.7.1 Verzeichnis `source` (Anzahl: 60 Files)

Namen der C-Quellcodefiles: (Anzahl: 60 Files)

```
j_acos.c  j_cosh.c  j_log.c   q_acot.c q_cos1.c  q_exp2.c  q_sin1.c
j_acot.c  j_cot.c   j_log2.c  q_acsh.c q_cosh.c  q_expm.c  q_sinh.c
j_acsh.c  j_coth.c  j_sin.c   q_acth.c q_cot.c   q_glbl.c  q_sqr.c
j_acth.c  j_ex10.c  j_sinh.c  q_asin.c q_coth.c  q_lg10.c  q_sqrt.c
j_asin.c  j_exp.c   j_sqr.c   q_asnh.c q_cth1.c  q_log.c   q_tan.c
j_asnh.c  j_exp2.c  j_sqrt.c  q_atan.c q_ep1.c   q_log1.c  q_tanh.c
j_atan.c  j_expm.c  j_tan.c   q_atn1.c q_epm1.c  q_log2.c
j_atnh.c  j_lg10.c  j_tanh.c  q_atnh.c q_ex10.c  q_rtrg.c
j_cos.c   j_lg1p.c  q_acos.c  q_cos.c   q_exp.c   q_sin.c
```

A.7.2 Verzeichnis source/source_p (Anzahl: 4 Files)

Namen der C-Quellcodefiles: (Anzahl: 1 File)

```
q_errm.c
```

Namen der Headerfiles: (Anzahl: 3 Files)

```
q_defs.h   q_errm.h  q_fcth.h
```

A.7.3 Verzeichnis source/source_c (Anzahl: 9 Files)

Namen der C-Quellcodefiles: (Anzahl: 5 Files)

```
q_comp.c   q_errm.c  q_pred.c  q_succ.c  q_ari.c
```

Namen der Headerfiles: (Anzahl: 5 Files)

```
o_defs.h   q_defs.h  q_errm.h  q_fcth.h  q_ari.h
```

A.7.4 Verzeichnis p_xsc/compile (Anzahl: 5 Files)

Namen des Pascal-XSC Moduls: (Anzahl: 1 File)

```
ff_ari.p
```

Namen der Batchdateien („Makefile“-Ersatz): (Anzahl: 5 Files)

```
compall.bat  compall.cc  compall.gcc  compall.hp  compall.sun
```

A.7.5 Verzeichnis ansi_c/compile (Anzahl: 4 Files)

Namen des Headerfiles: (Anzahl: 1 File)

```
fi_ari.h
```

Namen der Batchdateien („Makefile“-Ersatz): (Anzahl: 4 Files)

```
compall.bat  compall.cc  compall.gcc  compall.hp
```

A.8 Definitionsbereiche

Definitionsbereiche der Punktfunktionen

Funktion	Definitionsbereich
sqrt	[0,maxreal]
exp	[-maxreal, expmax]
expm1	[-maxreal, expm1max]
log	[nminreal, maxreal]
log1p	[-(one-eps),maxreal]
sinh	[-expmax,expmax]
cosh	[-expm1max,expm1max]
tanh	[-maxreal,maxreal]
coth	[-maxreal,cotmin] oder [cotmin,maxreal]
arsinh	[-maxreal,maxreal]
arcosh	[1,maxreal]
artanh	[-(one-eps),(one-eps)]
arcoth	[-maxreal,-(one+eps)] oder [(one+eps),maxreal]
sin	[-trimax,trimax]
cos	[-trimax,trimax]
tan	[-trimax,trimax]
cot	[-trimax,-nminreal] oder [nminreal,-trimax]
arcsin	[-1,1]
arccos	[-1,1]
arctan	[-maxreal,maxreal]
arccot	[-maxreal,maxreal]
exp2	[-maxreal,exp2max]
exp10	[-maxreal,exp10max]
log2	[nminreal,maxreal]
log10	[nminreal,maxreal]

Definitionsbereiche der Intervallfunktionen

Funktion	Definitionsbereich
sqrt	[0,maxreal]
exp	[-maxreal, expmax]
expm1	[-maxreal, expm1max]
log	[nminreal, maxreal]
log1p	[-(one-eps),maxreal]
sinh	[-expmax,expmax]
cosh	[-expm1max,expm1max]
tanh	[-maxreal,maxreal]
coth	[-maxreal,cotmin] oder [cotmin,maxreal]
arsinh	[-maxreal,maxreal]
arcosh	[1,maxreal]
artanh	[-(one-eps),(one-eps)]
arcoth	[-maxreal,-(one+eps)] oder [(one+eps),maxreal]
sin	[-maxreal,maxreal]
cos	[-maxreal,maxreal]
tan	[-maxreal,maxreal]
cot	[-maxreal,-nminreal] oder [nminreal,-maxreal]
arcsin	[-1,1]
arccos	[-1,1]
arctan	[-maxreal,maxreal]
arccot	[-maxreal,maxreal]
exp2	[-maxreal,exp2max]
exp10	[-maxreal,exp10max]
log2	[nminreal,maxreal]
log10	[nminreal,maxreal]

wobei die Konstanten die folgenden numerischen Werte haben:

Funktion	dezimal	hexadezimal
maxreal	1.797 693 134 862 315 8 E+308	7FEFFFFF FFFFFFFF
dminreal	4.940 656 458 412 465 4 E-324	00000000 00000001
nminreal	2.225 073 858 507 201 3 E-308	00100000 00000000
expmax	7.097 827 128 933 840 E+002	40862E42 FEFA39EF
expm1max	7.090 895 657 128 240 E+002	408628B7 6E3A7B60
cotmin	5.562 684 646 268 013 E-309	00040000 00000002
trimax	3.373 259 425 345 106 E+009	41E921FB 542B0B1C
exp2max	1.023 E+003	408FF800 00000000
exp10max	3.082 547 155 599 1 E+002	4008A90E 7BAD685D
(one-eps)	0.999 999 999 999 999 9	3FEFFFFF FFFFFFFF
(one+eps)	1.000 000 000 000 000 2	3FF00000 00000001

A.9 Fehlerschranken

Es gilt: $\varepsilon^* = \frac{1}{2}2^{-52} = 1.11022\ldots10^{-16}$. Die Tabelle listet die Fehlerschranken der in der Bibliothek `fi_lib` implementierten Funktionen. Dabei wird keine Voraussetzung über den Rundungsmodus der Grundoperationen gemacht.

Funktion	rel. Fehlerschranke	
sqr	$2.2205 \cdot 10^{-16}$	$2.00 \cdot \varepsilon^*$
sqrt	$2.2205 \cdot 10^{-16}$	$2.00 \cdot \varepsilon^*$
exp	$2.3580 \cdot 10^{-16}$	$2.13 \cdot \varepsilon^*$
expm1	$2.5926 \cdot 10^{-16}$	$2.34 \cdot \varepsilon^*$
log	$2.9398 \cdot 10^{-16}$	$2.65 \cdot \varepsilon^*$
log1p	$2.5082 \cdot 10^{-16}$	$2.26 \cdot \varepsilon^*$
sinh	$7.0933 \cdot 10^{-16}$	$6.39 \cdot \varepsilon^*$
cosh	$4.5817 \cdot 10^{-16}$	$4.13 \cdot \varepsilon^*$
tanh	$1.0546 \cdot 10^{-15}$	$9.50 \cdot \varepsilon^*$
coth	$8.3253 \cdot 10^{-16}$	$7.50 \cdot \varepsilon^*$
arsinh	$7.2075 \cdot 10^{-16}$	$6.50 \cdot \varepsilon^*$
arcosh	$1.6180 \cdot 10^{-15}$	$14.58 \cdot \varepsilon^*$
artanh	$1.2647 \cdot 10^{-15}$	$11.40 \cdot \varepsilon^*$
arcoth	$1.1479 \cdot 10^{-15}$	$10.34 \cdot \varepsilon^*$
sin	$1.0718 \cdot 10^{-15}$	$9.66 \cdot \varepsilon^*$
cos	$1.0718 \cdot 10^{-15}$	$9.66 \cdot \varepsilon^*$
tan	$2.9777 \cdot 10^{-15}$	$26.83 \cdot \varepsilon^*$
cot	$2.9777 \cdot 10^{-15}$	$26.83 \cdot \varepsilon^*$
arcsin	$2.1489 \cdot 10^{-15}$	$19.36 \cdot \varepsilon^*$
arccos	$2.1485 \cdot 10^{-15}$	$19.36 \cdot \varepsilon^*$
arctan	$1.3588 \cdot 10^{-15}$	$12.24 \cdot \varepsilon^*$
arccot	$1.8029 \cdot 10^{-15}$	$16.24 \cdot \varepsilon^*$
exp2	$2.3305 \cdot 10^{-16}$	$2.10 \cdot \varepsilon^*$
exp10	$2.4181 \cdot 10^{-16}$	$2.18 \cdot \varepsilon^*$
log2	$2.7754 \cdot 10^{-15}$	$25.00 \cdot \varepsilon^*$
log10	$2.7754 \cdot 10^{-15}$	$25.00 \cdot \varepsilon^*$

A.10 Fehlerschranken bei maximal genauer Arithmetik

Im folgenden werden die bereits vorhanden Ergebnisse der Fehlerabschaetzungen bei Rechnung mit $\overline{\varepsilon} := \varepsilon^* = 2^{-53} = 1.11022\ldots10^{-16}$ angegeben. Die Fehlerschranken werden sich nahezu halbieren. Man beachte, dass bei der verschärften über den Rundungsmodus die Approximationsgenauigkeit nicht geändert wird und dass bereits bei Annahme eines beliebigen Rundungsmodus (wie in A.9 angenommen) alle verwendeten Konstanten round-to-nearest abgespeichert sind.
Funktion expm1 (vergleiche mit den Ergebnissen auf Seite 52):

```
Test mit 1000 Zufallszahlen (Bereich I):
Minimale Groessenordnung der Fehlerschranke :  1.257640815122943E-016

Test mit 1000 Zufallszahlen (Bereich II):
Minimale Groessenordnung der Fehlerschranke :  1.220239774332858E-016

Bereich Ia:  Max. rel. Fehler =  1.171301864727144E-016
Bereich Ib:  Max. rel. Fehler =  1.225259470548786E-016
Bereich IIa: Max. rel. Fehler =  1.274432066933006E-016
Bereich IIb: Max. rel. Fehler =  1.243354537645288E-016
Bereich IIc: Max. rel. Fehler =  1.233948443989451E-016
Bereich IId: Max. rel. Fehler =  1.193419835087194E-016
Bereich Ic:  Max. rel. Fehler =  1.301739721374002E-016
Bereich Id:  Max. rel. Fehler =  1.236443521987139E-016
Bereich Ie:  Max. rel. Fehler =  1.186229075642638E-016

Max. rel. Fehlerschranke der ffexpm1-Funktion: 1.301739721374002E-016

------------------------------------------------------------
Fehlerkonstanten fuer ANSI-C Programme:
/* eps(q_....) = 1.301739721374002E-016;                  */
/*     q_...m  = 1 - eps(q_....) = 9.999999999999996E-001 */
double q_...m  = 9007199254740988.0 / 9007199254740992.0;
/*     q_...p  = 1 + eps(q_....) = 1.000000000000001E+000 */
double q_...p  = 4503599627370499.0 / 4503599627370496.0;
```

A.11 Programme zur Fehlerabschätzung

Die folgenden Tabelle listet die Namen der Dateien der in Pascal–XSC implementieren Fehlerabschätzungsprogramme auf. Beispielhaft ist das Programm `ffexpm1.p` ab Seite 41 abgedruckt.

Funktion	Programmname
exp	`ffexp.p`
expm1	`ffexpm1.p`
log	`fflog.p`
log1p	`fflog1p.p`
sinh	`ffsinh.p`
cosh	`ffcosh.p`
tanh	`fftanh.p`
coth	`ffcoth.p`
arsinh	`ffasinh.p`
arcosh	`ffacosh.p`
artanh	`ffatanh.p`
arcoth	`ffacoth.p`
sin	`ffsin.p`
cos	`ffcos.p`
tan	`fftan.p`
cot	`ffcot.p`
arcsin	`ffasin.p`
arccos	`ffacos.p`
arctan	`ffatan.p`
arccot	`ffacot.p`
exp2	`ffexp2.p`
exp10	`ffex10.p`
log2	`fflog2.p`
log10	`fflg10.p`

A.12 Übersicht der implementierten Funktionen und Unterprogramme

Funktionsname	Implementierung in File	Bedeutung	verwendete Funktionen	verwendete Konstanten
q_acos	q_acos.c	arccos-Punktfunktion	q_abortnan q_abortr1 sqrt q_atn1	q_piha
q_acot	q_acot.c	arccot-Punktfunktion	q_abortnan q_atn1	q_piha q_pi
q_acsh	q_acsh.c	arcosh-Punktfunktion	q_abortnan sqrt q_log1 q_l1p1	q_l2
q_acth	q_acth.c	arcoth-Punktfunktion	q_abortnan q_abortr1 q_l1p1	
q_asin	q_asin.c	arcsin-Punktfunktion	q_abortnan q_abortr1 sqrt q_atn1	q_piha q_atnt
q_asnh	q_asnh.c	arsinh-Punktfunktion	q_abortnan q_log1 q_l1p1 sqrt	q_l2
q_atan	q_atan.c	arctan-Punktfunktion	q_abortnan	q_piha q_atnb[*] q_atnc[*] q_atnd[*]
q_atn1	q_atn1.c	arctan-Punktfunktion, ohne Fehlerabbruch, darf nur mit zulässigen Argumenten aufgerufen werden		q_piha q_atnb[*] q_atnc[*] q_atnd[*]
q_atnh	q_atnh.c	artanh-Punktfunktion	q_abortnan q_abortr1 q_log1 q_l1p1	
q_cos	q_cos.c	cos-Punktfunktion	q_abortnan q_abortr1 q_rtrg	q_pi2i q_sint[*] q_sins[*] q_sinc[*]
q_cos1	q_cos1.c	cos-Punktfunktion, es wird keine Argumentreduktion durchgeführt	q_abortnan q_abortr1	q_pi2i q_sint[*] q_sins[*] q_sinc[*]

Funktionsname	Implementierung in File	Bedeutung	verwendete Funktionen	verwendete Konstanten
q_cosh	q_cosh.c	cosh-Punktfunktion	q_abortnan q_abortr1 q_ep1	q_ex2c
q_cot	q_cot.c	cot-Punktfunktion	q_abortnan q_abortr1 q_rtrg	q_pi2i q_sint[*] q_sins[*] q_sinc[*] q_minr
q_coth	q_coth.c	coth-Punktfunktion	q_abortnan q_abortr1 q_ep1 q_epm1	q_ln2h
q_cth1	q_cth1.c	coth-Punktfunktion, ohne Fehlerabbruch, darf nur mit zulässigen Argumenten aufgerufen werden	q_ep1 q_epm1	q_ln2h
q_ep1	q_ep1.c	exp-Punktfunktion, ohne Prüfung auf NaN, darf nur mit zulässigen Argumenten aufgerufen werden	q_abortr1	q_ext1 q_ex2a q_ex2b q_exil q_exl1 q_exl2 q_exa[*] q_exld[*] q_extl[*]
q_epm1	q_epm1.c	(exp-1)-Punktfunktion, ohne Prüfung auf NaN, darf nur mit zulässigen Argumenten aufgerufen werden	q_abortr1	q_ext1 q_ex2c q_ext3 q_ext4 q_ext5 q_p2mh q_exil q_exl1 q_exl2 q_exa[*] q_exb[*] q_exld[*] q_extl[*]
q_ex10	q_ex10.c	Exponential-Punktfunktion zur Basis 10	q_abortnan q_abortr1	q_ext1 q_e10i q_e1l1 q_e1l2 q_exd[*] q_exld[*] q_extl[*]

Funktionsname	Implementierung in File	Bedeutung	verwendete Funktionen	verwendete Konstanten
q_exp	q_exp.c	exp-Punktfunktion,	q_abortnan q_abortr1	q_ext1 q_ex2a q_exil q_exl1 q_exl2 q_exa[*] q_exld[*] q_extl[*]
q_exp2	q_exp2.c	Exponential-Punktfunktion zur Basis 2	q_abortnan q_abortr1	q_ext1 q_exc[*] q_exld[*] q_extl[*]
q_expm	q_expm.c	(exp-1)-Punktfunktion,	q_abortnan q_arbortr1	q_ext1 q_ex2c q_ext3 q_ext4 q_ext5 q_p2h q_p2mh q_exil q_exl1 q_exl2 q_exa[*] q_exb[*] q_exld[*] q_extl[*]
	q_glbl.c	Globale Konstanten, Fehlerkonstanten für die Intervallfunktionen		
q_lg10	q_lg10.c	Logarithmus-Punktfunktion zur Basis 10	q_abortnan q_log	q_l10i
q_log	q_log.c	log-Punktfunktion	q_abortnan q_arbortr1	q_minr q_lgt1 q_lgt2 q_lgb[*] q_lgc[*] q_lgld[*] q_lgtl[*]
q_lg1p	q_log.c	log(1+x)-Punktfunktion	q_abortnan q_abortr1	q_lgt3 q_lgt4 q_lgt5 q_lgt6 q_lgb[*] q_lgc[*] q_lgld[*] q_lgtl[*]

Funktionsname	Implementierung in File	Bedeutung	verwendete Funktionen	verwendete Konstanten
q_log1	q_log1.c	log-Punktfunktion, Version ohne Argumentprüfung	q_abortr1	q_minr q_lgt1 q_lgt2 q_lgb[*] q_lgc[*] q_lgld[*] q_lgtl[*]
q_l1p1	q_log1.c	log(1+x)-Punktfunktion, Version ohne Argumentprüfung	q_abortr1	q_lgt3 q_lgt4 q_lgt5 q_lgt6 q_lgb[*] q_lgc[*] q_lgld[*] q_lgtl[*]
q_log2	q_log2.c	Logarithmus-Punktfunktion zur Basis 2	q_abortnan q_log	q_l2i
q_rtrg	q_rtrg.c	Argumentreduktion für die trig. Funktionen	q_r2tr	q_pih[*]
q_r2tr	q_rtrg.c	Argumentreduktion für die trig. Funktionen		q_pih[*]
q_sin	q_sin.c	sin-Punktfunktion	q_abortnan q_arbortr1 q_rtrg	q_pi2i q_sint[*] q_sinc[*] q_sins[*]
q_sin1	q_sin1.c	sin-Punktfunktion ohne Argumentreduktion	q_abortnan q_arbortr1 q_rtrg	q_sint[*] q_sinc[*] q_sins[*]
q_sinh	q_sinh.c	sinh-Punktfunktion	q_abortnan q_arbortr1 q_ep1 q_epm1	
q_sqr	q_sqr.c	sqr-Punktfunktion	q_abortnan q_arbortr1	
q_sqrt	q_sqrt.c	sqrt-Punktfunktion verwendet sqrt aus <math.h>	q_abortnan q_arbortr1 sqrt	
q_tan	q_tan.c	tan-Punktfunktion	q_abortnan q_arbortr1 q_rtrg	q_pi2i q_sint[*] q_sinc[*] q_sins[*]
q_tanh	q_tanh.c	tanh-Punktfunktion	q_abortnan q_arbortr1 q_cth1	

Funktionsname	Implementierung in File	Bedeutung	verwendete Funktionen	verwendete Konstanten
j_acos	j_acos.c	arccos-Intervallfunktion	q_acos	q_ccsp q_ccsm
j_acot	j_acot.c	arccot-Intervallfunktion	q_acot	q_cctp q_cctm
j_acsh	j_acsh.c	arcosh-Intervallfunktion	q_abortr2 q_acsh	q_acsp q_acsm
j_acth	j_acth.c	arcoth-Intervallfunktion	q_abortr2 q_acth	q_actp q_actm
j_asin	j_asin.c	arcsin-Intervallfunktion	r_pred r_succ q_asin	q_csnp q_csnm q_atnt
j_asnh	j_asnh.c	arsinh-Intervallfunktion	r_pred r_succ q_asnh	q_asnp q_asnm q_minr
j_atan	j_atan.c	arctan-Intervallfunktion	r_pred r_succ q_atan	q_ctnp q_ctnm q_atnt
j_atnh	j_atnh.c	artanh-Intervallfunktion	r_pred r_succ q_abortr2 q_atnh	q_atnp q_atnm q_minr
j_cos	j_cos.c	cos-Intervallfunktion	q_cos q_rtrg q_cos1	q_sint[*] q_cosp q_cosm q_pi q_pi2i q_sinp q_sinm
j_cosh	j_cosh.c	cosh-Intervallfunktion	q_cosh	q_cshp q_cshm
j_cot	j_cot.c	cot-Intervallfunktion	q_cot q_abortr2 q_cos1	q_sint[*] q_cotp q_cotm q_pi2i
j_coth	j_coth.c	coth-Intervallfunktion	q_abortr2 q_coth	q_cthp q_cthm
j_ex10	j_ex10.c	Exponential-Intervallfunktion zur Basis 10	q_ex10	q_minr q_e10p q_e10m
j_exp	j_exp.c	Exponential-Intervallfunktion zur Basis e	q_exp	q_minr q_exep q_exem q_mine
j_exp2	j_exp2.c	Exponential-Intervallfunktion zur Basis 2	q_exp2	q_minr q_e2ep q_e2em

Funktionsname	Implementierung in File	Bedeutung	verwendete Funktionen	verwendete Konstanten
j_expm	j_expm.c	(exp-1)-Intervallfunktion	q_expm r_succ	q_minr q_exmp q_exmm
j_lg10	j_lg10.c	Logarithmus-Intervallfunktion zur Basis 10	q_lg10	q_l10p q_l10m
j_lg1p	j_lg1p.c	log(x+1) - Intervallfunktion	q_lg1p	q_lgpp q_lgpm
j_log	j_log.c	Logarithmus - Intervallfunktion	q_log	q_logp q_logm
j_log2	j_log2.c	Logarithmus-Intervallfunktion zur Basis 2	q_log2	q_lg2p q_lg2m
j_sin	j_sin.c	sin-Intervallfunktion	q_sin q_rtrg q_sin1 r_pred r_succ	q_sint[*] q_sinp q_sinm q_pi q_pi2i
j_sinh	j_sinh.c	sinh-Intervallfunktion	q_sinh r_pred r_succ	q_minr q_snhm q_snhp
j_sqr	j_sqr.c	Quadrat-Intervallfunktion	q_sqr q_abortnan q_abortr2 r_pred r_succ	
j_sqrt	j_sqrt.c	Quadratwurzel-Intervallfunktion	q_sqrt r_pred r_succ	
j_tan	j_tan.c	tan-Intervallfunktion	q_abortr2 q_tan r_pred r_succ r_succ	q_sint[*] q_tanp q_tanm q_pi2i
j_tanh	j_tanh.c	tanh-Intervallfunktion	q_tanh r_pred r_succ	q_minr q_tnhp q_tnhm
Version P-XSC	q_defs.h	Headerfile enthält Makros, Konstantenvereinbarungen	q_errm.h	
POWER2 Version P-XSC	q_defs.h	Makro, Ersatz für ldexp aus math.h		
FREXPO Version P-XSC	q_defs.h	Makro, Ersatz für frexp aus math.h		
CUT24 Version P-XSC	q_defs.h	Makro, Abschneiden auf 24 Bit Länge		
CUTINT Version P-XSC	q_defs.h	Makro, Zuweisung double an int		

Funktionsname	Implementierung in File	Bedeutung	verwendete Funktionen	verwendete Konstanten
Version P-XSC	q_errm.h	Headerfile Fehlerhandling	q_defs.h q_fcth.h	
NANTEST Version P-XSC	q_errm.h	Makro, Test x = NAN	t_exc_.h t_defs.h	
q_abortnan Version P-XSC	q_errm.c	Abbruch und Fehlermeldung	ieee_abortr1	
q_abortr1 Version P-XSC	q_errm.c	Abbruch und Fehlermeldung	ieee_abortr1	
q_abortr2 Version P-XSC	q_errm.c	Abbruch und Fehlermeldung	ieee_abortr2	
Version P-XSC	q_fcth.h	Headerfile Prototypen		
Version ANSI-C	o_defs.h	Headerfile Steuerung bed. Übers.	math.h	
r_succ Version ANSI-C	o_defs.h	Makro, ersetzt r_succ durch q_succ		
r_pred Version ANSI-C	o_defs.h	Makro, ersetzt r_pred durch q_pred		
Version ANSI-C	q_errm.h	Headerfile Fehlerhandling	q_defs.h q_fcth.h stdlib.h	
NANTEST Version ANSI-C	q_errm.h	Makro, Test x = NAN	t_exc_.h t_defs.h	
q_abortnan Version ANSI-C	q_errm.c	Abbruch und Fehlermeldung	exit	
q_abortr1 Version ANSI-C	q_errm.c	Abbruch und Fehlermeldung	exit	
q_abortr2 Version ANSI-C	q_errm.c	Abbruch und Fehlermeldung	exit	
Version ANSI-C	q_defs.h	Headerfile enthält Makros, Konstantenvereinbarungen	q_errm.h	
POWER2 Version ANSI-C	q_defs.h	Makro, Ersatz für ldexp aus math.h		
FREXPO Version ANSI-C	q_defs.h	Makro, Ersatz für frexp aus math.h		
CUT24 Version ANSI-C	q_defs.h	Makro, Abschneiden auf 24 Bit Länge		
CUTINT Version ANSI-C	q_defs.h	Makro, Zuweisung double an int		
Version ANSI-C	q_fcth.h	Headerfile Prototypen		
q_pred Version ANSI-C	q_pred.c	Gleitkomma-vorgänger		
q_succ Version ANSI-C	q_succ.c	Gleitkomma-nachfolger		

Funktionsname	Implementierung in File	Bedeutung	verwendete Funktionen	verwendete Konstanten
Version ANSI-C	`ti_ari.h`	Headerfile Intervallgrund-arithmetik	`o_defs.h` `q_fcth.h`	
... Version ANSI-C	`ti_ari.c`	Intervall-grundarithmetik	`q_pred` `q_succ`	

Alle Funktionen verwenden die Headerdateien `o_defs.h`, `q_defs.h` und `q_fcth.h`.

A.13 Verwendete Funktionen des Pascal-XSC Laufzeitsystems

Die Version für Pascal-XSC verwendet die folgenden Funktionen aus dem Laufzeitsystem von Pascal-XSC:

```
E_SPUSH()
E_SPOPP()
ieee_abortr1()
ieee_abortr2()
r_pred()
r_succ()
```

Die folgenden Headerfiles des Laufzeitsystems werden eingebunden:

```
o_defs.h
t_exc_.h
t_defs.h
```

A.14 Redundanter Programmcode

Aus Laufzeitgründen wurde an manchen Stellen bestimmter Programmcode mehrfach implementiert. Da dies bei späteren Änderungen und Modifikationen eine Fehlerquelle sein könnte, sind diese Stellen hier aufgeführt:

- Berechnung der arctan-Punktfunktion:
  `q_atan.c` und `q_atn1.c`
- Berechnung des cos-Punktfunktion:
  `q_cos.c` und `q_cos1.c`
- Teile der Argumentreduktion bei den trigonometrischen Funktionen:
  `q_cos.c`, `q_cot.c`, `q_rtrg.c`, `q_sin.c`, `q_tan.c`, `j_cos.c`, `j_cot.c`, `j_sin.c`, `q_tan.c`

- Approximation Sinusfunktion:
  `q_cos.c`, `q_cot.c`, `q_sin.c`, `q_sin1.c`, `q_tan.c`
- Approximation Cosinusfunktion:
  `q_cos.c`, `q_cot.c`, `q_sin.c`, `q_sin1.c`, `q_tan.c`
- „Intervalllogik“ der Sinus-/Cosinusfunktion:
  `j_cos.c` und `j_sin.c`
- Berechnung der Exponentialfunktion:
  `q_ep1.c` und `q_exp.c`
- Berechnung der Exponentialfunktion minus 1:
  `q_epm1.c` und `q_expm.c`
- Berechnung der Logarithmusfunktion und der Funktion $\log(1+x)$:
  `q_log.c` und `q_log1.c`

Eine Modifikation am Programmquellcode muß immer in allen betroffenen Dateien erfolgen!

Anhang B

Modul abs_ari

```
module abs_ari;
{------------------------------------------------------------------------}
{           F e h l e r s c h r a n k e n a r i t h m e t i k            }
{                                                                        }
{         zur sicheren Abschaetzung von   a b s o l u t e n   Fehlern    }
{                                                                        }
{                  Werner Hofschuster und Walter Kraemer                 }
{                                                                        }
{                  Letzte Aenderung am Modul: 29.10.1997                 }
{------------------------------------------------------------------------}

use i_ari;  { Intervallarithmetik einbinden }
use iostd;  { Ermoeglicht sauberen Programmabbruch }
$off
use ff_ari; { fuer Funktion expm1 }
$on

{------------------------------------------------------------------------}
{                Globale Vereinbarung des Fehlerdatentyps                }
{------------------------------------------------------------------------}
global type BoundType = global record  { Neuer Datentyp                  }
              Enclosure: interval;     { Einschliessung der korrekten Werte }
              AbsErr:    real;         { Zugehoeriger max. absoluter Fehler }
            end;
{                                                                        }
{ Ist a eine Variable vom Typ BoundType, so liegt die durch diese        }
{ Variable repraesentierte exakte Groesse im Intervall a.Enclosure. Fuer }
{ die gestoerten Groessen gilt die absolute Fehlerschranke a.AbsErr      }
{------------------------------------------------------------------------}

{------------------------------------------------------------------------}
{                Einige globale Groessen                                 }
{------------------------------------------------------------------------}

global const Eps53 = 1.110224E-16;
{ Maschinenepsilon: IEEE RoundToNearest, 2**(-53)= 1.110223...E-16 }

global const Eps52 = 2.220447E-16;
{ Maschinenepsilon: IEEE RoundToDown/Up, 2**(-52)= 2.220446...E-16 }
```

```
global const MinReal = 2.2250738585072013E-308;
{ Kleinste positive normalisierte Gleitkommazahl }

global const dMinReal = 4.9406564584124654e-324;
{ Kleinste positive normalisierte Gleitkommazahl }

global const MaxReal = 1.7976931348623158e308;
{ Groesste positive Gleitkommazahl }

var UnflowRange: interval;
{ Bereich des gradual Underflows und Underflows }

var EpsQuer: real;
{ Relativer Fehler fuer eine Maschinenoperation }

var EpsArctan,
    EpsExp,
    EpsExpm1,
    EpsLn,
    EpsLn1p,
    EpsCoth    : real;
{ Relative Fehler fuer Auswertung von Funktionen mit exaktem Argument       }

var DelArctan,
    DelExp,
    DelExpm1   : real;
{ Absolute Fehler fuer Auswertung von Funktionen mit exaktem Argument       }

global var EpsAriTest: boolean;        { Unterlaufwarnungen ein/ausschalten }

{-------------------------------------------------------------------------}
{                 Einige globale Hilfsfunktionen                          }
{-------------------------------------------------------------------------}

global procedure Set_EpsQuer(EpsAkt:real);
begin
  EpsQuer:=EpsAkt;
end;

global function pred(x:real; n:integer):real;
var i:integer;
begin
  for i:=1 to n do x:=pred(x);
  pred:=x;
end;

global function succ(x:real; n:integer):real;
var i:integer;
begin
  for i:=1 to n do x:=succ(x);
  succ:=x;
end;
```

```
global function MaxAbs(x: interval): real;
begin
  MaxAbs:= sup( abs(x) );
end;

global function MinAbs(x: interval): real;
begin
  MinAbs:= inf( abs(x) );
end;

global function Max(x, y: real): real;
begin
  if x >= y then Max:=x else Max:=y;
end;

global function Min(x, y: real): real;
begin
  if x <= y then Min:=x else Min:=y;
end;

{------- Hilfsfunktion: Umwandlung relativer in absoluter Fehler -----------}
global function rel2abs(a:interval; eps_a:real):real;
begin
  rel2abs:= MaxAbs(a)*>eps_a;
end;

{------- Hilfsprozedur: Ausgabe von Fehlerkonstanten fuer ANSI-C -----------}
global procedure em1_ep1(eps_f:real);

var {Hilfsvariablen fuer Zerlegung}
  rr: real;
  ss: real;
  cc: real;
  ex: integer;

begin
    writeln('-------------------------------------------------------------');
    writeln('Fehlerkonstanten fuer ANSI-C Programme:');
    cc:= (1 -< eps_f) *< (1 -< EpsQuer);
    ex:= expo(cc);
    rr:= cc*power(2, 53-ex); { transform to integer }
    ss:=    power(2, 53-ex); { denominator = 2**    }
    writeln('/* eps(q_....) =',eps_f,';                  */');
    writeln('/*     q_...m  = 1 - eps(q_....) =',cc:23,' */');
    writeln('double q_...m  =',  rr:19:1, ' / ', ss:18:1, ';');

    cc:= (1 +> eps_f) *> (1 +> EpsQuer);
    ex:= expo(cc);
    rr:= cc*power(2, 53-ex); { transform to integer }
    ss:=    power(2, 53-ex); { denominator = 2**    }
    writeln('/*     q_...p  = 1 + eps(q_....) =',cc:23,' */');
    writeln('double q_...p  =',  rr:19:1, ' / ', ss:18:1, ';');
end;
```

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

global function EpsInv(Eps: real; var code: integer): real;
  { 1/(1+Eps) = 1+EpsInv;  Eps<=0.5 ! }
begin
  code:= 0;
  if Eps > 0.5 then begin
    EpsInv:= -1;
    code:= 1
  end
  else EpsInv:= Eps *> (1 +> 2*>Eps)
end;


{------------------------------------------------------------------------}
{           F e h l e r s c h r a n k e n a r i t h m e t i k          }
{                          fuer +, -, *, /.                            }
{  Absolute Fehlerschranken der beiden Operanden sind bekannt.         }
{  Die exakten Werte der Operanden liegen in den Intervallen           }
{  alpha und beta.                                                     }
{------------------------------------------------------------------------}


global function DeltaAdd(
               alpha, beta: interval; DeltaA, DeltaB: real ): real;
  {-------------------------------------------------------------}
  { Berechnet wird die absolute Fehlerschranke bei einer        }
  {   Addition von fehlerbehafteten Groessen.                   }
  {-------------------------------------------------------------}
  { Die exakten Werte des 1. Summanden liegen im Intervall alpha.}
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaA beschraenkt.                                         }
  { Die exakten Werte des 2. Summanden liegen im Intervall beta.}
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaB beschraenkt.                                         }
  {-------------------------------------------------------------}
var u, v: real;
    ResultSet: interval;
begin
  if (DeltaA=0) and (DeltaB=0) and ((alpha=0) or (beta=0)) then
    DeltaAdd:= 0
  else begin
    if EpsAriTest then begin  { Unterlaufwarnung }
      ResultSet:= alpha + intval(-DeltaA, DeltaA)
                        + beta + intval(-DeltaB, DeltaB);
      if not (ResultSet >< UnflowRange) then begin
        write('  DeltaAdd: Ergebnis im Unterlauf!  Weiter mit <Return> ');
        readln;
      end;
    end; { Unterlaufwarnung }
    u:= EpsQuer *> MaxAbs(alpha+beta);
    v:= (DeltaA +> DeltaB) *> (1 +> EpsQuer);
    DeltaAdd:= u +> v +> MinReal;
```

```
  end;
end;


global function DeltaSub(
                alpha, beta: interval; DeltaA, DeltaB: real ): real;
  {-------------------------------------------------------------}
  { Berechnet wird die absolute Fehlerschranke bei einer        }
  {   Subtraktion von fehlerbehafteten Groessen.                }
  {-------------------------------------------------------------}
  { Die exakten Werte des 1. Operanden liegen im Intervall alpha.}
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaA beschraenkt.                                         }
  { Die exakten Werte des 2. Operanden liegen im Intervall beta. }
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaB beschraenkt.                                         }
  {-------------------------------------------------------------}
begin
  DeltaSub:=DeltaAdd(alpha, -beta, DeltaA, DeltaB);
end;


global function DeltaMul(
                alpha, beta: interval; DeltaA, DeltaB: real ): real;
  {-------------------------------------------------------------}
  { Berechnet wird die absolute Fehlerschranke bei einer        }
  {   Multiplikation von fehlerbehafteten Groessen.             }
  {-------------------------------------------------------------}
  { Die exakten Werte des 1. Faktors liegen im Intervall alpha.  }
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaA beschraenkt.                                         }
  { Die exakten Werte des 2. Faktors liegen im Intervall beta.   }
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaB beschraenkt.                                         }
  {-------------------------------------------------------------}

var u, v: real;
var ResultSet: interval;
begin
  if (alpha=1) and (DeltaA=0) then
    DeltaMul:= DeltaB
  else if (beta=1) and (DeltaB=0) then
    DeltaMul:= DeltaA
  else begin
    u:= MaxAbs(alpha) *> DeltaB + MaxAbs(beta) *> DeltaA;
    v:= (1 +> EpsQuer) *> (u +> DeltaA *> DeltaB);
    DeltaMul:=  EpsQuer *> MaxAbs(alpha*beta) +> v +> MinReal;
    if EpsAriTest then begin  { Unterlaufwarnung }
      ResultSet:= ( alpha + intval(-DeltaA,DeltaA) )
                    * ( beta + intval(-DeltaB,DeltaB) );
      if not (ResultSet >< UnflowRange) then begin
        write('  DeltaMul: Ergebnis im Unterlauf!  Weiter mit <Return> ');
        readln;
      end;
```

```
    end;  { Unterlaufwarnung }
  end;
end;


global function DeltaDiv(
               alpha, beta: interval; DeltaA, DeltaB: real ): real;
  {-------------------------------------------------------------}
  { Berechnet wird die absolute Fehlerschranke bei einer        }
  {   Division von fehlerbehafteten Groessen.                   }
  {-------------------------------------------------------------}
  { Die exakten Werte des 1. Operanden liegen im Intervall alpha.}
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaA beschraenkt.                                         }
  { Die exakten Werte des 2. Operanden liegen im Intervall beta.}
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaB beschraenkt.                                         }
  {-------------------------------------------------------------}

var u:real;
var code:integer;
var ResultSet: interval;

begin
  if (intval(0) <+ beta) then DeltaDiv:=(1.0/0);
                                    {Programmabbruch !  ( 0 in beta ) }

  if (beta=1) and (DeltaA=0) and (DeltaB=0) then
    DeltaDiv:= 0
  else begin
    u:= DeltaB /> MinAbs(beta);
    u:=EpsInv(u, code);
    if code=1 then writeln('Fehler bei EpsInv !');
    u:=EpsQuer+>u;
    u:=u*>(MaxAbs(alpha)+>DeltaA)+>DeltaA;
    DeltaDiv:=u /> ( MinAbs(beta) -< DeltaB ) +> MinReal;

    if EpsAriTest then begin  { Unterlaufwarnung }
      ResultSet:= ( alpha + intval(-DeltaA,DeltaA) )
                  / ( beta + intval(-DeltaB,DeltaB) );
      if not (ResultSet >< UnflowRange) then begin
        write('  DeltaDiv: Ergebnis im Unterlauf!  Weiter mit <Return> ');
        readln;
      end;
    end;  { Unterlaufwarnung }
  end;
end;


{-------------------------------------------------------------------------}
{                                                                         }
{             F e h l e r s c h r a n k e n a r i t h m e t i k           }
{                                                                         }
{  fuer Funktionen SQRT, EXP, EXPM1:=exp(x)-1, LN,  LN1P:=ln(1+x)         }
```

```
{                  und ARCTAN  ...                                          }
{                                                                           }
{ Die absolute Fehlerschranke DELTA des Argumentes ist bekannt.             }
{ Der exakte Wert des Operanden liegt im Intervall alpha.                   }
{---------------------------------------------------------------------------}

global function DeltaSqrt(Alpha: interval; DeltaA: real): real;
begin
   if (Alpha.inf > DeltaA) then
     DeltaSqrt:= EpsQuer *> MaxAbs(sqrt(alpha))
                 +> 0.5 *> (1 +> EpsQuer) *> DeltaA
                 /> MinAbs(sqrt(Alpha-intval(-DeltaA,DeltaA) ) )
                 +> dminreal
   else begin
     writeln('*** DeltaSqrt: Argument <= 0!');
     exit(1); { Programmabbruch }
   end;
end;

global function DeltaArctan(Alpha: interval; DeltaA: real): real;
begin
   DeltaArctan:= EpsArctan *> MaxAbs(arctan(alpha))
                 +> (1 +> EpsArctan) *> DeltaA
                 /> ( 1 +< MinAbs(sqr(Alpha+intval(-DeltaA,DeltaA) ) ) )
                 +> DelArctan;
end;

global function DeltaExp(Alpha: interval; DeltaA: real): real;
begin
   DeltaExp:= EpsExp *> MaxAbs(exp(alpha))
                 +> (1 +> EpsExp) *> DeltaA
                 *> MaxAbs(exp(Alpha+intval(-DeltaA,DeltaA)))
                 +> DelExp;
end;

global function DeltaExpm1(Alpha: interval; DeltaA: real): real;
begin
   DeltaExpm1:= EpsExpm1 *> MaxAbs(exp(alpha)-1)
                 +> (1 +> EpsExpm1) *> DeltaA
                 *> MaxAbs(exp(Alpha+intval(-DeltaA,DeltaA)))
                 +> DelExpm1;
$off
   if MaxAbs(Alpha) <= MinReal then  { Alpha ganz im Unterlaufbereich }
     DeltaExpm1:= DelExpm1
   else
     DeltaExpm1:= EpsExpm1 *> MaxAbs(ffexpm1(alpha))
                 +> (1 +> EpsExpm1) *> DeltaA
                 *> MaxAbs(exp(Alpha+intval(-DeltaA,DeltaA)))
                 {+> DelExpm1};
$on
end;

global function DeltaLn(Alpha: interval; DeltaA: real): real;
begin
```

```
   DeltaLn:= EpsLn *> MaxAbs(ln(alpha))
                +> (1 +> EpsLn) *> DeltaA
                /> MinAbs(Alpha+intval(-DeltaA,DeltaA));
end;

global function DeltaLn1p(Alpha: interval; DeltaA: real): real;
var ResultSet: interval;
begin
   DeltaLn1p:= EpsLn1p *> MaxAbs(ln(alpha+1))
                +> (1 +> EpsLn1p) *> DeltaA
                /> (1 +< MinAbs(Alpha+intval(-DeltaA,DeltaA)));
   if EpsAriTest then begin  { Unterlaufwarnung }
      ResultSet:= ln( alpha + intval(-DeltaA,DeltaA) + 1 );
      if not (ResultSet >< UnflowRange) then begin
        write('  DeltaLn1p: Ergebnis im Unterlauf!  Weiter mit <Return> ');
        readln;
      end;
   end;  { Unterlaufwarnung }
end;

global function DeltaCoth(Alpha: interval; DeltaA: real): real;
begin
   DeltaCoth:= EpsCoth *> MaxAbs(coth(alpha))
                +> (1 +> Epscoth) *> DeltaA
                /> MinAbs( sqr( sinh( Alpha+intval(-DeltaA,DeltaA) ) ) );
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 BoundType }
{-------------------------------------------------------------------------}

global function sqrt(x: BoundType): BoundType;
var erg: BoundType;
begin
   erg.AbsErr    := DeltaSqrt(x.Enclosure, x.AbsErr); { abs. Fehlerschranke }
   erg.Enclosure := Sqrt(x.Enclosure);               { Werteeinschliessung }
   sqrt:= erg;
end;

global function arctan(x: BoundType): BoundType;
var erg: BoundType;
begin
   erg.AbsErr    := DeltaArctan(x.Enclosure, x.AbsErr); { abs. Fehlerschranke }
   erg.Enclosure := Arctan(x.Enclosure);               { Werteeinschliessung }
   arctan:= erg;
end;

global function exp(x: BoundType): BoundType;
var erg: BoundType;
begin
   erg.AbsErr    := DeltaExp(x.Enclosure, x.AbsErr);    { abs. Fehlerschranke }
   erg.Enclosure := Exp(x.Enclosure);                   { Werteeinschliessung }
   exp:= erg;
end;
```

```
global function expm1(x: BoundType): BoundType;
var erg: BoundType;
begin
   erg.AbsErr    := DeltaExpm1(x.Enclosure, x.AbsErr);  { abs. Fehlerschranke }
   erg.Enclosure := Exp(x.Enclosure)-1;                 { Werteeinschliessung }
$off
   erg.Enclosure := ffexpm1(x.Enclosure);
$on
   expm1:= erg;
end;

global function ln(x: BoundType): BoundType;
var erg: BoundType;
begin
   erg.AbsErr    := DeltaLn(x.Enclosure, x.AbsErr);     { abs. Fehlerschranke }
   erg.Enclosure := Ln(x.Enclosure);                    { Werteeinschliessung }
   ln:= erg;
end;

global function ln1p(x: BoundType): BoundType;
var erg: BoundType;
begin
   erg.AbsErr    := DeltaLn1p(x.Enclosure, x.AbsErr);   { abs. Fehlerschranke }
   erg.Enclosure := Ln(1+x.Enclosure);                  { Werteeinschliessung }
   ln1p:= erg;
end;

global function coth(x: BoundType): BoundType;
var erg: BoundType;
begin
   erg.AbsErr    := DeltaCoth(x.Enclosure, x.AbsErr);   { abs. Fehlerschranke }
   erg.Enclosure := coth(x.Enclosure);                  { Werteeinschliessung }
   coth:= erg;
end;

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

global operator := ( var res: BoundType; rhs: real);
begin
   res.Enclosure.inf := pred(rhs);
   res.Enclosure.sup := succ(rhs);
   res.AbsErr        := succ(abs(rhs)) *> EpsQuer ;
end;

global operator := ( var res: BoundType; rhs: interval);
begin
   res.Enclosure := rhs;
   res.AbsErr    := MaxAbs(rhs) *> EpsQuer;
end;

global function rnd_to_n ( rhs: real):BoundType;
```

```
begin
   rnd_to_n.Enclosure.inf := pred(rhs);
   rnd_to_n.Enclosure.sup := succ(rhs);
   rnd_to_n.AbsErr        := succ(abs(rhs)) *> Eps53; { round-to-nearest !!! }
end;

global function Exact( rhs: real):BoundType;
begin
   Exact.Enclosure := rhs;
   Exact.AbsErr    := 0.0;
end;

global function Exact( rhs: interval):BoundType;
begin
   Exact.Enclosure := rhs;
   Exact.AbsErr:= 0.0;
end;

global operator + (x, y: BoundType) erg: BoundType;
begin
  erg.AbsErr    := DeltaAdd(x.Enclosure, y.Enclosure, x.AbsErr, y.AbsErr);
  erg.Enclosure := x.Enclosure + y.Enclosure;
end;

global operator + (x: integer; y: BoundType) erg: BoundType;
begin
  erg.AbsErr    := DeltaAdd(intval(x), y.Enclosure, 0.0, y.AbsErr);
  erg.Enclosure := x + y.Enclosure;
end;

global operator - (x: BoundType) erg: BoundType;
begin
  erg.AbsErr    := x.AbsErr;
  erg.Enclosure := -x.Enclosure;
end;

global operator - (x, y: BoundType) erg: BoundType;
begin
  erg.AbsErr    := DeltaAdd(x.Enclosure, -y.Enclosure, x.AbsErr, y.AbsErr);
  erg.Enclosure := x.Enclosure - y.Enclosure;
end;

global operator - (x: integer; y: BoundType) erg: BoundType;
begin
  erg.AbsErr    := DeltaAdd(intval(x), -y.Enclosure, 0.0, y.AbsErr);
  erg.Enclosure := x - y.Enclosure;
end;

global operator - (x: BoundType; y: integer) erg: BoundType;
begin
  erg.AbsErr    := DeltaAdd(x.Enclosure, intval(-y), x.AbsErr, 0.0);
  erg.Enclosure := x.Enclosure - y;
end;
```

```
global operator * (x, y: BoundType) erg: BoundType;
begin
  erg.AbsErr     := DeltaMul(x.Enclosure, y.Enclosure, x.AbsErr, y.AbsErr);
  erg.Enclosure  := x.Enclosure * y.Enclosure;
end;

global operator * (x: integer; y: BoundType) erg: BoundType;
begin
  erg.AbsErr     := DeltaMul(intval(x), y.Enclosure, 0.0, y.AbsErr);
  erg.Enclosure  := x * y.Enclosure;
end;

global operator / (x, y: BoundType) erg: BoundType;
begin
  erg.AbsErr     := DeltaDiv(x.Enclosure, y.Enclosure, x.AbsErr, y.AbsErr);
  erg.Enclosure  := x.Enclosure / y.Enclosure;
end;

global operator / (x: integer; y: BoundType)erg: BoundType;
begin
  erg.AbsErr    := DeltaDiv(intval(x), y.Enclosure, 0.0, y.AbsErr);
  erg.Enclosure := intval(x) / y.Enclosure;
end;

{------------------------------------------------------------------------------}

begin { Initialisierungen }
  UnflowRange:= intval(-MinReal, MinReal);
  EpsAriTest := false;
  EpsQuer:= Eps52;
  EpsArctan := (> 1.358774060669230E-015);
  DelArctan := dminreal;
  EpsExp    := (> 2.3580e-16);
  DelExp    := dminreal;
  EpsExpm1  := (> 2.5926e-16);
  DelExpm1  := dminreal;
  EpsLn     := (> 2.9398e-16);
  EpsLn1p   := (> 2.5082e-16);
  EpsCoth   := (> 8.3253e-16);
end.  { module abs_ari }
{------------------------------------------------------------------------------}
```

Anhang C

C++ – Schnittstelle der ANSI-C Version

```
/****************************************************************/
/*                                                              */
/*   fi_lib  --- A fast interval library (Version 1.1)          */
/*                                                              */
/*  Authors:                                                    */
/*  --------                                                    */
/*  Werner Hofschuster, Walter Kraemer                          */
/*  Institut fuer Wissenschaftliches Rechnen                    */
/*           und  Mathematische Modellbildung (IWRMM)    and    */
/*  Institut fuer Angewandte Mathematik                         */
/*  Universitaet Karlsruhe (TH)                                 */
/*                                                              */
/*  Disclaimer:                                                 */
/*  -----------                                                 */
/*  This Library is distributed in the hope that it will be useful,  */
/*  but WITHOUT ANY WARRANTY; without even the implied warranty of   */
/*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.        */
/*  Neither the Institut fuer Angewandte Mathematik nor any other    */
/*  facility are responsible for this software. This software does   */
/*  not grant for anything and you can't make the autors responsible */
/*  for any possible damages or errors!                         */
/*                                                              */
/*  Copyright:                                                  */
/*  ----------                                                  */
/*  This package is proprietary of the authors. It may be freely     */
/*  distributed but must not be changed or used for commercial       */
/*  purposes without permission of the authors.                 */
/*                                                              */
/****************************************************************/

#include "fi_lib.h"
#include <iostream.h>
#include <iomanip.h>

/* ------------------------------------------------------------ */
/* --- assignment                                           --- */
```

```
/* ---------------------------------------------------------------- */

interval _interval (double x) {
  interval w;

  w.INF = x;
  w.SUP = x;

  return w;
}

interval _interval (double x, double y) {
  interval w;

  if (x > y) {
    cout << "Error: Invalid arguments in function _interval" << endl;
  }
  w.INF = x;
  w.SUP = y;

  return w;
}

/* ---------------------------------------------------------------- */
/* --- IO (input/output)                                        --- */
/* ---------------------------------------------------------------- */

istream& operator>> (istream& is, interval& a) {
  double help, ioconst;

  ioconst = (1e17-1)*1e27;

  is >> help;
  if ((help<ioconst) || (help>ioconst))
    a.INF = q_pred(q_pred(help));
  else
    a.INF = q_pred(help);

  is >> help;
  if ((help<ioconst) || (help>ioconst))
    a.SUP = q_succ(q_succ(help));
  else
    a.SUP = q_succ(help);

  return is;
}

ostream& operator<< (ostream& os, interval a) {
  interval help;

  help.INF = q_pred(q_pred(a.INF));
  help.SUP = q_succ(q_succ(a.SUP));

  long int aktform = cout.flags();
```

```
  os << "[" << setprecision(15) << setw(23) << setiosflags(ios::scientific);
  os <<  help.INF;
  cout.flags(aktform);
  os << "," << setprecision(15) << setw(23) << setiosflags(ios::scientific);
  os << help.SUP;
  cout.flags(aktform);
  os << " ]";

  return os;
}

/* ------------------------------------------------------------------- */
/* --- interval arithmetic (basic operations)                      --- */
/* ------------------------------------------------------------------- */

interval operator+ (interval a, interval b) {
  return add_ii (a, b);
}

interval operator+ (interval a, double b) {
  return add_id (a, b);
}

interval operator+ (double a, interval b) {
  return add_di (a, b);
}

interval operator+ (interval a) {
  return a;
}

interval operator- (interval a, interval b) {
  return sub_ii (a, b);
}

interval operator- (interval a, double b) {
  return sub_id (a, b);
}

interval operator- (double a, interval b) {
  return sub_di (a, b);
}

interval operator- (interval a) {
  return _interval (-a.SUP, -a.INF);
}

interval operator* (interval a, interval b) {
  return mul_ii (a, b);
}

interval operator* (interval a, double b) {
  return mul_id (a, b);
}
```

```
interval operator* (double a, interval b) {
  return mul_di (a, b);
}

interval operator/ (interval a, interval b) {
  return div_ii (a, b);
}

interval operator/ (interval a, double b) {
  return div_id (a, b);
}

interval operator/ (double a, interval b) {
  return div_di (a, b);
}

/* ------------------------------------------------------------------ */
/* --- interval arithmetic (logical operations)                   --- */
/* ------------------------------------------------------------------ */

int operator== (interval a, interval b) {
  return ieq_ii (a, b);
}

int operator== (interval a, double b) {
return ieq_ii (a, _interval(b));
}

interval operator| (interval a, interval b) {
  return hull (a, b);
}

int operator<= (double a, interval b) {
  return in_di (a, b);
}

int in (double a, interval b) {
return in_di (a, b);
}

int in (interval a, interval b) {
return in_ii (a, b);
}

interval operator& (interval a, interval b) {
  return intsec (a, b);
}

int operator< (interval a, interval b) {
  return in_ii (a, b);
}

int operator< (double a, interval b) {
```

```
  if (b.INF<a && a<b.SUP) return 1; else return 0;
}

int operator>= (interval a, double b) {
  if (a.INF<=b && b<=a.SUP) return 1; else return 0;
}

int operator> (interval a, double b) {
  if (a.INF<b && b<a.SUP) return 1; else return 0;
}

int operator!= (interval a, interval b) {
  if (!(a.INF==b.INF && a.SUP==b.SUP)) return 1; else return 0;
}

int operator<= (interval a, interval b) {
  if (b.INF<=a.INF && a.SUP<=b.SUP) return 1; else return 0;
}

int operator>= (interval a, interval b) {
  if (b.INF>=a.INF && a.SUP>=b.SUP) return 1; else return 0;
}

int operator> (interval a, interval b) {
  if (b.INF>a.INF && a.SUP>b.SUP) return 1; else return 0;
}

/* ------------------------------------------------------------------- */
/* --- utilities, mid, diam, ...                                   --- */
/* ------------------------------------------------------------------- */

double inf (interval a) {
  return a.INF;
}

double sup (interval a) {
  return a.SUP;
}

double mid (interval a) {
  return q_mid (a);
}

int disjoint (interval a, interval b) {
  return dis_ii (a, b);
}

double diam (interval a) {
  return q_diam (a);
}


double drel (interval a)
{ if ((a.SUP<=-q_minr) || (q_minr<=a.INF)) {
```

```
    if (a.INF > 0) return diam(a)/a.INF;
    else           return diam(a)/(-a.SUP);
  } else {
    return diam(a);
  }
}


interval blow ( interval x, double eps) {
interval y;
y = (1.0 +eps) * x - eps*x;
return (_interval(q_pred(y.INF),q_succ(y.SUP)));
}

/* ------------------------------------------------------------------- */
/* --- interval arithmetic (elementary functions)                  --- */
/* ------------------------------------------------------------------- */

interval exp (interval a) {
  return j_exp (a);
}

interval expm (interval a) {
  return j_expm (a);
}

interval sinh (interval a) {
  return j_sinh (a);
}

interval cosh (interval a) {
  return j_cosh (a);
}

interval coth (interval a) {
  return j_coth (a);
}

interval tanh (interval a) {
  return j_tanh (a);
}

interval log (interval a) {
  return j_log (a);
}

interval ln (interval a) {
  return j_log (a);
}

interval lg1p (interval a) {
  return j_lg1p (a);
}
```

```
interval sqrt (interval a) {
  return j_sqrt (a);
}

interval sqr (interval a) {
  return j_sqr (a);
}

interval asnh (interval a) {
  return j_asnh (a);
}

interval asinh (interval a) {
  return j_asnh (a);
}

interval acsh (interval a) {
  return j_acsh (a);
}

interval acosh (interval a) {
  return j_acsh (a);
}

interval acth (interval a) {
  return j_acth (a);
}

interval acoth (interval a) {
  return j_acth (a);
}

interval atnh (interval a) {
  return j_atnh (a);
}

interval atanh (interval a) {
  return j_atnh (a);
}

interval asin (interval a) {
  return j_asin (a);
}

interval acos (interval a) {
  return j_acos (a);
}

interval acot (interval a) {
  return j_acot (a);
}

interval atan (interval a) {
  return j_atan (a);
```

```
}

interval sin (interval a) {
  return j_sin (a);
}

interval cos (interval a) {
  return j_cos (a);
}

interval cot (interval a) {
  return j_cot (a);
}

interval tan (interval a) {
  return j_tan (a);
}

interval exp2 (interval a) {
  return j_exp2 (a);
}

interval ex10 (interval a) {
  return j_ex10 (a);
}

interval log2 (interval a) {
  return j_log2 (a);
}

interval lg10 (interval a) {
  return j_lg10 (a);
}
```

Anhang D

Erste Anwendungsbeispiele

In den folgenden Unterabschnitten finden sich Beispiele zu:

- Intervallmäßiges Hornerschema
- Aufruf einer Intervallfunktion
- Verifizierte Nullstellenbestimmung mit Bisektionsverfahren
- Einschließung aller Nullstellen mit dem erweiterten Intervall-Newton-Verfahren

D.1 Intervallgrundarithmetik in ANSI-C und C++

```
/*****************************************************************/
/*                                                               */
/*   Example: hornerc.c  (Interval Horner's scheme in ANSI-C)    */
/*          (For copyright and info's see file "fi_lib.h")       */
/*                                                               */
/*****************************************************************/

#include<stdio.h>
#include<string.h>
#include"fi_lib.h"     /* use library fi_lib */

/* --- main program ------------------------------------------------- */

int main()
{
  interval coeff[16];
  interval p, x, res;
  int i;

  /* --- Computation of the coefficients ------------------------------ */
  coeff[0] = eq_id(1.0);
  coeff[1] = eq_id(1.0);
  p = eq_id(1.0);
  for (i=2; i<=15; i++){
    p = mul_id(p,(double)i);
```

```
    coeff[i] = div_di(1.0,p);
  }

  printf("Interval Horner's scheme in ANSI-C with fi_lib\n");
  printf("=============================================\n\n");

  printf("Computation of the polynom (sum_i=0^15 1/i! x^i), \n");
  printf(" that means the first terms of the taylor series \n");
  printf(" of the exponential function.\n\n");
  printf("Enclosures for the polynom coefficients:\n");

  for (i=0; i<=15; i++){
    printf("   coeff[%2d] = ",i);
    printInterval(coeff[i]);
    printf("\n");
  }
  printf("\n");

  printf("Now, you can choose an interval argument (e.g. 'x = 1 1', \n");
  printf("'x = 1.01 1.02', 'x= -1 -1' or 'x= -2.0 -1.99' ...): \n\n");
  printf("x = ");
  x = scanInterval();

  /* --- interval Horner's scheme ---------------------------------------- */
  res = coeff[15];
  for (i=14; i>=0; i--) {
    res = mul_ii(res,x);
    res = add_ii(res,coeff[i]);
  }

  printf("Result: ");
  printInterval(res);
  printf("\n");

  return 0;
}



/*****************************************************************/
/*                                                               */
/*   Example: hornercpp.C  (Interval Horner's scheme in C++)     */
/*          (For copyright and info's see file "fi_lib.h")       */
/*                                                               */
/*****************************************************************/

#include<stdio.h>
#include<string.h>
#include"interval.hpp"    /* use library fi_lib with C++ - interface */

/* --- main program ------------------------------------------------- */

int main()
{
```

```
  interval coeff[16];
  interval p, x, res;
  int i;

  /* --- Computation of the coefficients ------------------------------- */
  coeff[0] = _interval(1.0);
  coeff[1] = _interval(1.0);
  p = _interval(1.0);
  for (i=2; i<=15; i++){
    p = p*(double)i;
    coeff[i] = 1.0/p;
  }

  cout << "Interval Horner's scheme in C++ with fi_lib" << endl;
  cout << "==========================================" << endl << endl;

  cout << "Computation of the polynom (sum_i=0^15 1/i! x^i)," << endl;
  cout << "that means the first terms of the taylor series" << endl;
  cout << "of the exponential function." << endl << endl;
  cout << "Enclosures for the polynom coefficients:" << endl;

  for (i=0; i<=15; i++)
    cout << "   coeff[" << setw(2) << i << "] = " << coeff[i] << endl;
  cout << endl;

  cout << "Now, you can choose an interval argument (e.g. 'x = 1 1', " << endl;
  cout << "'x = 1.01 1.02', 'x= -1 -1' or 'x= -2.0 -1.99' ...): " << endl;
  cout << endl;
  cout << "x = ";
  cin >> x;

  /* --- interval Horner's scheme -------------------------------------- */
  res = coeff[15];
  for (i=14; i>=0; i--)
    res = res*x + coeff[i];

  cout << "Result: " << res << endl;

  return 0;
}
```

D.2 Aufruf von Standardfunktionen

```
/*****************************************************************/
/*                                                               */
/*   Example: comp_exp.c  (Computation of the exponential function)  */
/*          (For copyright and info's see file "fi_lib.h")       */
/*                                                               */
/*****************************************************************/

#include<stdio.h>
```

```
#include<string.h>
#include"fi_lib.h"     /* use library fi_lib */

/* --- main program ------------------------------------------------------ */

int main()
{
  interval x;

  printf("\n");
  printf("Computation of the exponential function in ANSI-C with fi_lib\n");
  printf("=============================================================\n\n");

  printf("Insert an interval argument (e.g. 'x = 1 1' or 'x = 1.01 1.02') \n");
  printf("x = ");
  x = scanInterval();

  printf("Argument x = ");
  printInterval(x);
  printf("\n");

  printf("    exp(x) = ");
  printInterval( j_exp(x) );
  printf("\n\n");

  return 0;
}



/*************************************************************************/
/*                                                                       */
/*   Example: comp_sin.c  (Computation of the sine function)             */
/*          (For copyright and info's see file "fi_lib.h")               */
/*                                                                       */
/*************************************************************************/

#include<stdio.h>
#include<string.h>
#include"fi_lib.h"     /* use library fi_lib */

/* --- main program ------------------------------------------------------ */

int main()
{
  interval x;

  printf("\n");
  printf("Computation of the sine function in ANSI-C with fi_lib\n");
  printf("======================================================\n\n");

  printf("Insert an interval argument (e.g. 'x = 1 1' or 'x = 1.01 1.02') \n");
  printf("x = ");
  x = scanInterval();
```

```
  printf("Argument x = ");
  printInterval(x);
  printf("\n");

  printf("    sin(x) = ");
  printInterval( j_sin(x) );
  printf("\n\n");

  return 0;
}
```

D.3 Einschluß von Nullstellen (Bisektionsverfahren)

```
/*****************************************************************/
/*                                                               */
/*   Example: bisection.C   (finding zeros with bisection-method)*/
/*         (For copyright and info's see file "fi_lib.h")        */
/*                                                               */
/*****************************************************************/

#include "interval.hpp"
#include "iostream.h"

// ---------------------------------------------------------------
// ---   Test function f1 = e^(-3x) - (sin x)^3                ---
// ---------------------------------------------------------------

interval f1(interval x) {
  return ( exp(-3.0*x) - sin(x)*sin(x)*sin(x) );
}

// ---------------------------------------------------------------
// ---   Test function f2 = - \sum_k=1^5 (k*sin((k+1)x+k))     ---
// ---------------------------------------------------------------

interval f2(interval x) {
  int k;
  interval res=_interval(0.0);

  for (k=1; k<=5; k++)
    res = res - k * sin( (k+1)*x + k );
  return ( res );
}

// ---------------------------------------------------------------
// ---   Test function f3 = x^3 - 2x^2 +  x                    ---
// ---------------------------------------------------------------

interval f3(interval x) {
  return ( x*sqr(x) - 2*sqr(x) + x);
}
```

```
// ------------------------------------------------------------------
// ---   Test function f4 = 0                                     ---
// ------------------------------------------------------------------

interval f4(interval x) {
  return _interval(0.0);
}

// ------------------------------------------------------------------
// ---   Test function f5 = 1                                     ---
// ------------------------------------------------------------------

interval f5(interval x) {
  return _interval(1.0);
}

// ------------------------------------------------------------------
// ---   Test function f6 = (sin x)^2 - (1-cos 2x)/2 = 0          ---
// ------------------------------------------------------------------

interval f6(interval x) {
  return ( sqr(sin(x)) - ( 1.0 - cos(2*x) ) / 2.0 );
}

// ------------------------------------------------------------------
// ---   Test function f7 = x^3-x^2-17x-15 = (x-5)(x+1)(x+3)      ---
// ------------------------------------------------------------------

interval f7(interval x) {
  return( x*sqr(x) - sqr(x) - 17*x - 15 );
}


// ------------------------------------------------------------------
// ---   Initialisation, data type list                           ---
// ------------------------------------------------------------------

const int MaxDepth = 10000;  // Maximum number of bisection steps

struct list                  // Data type for resulting list
{
  interval intval;
  list* next;

  list() {intval=_interval(0.0); next=NULL;}
};


// ------------------------------------------------------------------
// ---   a special sign function (for verification test)          ---
// ------------------------------------------------------------------

int VZ( interval (*fct) (interval), interval x) {
```

```
  return ( sup( (*fct)(_interval(inf(x))) * (*fct)(_interval(sup(x))) ) < 0 );
}

// ------------------------------------------------------------------
// ---   function PrintZeros (output and verfication)              ---
// ------------------------------------------------------------------

void PrintZeros( interval (*fct) (interval), list* reslist, int Depth) {

  int k=0;

  if (reslist==NULL)
     cout << "Function contains no zeros in the search interval!" << endl;
  else {
    cout << "Ranges for zeros:" << endl;
    while (reslist != NULL) {
       cout << " k=" << ++k << " " << reslist->intval << " verified: ";
       if (VZ((*fct),reslist->intval)) cout << "Yes" << endl;
       else                            cout << "No" << endl;
       reslist=reslist->next;
    }
  }
  cout << "Number of bisection steps: " << Depth << endl << endl;
}

// ------------------------------------------------------------------
// ---   function Bisect (bisection-method)                        ---
// ------------------------------------------------------------------

void Bisect( interval (*fct) (interval), interval x, double eps,
                                        list* &reslist, int &Depth) {

  interval fx, x1, x2;
  double infx, supx, m;
  int k;
  int absorbed;
  list* pointer;
  pointer = reslist;

  Depth++;
  fx = (*fct)(x);

  if (0 <= fx) {
    infx = inf(x);
    supx = sup(x);
    m = mid(x);
    if ((diam(x)<eps) || (m==infx) || (m==supx) || (Depth > MaxDepth) ) {
      k=1;
      absorbed=0;
      //- Try to absorb x by an already computed element of resulting list -
      if (reslist != NULL) while ((pointer != NULL) && (!absorbed)) {
        absorbed = (    (inf(pointer->intval)<=sup(x) &&
                                              sup(x)<=sup(pointer->intval))
                    || (inf(pointer->intval)<=inf(x) &&
```

```
                                                inf(x)<=sup(pointer->intval))
                    || (inf(x)<=sup(pointer->intval) &&
                                                sup(pointer->intval)<=sup(x))
                    || (inf(x)<=inf(pointer->intval) &&
                                                inf(pointer->intval)<=sup(x))
                  );
        if (absorbed) pointer->intval = ((pointer->intval) | x);
        pointer = pointer -> next;
      }

      if (!absorbed) {   // Store x in the resulting list
         pointer=reslist;
         if (reslist != NULL) {
           while (pointer->next != NULL) pointer = pointer->next;
           list* insert = new list;
           pointer->next = insert;
           insert->next = NULL;
           insert->intval = x;
         } else {
           reslist = new list;
           reslist->next = NULL;
           reslist->intval = x;
         }
      }
      if (Depth> MaxDepth)
         cout << "Maximal number of bisection steps is reached!" << endl;
    } else {
      x1 = _interval(infx,m);
      Bisect( (*fct), x1, eps, reslist, Depth);
      x2 = _interval(m,supx);
      Bisect( (*fct), x2, eps, reslist, Depth);
    }
  }
}

// ------------------------------------------------------------------------
// ---   function PrintAllZeros (root finding and output)           ---
// ------------------------------------------------------------------------

void PrintAllZeros( interval (*fct) (interval), interval x, double eps) {

  int Depth = 0;
  list* reslist;
  reslist = NULL;
  Bisect( (*fct), x, eps, reslist, Depth );
  PrintZeros( (*fct), reslist, Depth );
}

// ------------------------------------------------------------------------
// ---   function main                                                ---
// ------------------------------------------------------------------------

int main() {
```

```
  double eps;
  interval y;

  cout << "Bisection method in C++ with fi_lib" << endl;
  cout << "===================================" << endl << endl;


  cout << "Toleranz (relativ): (e.g. 'eps = 1e-3' or 'eps = 1e-8') \n eps = ";
  cin >> eps;
  cout << endl;

  cout << "Test function f1 = e^(-3x) - (sin x)^3:" << endl;
  y = _interval(  0, 20);
  cout << "Search interval: " << y << endl;
  PrintAllZeros( f1, y, eps);

  cout << "Test function f2 = - sum_k=1^5 (k*sin((k+1)x+k)):" << endl;
  y = _interval(  -5, 5);
  cout << "Search interval: " << y << endl;
  PrintAllZeros( f2, y, eps);

  cout << "Test function f3 = x^3 - 2x^2 +  x = x(x-1)^2:" << endl;
  y = _interval(  -1, 2);
  cout << "Search interval: " << y << endl;
  PrintAllZeros( f3, y, eps);

  cout << "Test function f4 = 0:" << endl;
  y = _interval(  1, 2);
  cout << "Search interval: " << y << endl;
  PrintAllZeros( f4, y, eps);

  cout << "Test function f5 = 1:" << endl;
  y = _interval(  1, 2);
  cout << "Search interval: " << y << endl;
  PrintAllZeros( f5, y, eps);

  cout << "Test function f6 = (sin x)^2 - (1-cos 2x)/2 = 0:" << endl;
  y = _interval(  -1, 1);
  cout << "Search interval: " << y << endl;
  PrintAllZeros( f6, y, eps);

  cout << "Test function f7 = x^3-x^2-17x-15 = (x-5)(x+1)(x+3):" << endl;
  y = _interval( -10, 10);
  cout << "Search interval: " << y << endl;
  PrintAllZeros( f7, y, eps);

  return 0;
}



/* --------------------------  output  ------------------------------

Toleranz (relativ): (e.g. 'eps = 1e-3' or 'eps = 1e-8')
```

```
 eps = 1e-3

Test function f1 = e^(-3x) - (sin x)^3:
Search interval: [-9.881312916824931e-324,  2.000000000000001e+01 ]
Ranges for zeros:
 k=1 [  5.883789062499998e-01,  5.889892578125002e-01 ] verified: Yes
 k=2 [  3.096313476562499e+00,  3.096923828125001e+00 ] verified: Yes
 k=3 [  6.284790039062498e+00,  6.285400390625002e+00 ] verified: Yes
 k=4 [  9.424438476562496e+00,  9.425048828125004e+00 ] verified: Yes
 k=5 [  1.256591796875000e+01,  1.256652832031250e+01 ] verified: Yes
 k=6 [  1.570739746093750e+01,  1.570800781250000e+01 ] verified: Yes
 k=7 [  1.884948730468749e+01,  1.885009765625001e+01 ] verified: Yes
Number of bisection steps: 191

Test function f2 = - sum_k=1^5 (k*sin((k+1)x+k)):
Search interval: [ -5.000000000000002e+00,  5.000000000000002e+00 ]
Ranges for zeros:
 k=1 [ -4.946289062500002e+00, -4.945068359374998e+00 ] verified: Yes
 k=2 [ -4.488525390625002e+00, -4.486083984374998e+00 ] verified: Yes
 k=3 [ -3.977050781250001e+00, -3.975219726562499e+00 ] verified: Yes
 k=4 [ -3.505859375000001e+00, -3.504028320312499e+00 ] verified: Yes
 k=5 [ -3.015136718750001e+00, -3.013916015624999e+00 ] verified: Yes
 k=6 [ -2.511596679687501e+00, -2.510375976562499e+00 ] verified: Yes
 k=7 [ -2.055664062500001e+00, -2.054443359374999e+00 ] verified: Yes
 k=8 [ -1.455078125000000e+00, -1.454467773437500e+00 ] verified: Yes
 k=9 [ -7.861328125000002e-01, -7.855224609374998e-01 ] verified: Yes
 k=10 [ -1.470947265625001e-01, -1.464843749999999e-01 ] verified: Yes
 k=11 [  3.521728515624999e-01,  3.527832031250001e-01 ] verified: Yes
 k=12 [  8.209228515624998e-01,  8.221435546875002e-01 ] verified: Yes
 k=13 [  1.336669921875000e+00,  1.338500976562500e+00 ] verified: Yes
 k=14 [  1.795043945312500e+00,  1.796875000000000e+00 ] verified: Yes
 k=15 [  2.305908203124999e+00,  2.308349609375001e+00 ] verified: Yes
 k=16 [  2.777099609374999e+00,  2.778930664062501e+00 ] verified: Yes
 k=17 [  3.268432617187499e+00,  3.269653320312501e+00 ] verified: Yes
 k=18 [  3.771362304687499e+00,  3.772583007812501e+00 ] verified: Yes
 k=19 [  4.227905273437498e+00,  4.228515625000002e+00 ] verified: Yes
 k=20 [  4.827880859374998e+00,  4.828491210937502e+00 ] verified: Yes
Number of bisection steps: 811

Test function f3 = x^3 - 2x^2 +  x = x(x-1)^2:
Search interval: [ -1.000000000000000e+00,  2.000000000000001e+00 ]
Ranges for zeros:
 k=1 [ -2.441406250000001e-04,  4.882812500000002e-04 ] verified: Yes
 k=2 [  9.460449218749998e-01,  1.054443359375000e+00 ] verified: No
Number of bisection steps: 709

Test function f4 = 0:
Search interval: [  9.999999999999998e-01,  2.000000000000001e+00 ]
Ranges for zeros:
 k=1 [  9.999999999999998e-01,  2.000000000000001e+00 ] verified: No
Number of bisection steps: 2047

Test function f5 = 1:
Search interval: [  9.999999999999998e-01,  2.000000000000001e+00 ]
```

```
Function contains no zeros in the search interval!
Number of bisection steps: 1

Test function f6 = (sin x)^2 - (1-cos 2x)/2 = 0:
Search interval: [ -1.000000000000000e+00,  1.000000000000000e+00 ]
Ranges for zeros:
 k=1 [ -1.000000000000000e+00,  1.000000000000000e+00 ] verified: No
Number of bisection steps: 4095

Test function f7 = x^3-x^2-17x-15 = (x-5)(x+1)(x+3):
Search interval: [ -1.000000000000000e+01,  1.000000000000000e+01 ]
Ranges for zeros:
 k=1 [ -3.001098632812501e+00, -2.999267578124999e+00 ] verified: Yes
 k=2 [ -1.000366210937500e+00, -9.991455078124998e-01 ] verified: Yes
 k=3 [  4.999389648437498e+00,  5.000610351562502e+00 ] verified: Yes
Number of bisection steps: 179

------------------------------------------------------------------------*/
```

D.4 Erweitertes Intervall-Newton-Verfahren

```
/*****************************************************************/
/*                                                               */
/*   Example: xinterval.hpp  (Extended interval arithmetic)      */
/*       (For copyright and info's see file "fi_lib.h")          */
/*                                                               */
/*****************************************************************/

#include"interval.hpp"

typedef enum { Finite, PlusInfty, MinusInfty, Double, Empty } KindType;

//---------------------------------------------------------------------------

class xinterval {                  // Extended intervals according
  public:                          // to the above definition
    KindType  kind;                //-----------------------------
    double    inf, sup;

    xinterval ( );
    xinterval ( const KindType&, const double&, const double& );
    xinterval ( const xinterval& );

    xinterval& operator= ( const xinterval& );

    friend xinterval operator% (const interval& A, const interval& B );
    friend xinterval operator- ( const double a, const xinterval B );
    friend interval* operator& ( interval X, const xinterval Y );
};

//---------------------------------------------------------------------------
```

```
interval EmptyIntval ( );          // Irregular (empty) interval

//---------------------------------------------------------------------------

interval EmptyIntval ( )           // Irregular (empty) interval
{                                  //--------------------------
  interval x;
  x.INF = 999999999.0;
  x.SUP = -999999999.0;
  return x;
}

xinterval::xinterval ( )
{
  kind = Finite;
  inf  = 0.0;
  sup  = 0.0;
}

xinterval::xinterval ( const KindType& k, const double& i, const double& s )
{
  kind = k;
  inf  = i;
  sup  = s;
}

xinterval::xinterval ( const xinterval& a )
{
  kind = a.kind;
  inf  = a.inf;
  sup  = a.sup;
}

xinterval& xinterval::operator= ( const xinterval& a )
{
  kind = a.kind;
  inf  = a.inf;
  sup  = a.sup;
  return *this;
}

//---------------------------------------------------------------------------
// Extended interval division 'A / B' where 0 in 'B' is allowed.
//---------------------------------------------------------------------------

xinterval operator% (const interval& A, const interval& B )
{
  interval  c;
  xinterval Q;

  if ( in(0.0, B) ) {
    if ( in(0.0, A) ) {
      Q.kind = Double;                    // Q = [-oo,+oo] = [-oo,0] v [0,+oo]
      Q.sup  = 0.0;                       //-----------------------------------
```

```
      Q.inf  = 0.0;
    }
    else if ( B == 0.0 ) {                                          // Q = [/]
      Q.kind = PlusInfty;                                           //--------
      Q.inf  = q_pred(sup(A)/inf(B));
    }
    else if ( (sup(A) < 0.0) && (sup(B) == 0.0) ) {         // Q = [Q.inf,+oo]
      Q.kind = PlusInfty;                                           //----------------
      Q.inf  = q_pred(sup(A)/inf(B));
    }
    else if ( (sup(A) < 0.0) && (inf(B) < 0.0) && (sup(B) > 0.0) ) {
      Q.kind = Double;                          // Q = [-oo,Q.sup] v [Q.inf,+oo]
      Q.sup  = q_succ(sup(A)/sup(B));           //------------------------------
      Q.inf  = q_pred(sup(A)/inf(B));
    }
    else if ( (sup(A) < 0.0) && (inf(B) == 0.0) ) {         // Q = [-oo,Q.sup]
      Q.kind = MinusInfty;                                          //----------------
      Q.sup  = q_succ(sup(A)/sup(B));
    }
    else if ( (inf(A) > 0.0) && (sup(B) == 0.0) ) {         // Q = [-oo,Q.sup]
      Q.kind = MinusInfty;                                          //----------------
      Q.sup  = q_succ(inf(A)/inf(B));
    }
    else if ( (inf(A) > 0.0) && (inf(B) < 0.0) && (sup(B) > 0.0) ) {
      Q.kind = Double;                          // Q = [-oo,Q.sup] v [Q.inf,+oo]
      Q.sup  = q_succ(inf(A)/inf(B));           //------------------------------
      Q.inf  = q_pred(inf(A)/sup(B));
    }
    else { // if ( (Inf(A) > 0.0) && (Inf(B) == 0.0) )
      Q.kind = PlusInfty;                                           // Q = [Q.inf,+oo]
      Q.inf  = q_pred(inf(A)/sup(B));                               //----------------
    }
  } // in(0.0,B)
  else {  // !in(0.0,B)
    c = A / B;                                                  // Q = [C.inf,C.sup]
    Q.kind = Finite;                                            //------------------
    Q.inf  = inf(c);
    Q.sup  = sup(c);
  }

  return Q;
}

//---------------------------------------------------------------------------
// Subtraction of an extended interval 'B' from a double value 'a'.
//---------------------------------------------------------------------------

xinterval operator- ( const double a, const xinterval B )
{
  xinterval D;

  switch (B.kind) {
    case Finite     : D.kind = Finite;                          // D = [D.inf,D.sup]
                      D.inf  = q_pred(a-B.sup);                 //------------------
```

```
                    D.sup  = q_succ(a-B.inf);
                    break;
    case PlusInfty  : D.kind = MinusInfty;                    // D = [inf,+oo]
                    D.sup  = q_succ(a-B.inf);                 //--------------
                    break;
    case MinusInfty : D.kind = PlusInfty;                     // D = [-oo,sup]
                    D.inf  = q_pred(a-B.sup);                 //--------------
                    break;
    case Double     : D.kind = Double;        // D = [-oo,D.sup] v [D.inf,+oo]
                    D.inf  = q_pred(a-B.sup); //----------------------------
                    D.sup  = q_succ(a-B.inf);
                    if (D.inf < D.sup) D.inf = D.sup;
                    break;
    case Empty      : D.kind = Empty;                                 // D = [/]
                    D.inf  = q_pred(a-B.sup);                       //--------
                    break;
  } // switch
  return D;
}

//---------------------------------------------------------------------------
// Intersection of an interval 'X' and an extended interval 'Y'. The result
// is given as a pair (vector) of intervals, where one or both of them can
// be empty intervals.
//---------------------------------------------------------------------------

interval* operator& ( interval X, const xinterval Y )
{
  interval H;
  interval* IS = new interval[3];

  IS[1] = EmptyIntval();
  IS[2] = EmptyIntval();

  switch (Y.kind) {
    case Finite     : // [X.inf,X.sup] & [Y.inf,Y.sup]
                      //------------------------------
                      H = _interval(Y.inf,Y.sup);
                      if ( !disjoint(X,H) ) IS[1] = X & H;

                      break;
    case PlusInfty  : // [X.inf,X.sup] & [Y.inf,+oo]
                      //---------------------------
                      if (sup(X) >= Y.inf)
                        if (inf(X) > Y.inf)
                          IS[1] = X;
                        else
                          IS[1] = _interval(Y.inf,sup(X));

                      break;
    case MinusInfty : // [X.inf,X.sup] & [-oo,Y.sup]
                      //---------------------------
                      if (Y.sup >= inf(X))
                        if (sup(X)<Y.sup)
```

```
                        IS[1] = X;
                      else
                        IS[1] = _interval(inf(X),Y.sup);

                    break;
    case Double     : if ( (inf(X) <= Y.sup) && (Y.inf <= sup(X)) ) {
                        IS[1] = _interval(inf(X),Y.sup);    // X & [-oo,Y.sup]
                        IS[2] = _interval(Y.inf,sup(X));    // X & [Y.inf,+oo]
                      }
                      else if (Y.inf <= sup(X)) // [X.inf,X.sup] & [Y.inf,+oo]
                        if (inf(X) >= Y.inf)    //----------------------------
                          IS[1] = X;
                        else
                          IS[1] = _interval(Y.inf,sup(X));
                      else if (inf(X) <= Y.sup) // [X.inf,X.sup] & [-oo,Y.sup]
                        if (sup(X) <= Y.sup)    //----------------------------
                          IS[1] = X;
                        else
                          IS[1] = _interval(inf(X),Y.sup);

                    break;
    case Empty      : break;                            // [X.inf,X.sup] ** [/]
  } // switch                                           //---------------------

  return IS;
} // operator&



/*********************************************************************/
/*                                                                   */
/*   Example: xinewton.c  (Compute all zeros of a function)          */
/*       (For copyright and info's see file "fi_lib.h")              */
/*                                                                   */
/*********************************************************************/

#include "xinterval.hpp"
#include <iostream.h>
#include <stdio.h>         // for function sprintf

// ------------------------------------------------------------------
// ---   Initialisation, data type list                           ---
// ------------------------------------------------------------------

struct list                  // Data type for resulting list
{
  interval intval;
  int info;
  list* next;

  list() {intval=_interval(0.0); next=NULL;}
};

static int MaxZeroNo=10;
```

```
const  int NoError = 0,    // Error constants
       IllMaxZeroNo = 1,
       NotAllZeros = 2;
const  int MaxCount = 10000;

//---------------------------------------------------------------------------
// Definition of the function f(x) and the derivate df(x)
//     Test function f = cosh(x) + 10 * x^2 * sin(x)^2 - 34
//---------------------------------------------------------------------------

interval f(interval x) {
  return ( cosh(x)+10*sqr(x)*sqr(sin(x))-34 );
}

interval df(interval x) {
  return( sinh(x)+20*x*sqr(sin(x))+20*sqr(x)*sin(x)*cos(x) );
}

//---------------------------------------------------------------------------
// main function xinewton
//---------------------------------------------------------------------------

void xinewton(interval fy, interval y, double eps, int yUnique,
                              list* &zerolist, int& zerono)
{
   if (zerono > MaxZeroNo) return;
   interval* V;
   xinterval z;
   double c;
   interval ci;
   int i;
   int absorbed;
   list* pointer;
   pointer = zerolist;

   if (!in(0.0,fy)) return;

   c = mid(y);
   ci=_interval(c);
   z = c - f(ci)%df(y);
   V = y & z;

   if (V[1] == y) {                      // bisection
      V[1] = _interval(y.INF,c);
      V[2] = _interval(c,y.SUP);
   }

   if ( (V[1] != EmptyIntval() ) && (V[2] == EmptyIntval()))
      yUnique = yUnique || in(V[1],y);
   else
      yUnique = 0;

   for (i=1;i<=2;i++) {
      if (V[i] == EmptyIntval())  continue;
```

```
      if (drel(V[i])<=eps) {
         fy=f(V[i]);
         if (in(0.0,fy)) {
            zerono ++;
            if (zerono > MaxZeroNo) return;

            absorbed=0;
            pointer=zerolist;
            //- Try to absorb V[i] by an already computed element of list -
            if (zerolist != NULL) {
               while ((pointer != NULL) && (!absorbed)) {
                  absorbed = !( (sup(V[i])<inf(pointer->intval)) ||
                                (sup(pointer->intval)<inf(V[i]) ));
                  if (absorbed) {
                     pointer->intval = ((pointer->intval) | V[i]);
                     pointer->info = 0;
                     zerono --;
                  }
                  pointer = pointer -> next;
               }
            }

            if (!absorbed) {   // Store x in the resulting list
               pointer=zerolist;
               if (zerolist != NULL) {
                  while (pointer->next != NULL) pointer = pointer->next;
                  list* insert = new list;
                  pointer->next = insert;
                  insert->next = NULL;
                  insert->intval = V[i];
                  insert->info = yUnique;
               } else {
                  zerolist = new list;
                  zerolist->next = NULL;
                  zerolist->intval = V[i];
                  zerolist->info = yUnique;
               }
            }

         }
       } else {
         xinewton(f(V[i]),V[i],eps,yUnique,zerolist,zerono);
       }
   }
}

// ------------------------------------------------------------------
// ---   function AllZerosErrMsg (error message)                  ---
// ------------------------------------------------------------------

char* AllZerosErrMsg (int Err)
{
   static char Msg[80] = "";
```

```
   switch (Err) {
      case NoError:      break;
      case IllMaxZeroNo: sprintf(Msg,"Error: Parameter for maximum number of zeros must lie in 1,...
                         break;
      case NotAllZeros:  sprintf(Msg,"Warning: Not all zeros found due to the user limit of %l1d zer
                         break;
      default:           sprintf(Msg,"Error Code not defined");
   }

   return (Msg);
}

// ------------------------------------------------------------------------
// ---   function VerificationStep                                      ---
// ------------------------------------------------------------------------

static void VerificationStep(interval& y, int& unique)
{
   const int kmax= 10;
   interval yIn, yOld, fc, dfy;
   double c,eps;
   int k;

   k = 0;
   yIn = y;
   eps = 0.25;
   unique = 0;

   while (!unique && (k < kmax) ) {
      yOld = blow(y,eps);
      dfy = df(y);
      if (in(0.0,dfy)) break;
      k++;
      c = mid(yOld);
      fc=f(_interval(c));
      y = c - fc / dfy;
      if (disjoint(y, yOld)) break;
      unique = in(y,yOld);
      y = y & yOld;
      if (y == yOld) eps=eps*8.0;
   }
   if (!unique) y = yIn;
}

// ------------------------------------------------------------------------
// ---   function AllZeros (root finding and verification)              ---
// ------------------------------------------------------------------------

void AllZeros(interval Start, double Epsilon, list* &zerolist,
                  int& NumberOfZeros, int& Err, int MaxNumberOfZeros) {

   double MinEpsilon;
   list* pointer;
```

```
   if (1<=MaxNumberOfZeros && MaxNumberOfZeros <= MaxCount) {
      MaxZeroNo = MaxNumberOfZeros;
      Err = NoError;
      NumberOfZeros = 0;
      MinEpsilon = q_succ(1.0)-1.0; // 1ulp
      if (Epsilon < MinEpsilon)
         Epsilon = MinEpsilon;

      xinewton(f(Start), Start, Epsilon, 0 , zerolist, NumberOfZeros);

      if (NumberOfZeros > MaxNumberOfZeros) {
         Err = NotAllZeros;
         NumberOfZeros = MaxNumberOfZeros;
      }
   } else {
      Err = IllMaxZeroNo;
      NumberOfZeros = 0;
   }

   pointer = zerolist;
   while (pointer != NULL) {
      if (pointer->info==0) {
        VerificationStep(pointer->intval,pointer->info);
      }
      pointer = pointer->next;
   }
}

// ------------------------------------------------------------------------
// ---   function main                                                  ---
// ------------------------------------------------------------------------

int main ()
{
   interval SearchInterval;
   double Tolerance;
   int NumberOfZeros, Error;
   list* Zero;
   Zero = NULL;

   cout << "Extended-Interval-Newton method in C++ with fi_lib" << endl;
   cout << "=================================================" << endl << endl;

   cout << "Computing all zeros of the function f(x) = cosh(x) + 10 * x^2 * sin(x)^2 - 34 "
        << endl;
   cout << "Search interval (e.g. '-10 10'): " ;
   cin >> SearchInterval;
   cout << endl << "Search interval = " << SearchInterval << endl;
   cout << "Tolerance (relative) (e.g. '1e-3' or '1e-15'): ";
   cin >> Tolerance;
   cout << endl;

   AllZeros(SearchInterval, Tolerance, Zero, NumberOfZeros,
                                                    Error, MaxCount);
```

```
   if (Zero==NULL)
     cout << "Function contains no zeros in the search interval!" << endl;
   else {
     cout << "Ranges for zeros:" << endl;
     while (Zero != NULL) {
       cout << Zero->intval << endl;
       if (Zero->info)
          cout << " encloses a locally unique zero !" << endl;
       else
          cout << " may contain a zero (not verified unique)!"<< endl;
       Zero = Zero->next;
     }
   }
   cout << endl << NumberOfZeros << " interval enclosure(s)" << endl;
   if (Error) cout << endl << AllZerosErrMsg(Error) << endl;

   return 0;
}
```

Anhang E

Erweiterungen, Ausblick

E.1 Bekannte Fehler

Es sind derzeit keine Fehler bekannt (Stand: Juli 1998).

E.2 Ideen, Verbesserungsvorschläge

Areacosinus: Fehlerschranke kann evtl. verbessert werden, wenn der Teilpunkt 1.025 größer gewählt wird. Evtl. auch Überschätzung durch `eps_ari` – mit Handrechung überprüfen?
Logarithmus: Bei der Intervallfunktion sollten mögliche exakte Ergebnisse (z.B. $\log_2(2^k)$, $\log_{10}(10^k)$) getrennt behandelt werden.
Potenzfunktion: x^y zur Verfügung stellen.

Anhang F

Sonstige Hinweise

F.1 Schaubilder der Funktionen

Die Schaubilder der Funktionen, die in diese Dokumentation eingebunden sind, wurden alle mit dem Programm „Gnuplot“ erzeugt. Es wurden jeweils Bilder im eps-Format (encapsulated postscript) generiert. Verwendet wurden dazu die generell die folgenden Befehle:

```
set size 0.5,0.5
set terminal postscript eps
set output "<filename>"
```

Die Funktionen selbst wurden dann mit dem entsprechenden Plot-Befehl gezeichnet:

```
Dateiname   Plot-Befehl
-------------------------------------------------------------------------
sqr.eps     plot [x=-5:5] [0:10] x**2 title "sqr(x)"
sqrt.eps    plot [x=-0.2:7] [-1.5:4.5] sqrt(x)
exp.eps     plot [x=-5:5] [0:10] exp(x)
expm1.eps   plot [x=-5:5] [-1:9] exp(x)-1 title "expm1"
exp2.eps    plot [x=-5:5] [0:10] 2**x title "exp2(x)  "
exp10.eps   plot [x=-5:5] [0:10] 10**x title "10^x"
log1p.eps   plot [x=-1.1:6] [-3.5:3.5] log(1+x) title "ln(1+x)"
log.eps     plot [x=-0.1:7] [-3.5:3.5] log(x) title "ln(x)"
log2.eps    plot [x=-0.1:7] [-3.5:3.5] log(x)/log(2) title "log2(x)"
log10.eps   plot [x=-0.1:7] [-3.5:3.5] log10(x) title "log10(x)"
sin.eps     set size 0.5,0.25
            plot [x=-9.5:9.5] [-1.2:1.2] sin(x)
cos.eps     set size 0.5,0.25
            plot [x=-9.5:9.5] [-1.2:1.2] cos(x)
tan.eps     plot [x=-4.7:4.7] [-10:10] tan(x)
cot.eps     plot [x=-6.2:6.2] [-10:10] 1/tan(x) title "cot(x)"
asin.eps    plot [x=-1.5:1.5] asin(x) title "arcsin(x)"
acos.eps    plot [x=-1.5:1.5] [0:3.15] acos(x) title "arccos(x)"
atan.eps    plot [x=-10:10] [-5:5] atan(x) title "arctan(x)"
acot.eps    plot [x=-10:10] [-3:7] pi/2-atan(x) title "arccot(x)"
sinh.eps    plot [x=-4.9:4.9] [-4.9:4.9] sinh(x)
cosh.eps    plot [x=-4.9:4.9] [-0.9:8.9] cosh(x)
tanh.eps    set size 1,0.25
            plot [x=-4.9:4.9] [-1.2:1.2] tanh(x)
```

```
coth.eps    plot [x=-4.9:4.9] [-4.9:4.9] coth(x)
asnh.eps    plot [x=-4.9:4.9] [-4.9:4.9] log(x+sqrt(x**2+1)) title "arsinh(x)"
acsh.eps    plot [x=-0.1:9.8] [-1.9:4.9] log(x+sqrt(x**2-1)) title "arcosh(x)"
atnh.eps    plot [x=-1.4:1.4] [-3.2:3.2] 0.5*log((1+x)/(1-x)) title "artanh(x)"
acth.eps    plot [x=-4.9:4.9] 0.5*log((x+1)/(x-1)) title "arcoth(x)"
```

In das LaTeX-Dokument wurden die Schaubilder dann mit Hilfe der folgenden Befehle eingebunden:

```
\usepackage{graphicx}
\includegraphics{<filename>}
```

F.2 Unterschiede PC - Workstation

Die in PC's verwendeten Prozessoren (z.B. INTEL) verwenden intern ein 80-Bit Zahlenformat, Workstations führen die gesamte Berechnung im 64-Bit double-Format durch. Dies führt (je nach verwendetem Compiler) dazu, daß die folgenden Programmanweisungen in C unterschiedlich berechnet werden:

```
...
  a = k * konst;
  a = x - a;
  b = x - k * konst;
...
```

Nach Ausführung auf einer Workstation gilt `a=b`, auf einem PC unter Linux (Gnu-C-Compiler) gilt dies jedoch für bestimmte Werte von `x` und `konst` nicht!

F.3 Fehler in früheren Implementierungen

Die im folgenden aufgelisteten Fehler in früheren Implementierungen der Standardfunktionen wurden bei der Entwicklung der neuen Bibliothek entdeckt.

F.3.0.1 PASCAL-XSC - Linux-Version T3.01

- Große Überschätzung bei der Logarithmusfunktion zur Basis 10:

  $$\log_{10}(10) = [0,1]$$

  $$\log_{10}(100) = [1,2]$$

- Potenzfunktion

  $$[-2,-1]^{[2,3]}$$

F.3.0.2 PASCAL-XSC-Modul `mpi_ari`

- In den Kommentarzeilen des PASCAL-XSC-Moduls bei der Funktion `power` muß es im 4. Fall heißen: `x.inf < 1 < x.sup`

Falls weitere Ungereimtheiten auftreten sollten, werden diese in der Datei `error_reports.xsc` dokumentiert.

Literaturverzeichnis

[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] Alefeld, G. und Herzberger, J.: *Einführung in die Intervallarithmetik.* Bibliographisches Institut, Mannheim, 1974.

[3] *American National Standard for Information Systems – Programming Language C.* X3.159-1989.

[4] 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).

[5] Blomquist, F.: *Implementierung und Fehlerabschätzungen von PASCAL-XSC Standardfunktionen für ein dezimales Datenformat*. Universität Karlsruhe, 1992.

[6] Blomquist, F.: *Implementierung einiger spezieller mathematischer Funktionen in PASCAL-XSC*. Private Mitteilung, 1992.

[7] Bohlender, G., Grüner, K., Kaucher, E., Klatte, R., Krämer, W., Kulisch, U., Rump, S., Ullrich, Ch., Wolff von Gudenberg, J., Miranker, W.: *PASCAL-SC: A PASCAL for Contemporary Scientific Computation*. Research Report RC 9009, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1981.

[8] Bohlender, G., Rall, L., Ullrich, Ch., Wolff von Gudenberg, J.: *PASCAL-SC: A Computer Language for Scientific Computation*. Academic Press, New York, 1987.

[9] Bohlender, D., Ullrich, C.: *Standards zur Computerarithmetik.* In *Wissenschaftliches Rechnen*, Herzberg, J. (Herausgeber), Akademie Verlag, Berlin, 1995.

[10] Braune, K., Krämer, W.: *Standard Functions for Intervals with Maximum Accuracy*. 11th IMACS World Congress, Proceedings Vol. **1**, pp 167–170, 1985.

[11] Braune, K., Krämer, W.: *High-Accuracy Standard Functions for Intervals*. In M. Ruschitzka (Ed.): *Computer Systems: Performance and Simulation.* Elsevier Science Publishers, 1985.

[12] 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*, pp81–114, Teubner, Stuttgart, 1987.

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

[14] Braune, K.: *Standard Functions for Real and Complex Point and Interval Arguments with Dynamic Accuracy*. Computing Supplement **6**, pp 227–244, 1988.

[15] Braune, K.: *A-posteriori Fehlerschranken bei der Berechnung inverser Standardfunktionen mit Hilfe des Newton-Verfahrens*. ZAMM **70**, pp T579–T581, 1990.

[16] Cordes, D., Krämer, W.: *PASCAL-XSC Module for Multiple-Precision Operations and Functions*. Universität Karlsruhe, 1991.

[17] Dekker, T.J.: *Floating-Point Technique for Extending the Availible Precision.* Numerische Mathematik 18, S. 224-242, 1971.

[18] Hart, J. F. et al.: *Computer Approximations.* Wiley, New York / London / Sydney, 1968.

[19] Heck, A.: *Introduction to Maple* Springer Verlag, 1993

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

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

[22] Kernighan, B. W., Ritchie, D. M.: *Programmieren in C — Mit dem C-Reference Manual in deutscher Sprache*. Zweite Ausgabe ANSI C, Hanser-Verlag, München/Wien, 1990.

[23] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: *PASCAL–XSC — Sprachbeschreibung mit Beispielen*. Springer-Verlag, Berlin/Heidelberg/New York, 1991.

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

[25] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: *PASCAL–XSC — Language Reference with Examples (In Russian)*. MIR-Verlag, Moskau. To appear 1995.

[26] Klatte R., et. al.: *C-XSC, A C++ Class Library for Scientific Computing*, Springer 1993.

[27] Knüppel, O.: *BIAS–Basic Interval Arithmetic Subroutines*, TU Hamburg-Harburg, Bericht 93.3, 1993.

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

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

[30] Krämer, W.: *Die Berechnung von Standardfunktionen in Rechenanlagen*. In Chatterji, S. D., Kulisch, U., Laugwitz, D., Liedl, R., Purkert, W. (Eds.): *Jahrbuch Überblicke Mathematik 1992*, Vieweg, Braunschweig, 1992.

[31] Krämer, W.: *Eine portable Langzahl- und Langzahlintervallarithmetik mit Anwendungen*. ZAMM **73**, 1992.

[32] 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.

[33] Krämer, W.: *Die Berechnung von Funktionen und Konstanten in Rechenanlagen*. Habilitationsschrift, Universität Karlsruhe, 1993.

[34] Krämer, W.: *Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen*, Bericht des Instituts für Angewandte Mathematik, Universität Karlsruhe, 1996.

[35] Linnainmaa, F.: *Software for Double-Precision Floating-Point Computations*. ACM Trans. on Math. Software, Vol. **7**, No. 3, pp 272-282, 1981.

[36] Ratz, D.: *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*. Dissertation, Universität Karlsruhe, 1992.

[37] 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.

[38] 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.

[39] 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.

Folgende Arbeiten sind bisher in der Preprintreihe des IWRMM erschienen:

Nr. 93/1: G. Aumann, K. Bentz: Geometrische Stetigkeit beliebiger Ordnung zwischen Tensor-Produkt-Bézier-Flächen

Nr. 93/2: G. Alefeld, G. Mayer: A Computer Aided Existence and Uniqueness Proof for an Inverse Matrix Eigenvalue Problem

Nr. 93/3: B. Weber: Symbolische Programmierung in der Mehrkörperdynamik

Nr. 93/4: R. Rihm: Über Einschließungsverfahren für gewöhnliche Anfangswertprobleme und ihre Anwendung auf Differentialgleichungen mit unstetiger rechter Seite

Nr. 93/5: J. Wittenburg: Explizite Lösungen für lineare Gleichungssysteme mit tridiagonalen Koeffizientenmatrizen. Anwendungen in der Mechanik

Nr. 93/6: N. Henze, B. Klar: Goodness–of–Fit Testing for a Space–Time Model for Daily Rainfall

Nr. 93/7: K. Schweizerhof, J. Riccius, M. Baumann: Verbesserung von Finite Element Berechnungen durch Adaptivität und Netzglättung am Beispiel ebener und gekrümmter Flächentragwerke

Nr. 93/8: G. Starke: Subspace Orthogonalization for Substructuring Preconditioners for Nonselfadjoint Elliptic Problems

Nr. 93/9: N. Henze, B. Klar: Empirical Distribution Function Tests for the Generalized Poisson Model

Nr. 94/1: G. Aumann: Geometric Continuity of Parametric Curves and Surfaces

Nr. 94/2: T. Dehn, M. Eiermann, K. Giebermann, V. Sperling: Structured Sparse Matrix-Vector Multiplication on Massively Parallel Architectures

Nr. 94/3: W. Krämer: Bericht über die Begutachtung des IWRMM im Dezember 1993

Nr. 95/1: L. Kobbelt: Interpolatory Refinement is Low Pass Filtering

Nr. 95/2: M. Paluszny, H. Prautzsch, M. Schäfer: Corner cutting and interpolatory refinement

Nr. 95/3: B. Klar: Analysis of and Goodness of Fit Testing for a Flexible Discrete Time Failure Model

Nr. 95/4: P. Vielsack: Regularisierung von Haftkräften bei Coulombscher Reibung

Nr. 95/5: P. Vielsack, M. Storz: Bifurcation of Motion in a Technical System with Stick-Slip and Impact

Nr. 95/6: M. Brühl: A Curve Tracing Algorithm for Computing the Pseudospectrum

Nr. 95/7: J. Riccius, K. Schweizerhof, M. Baumann: On the treatment of shell intersections in adaptive finite element analysis and combination with mesh smoothing

Nr. 96/1: M. Dormanns, H.-U. Heiß: Nutzung von Asynchronität bei iterativen Gleichungslösern auf Multirechnersystemen

Nr. 96/2: P. Vielsack, J. Kirillowa: Nichteindeutigkeit der Bewegungen eines Reibschwingers mit Selbsterregung

Nr. 96/3: L. Kobbelt, T. Hesse, H. Prautzsch, K. Schweizerhof: Diskrete Freiformflächenerzeugung für FEM-Anwendungen

Nr. 96/4: M. Brühl, M. Hanke, H. Wanzki: Ein Rekonstruktionsverfahren für die elektrische Impedanztomographie

Nr. 96/5 : W. Hofschuster, W. Krämer: Ein rechnergestützter Fehlerkalkül mit Anwendung auf ein genaues Tabellenverfahren

Nr. 96/6: W. Niethammer, W. Krämer (Herausgeber): Tagungsband zum Workshop „Wissenschaftliches Rechnen in den Ingenieurwissenschaften“

Nr. 96/7: G. Freimann: FAS-Verfahren zur Lösung strukturmechanischer Probleme

Nr. 97/1: P. Vielsack, A. Hartung: Orbitale Stabilität von Bewegungen mit Pausen bei Einwirkung permanenter Störungen

Nr. 97/2: J. G. Schmidt, G. Starke: Coarse Space Orthogonalization for Indefinite Linear Systems of Equations Arising in Geometrically Nonlinear Elasticity

Nr. 97/3: F. Blomquist, W. Krämer: Algorithmen mit garantierten Fehlerschranken für die Fehler- und die komplementäre Fehlerfunktion

Nr. 98/1: S. Doll, R. Hauptmann, K. Schweizerhof, C. Freischläger: Selective Reduced Integration and Volumetric Locking in Finite Deformation Elastoviscoplasticity

Nr. 98/2: P. Vielsack, H. Kammerer: Finite Element Formulierung nichtglatter Schwingungen eines Balkens mit Reibglied

Nr. 98/3: H. Prautzsch: How Smooth are Subdividable Surfaces at Extraordinary Points?

Nr. 98/4: W. Wu, W. Rodi, Th. Wenka: 3D Numerical Modeling of Flow and Sediment Transport in Open Channels

Nr. 98/5: A. Bantle, W. Krämer: Ein Kalkül für verläßliche absolute und relative Fehlerabschätzungen

Nr. 98/6 R. Hauptmann, K. Schweizerhof, S. Doll: Extension of the "solid-shell" concept for application to large elastic and large elastoplastic deformations

Nr. 98/7 W. Hofschuster, W. Krämer: FI_LIB, eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-Double-Format

Weitere Arbeiten sind in Vorbereitung.